UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN

PROYECTO FIN DE CARRERA

Diseño, implementación y evaluación de un protocolo de tiempo para redes de sensores inalámbricas.

AUTOR: María Illera Bermejo
TUTOR: M. Soledad Escolar Díaz

Septiembre, 2012

“El verdadero progreso es el que pone la tecnología al alcance de todos”
Henry Ford

AGRADECIMIENTOS

Dedicado a mis padres y a mi hermana por confiar en mí.
A Julio por apoyarme.
A Soledad por sus enseñanzas.

RESUMEN

Este Proyecto Fin de Carrera se encarga de desarrollar un protocolo de tiempo para redes de sensores inalámbricas (Wireless Sensor Networks, WSN).

Una WSN es una red que consiste en una serie de dispositivos autónomos (nodos sensores) distribuidos en un área susceptible de estudio y que cuentan con restricciones en cuanto a capacidad de cómputo, comunicación y energía se refiere. Estos dispositivos usan sensores para supervisar y monitorizar condiciones físicas o ambientales, tales como la temperatura, humedad, sonido, presión o agentes contaminadores, entre otros.

Estos dispositivos hacen posible medir variables en sitios inaccesibles y en cualquier condición, es decir llegan donde el hombre no puede, pero son limitados tanto en tiempo de vida como en su propio hardware. Una de sus grandes limitaciones es que no comparten ningún reloj que les permita tener una referencia horaria, por lo que operaciones que deben realizarse de manera sincronizada entre los nodos (por ejemplo, activación de la radio) pueden verse afectadas negativamente.

En este estudio se pretende desarrollar un protocolo de comunicación que permita tener todos los nodos de una red sincronizados.

El sistema construido estará formado por los nodos sensores que muestrean la información y la transmiten al nodo base o gateway. Este hace de puente entre la red y el centro de datos (PC), donde se visualizan los paquetes recibidos para su posterior estudio.

.

ABSTRACT

This Project takes charge of the developing of a time protocol for wireless sensor networks (WSN). A WSN is a network that consists of a set of autonomous devices (sensor nodes) distributed over an area capable of sensing the environment and that possess restrictions as computation, communication and energy.

These devices use sensors to supervise and monitor physical or environmental conditions such as the temperature, humidity, sound, pressure or contaminating agents, among others.

These devices make possible to measure variables in inaccessible sites and in any condition. However, these devices have limited lifetime and strong hardware restrictions. One of the main limitations is that they do not have a clock that allows them to have an hourly reference, and then synchronized operations (like radio activation) can be negatively affected.

This study tries to develop a protocol of communication that allows the nodes of a network be synchronized.

The system was built by mean of a set of nodes that get samples from the environment and delivery them to the base station or gateway, which is the bridge between the network and the data center (PC) where the packages can be visualized and processed for further study.

ÍNDICE GENERAL

Índice de figuras

Índice de tablas

1. Introducción

Uno de los objetivos de todo ser vivo es conocer. Necesitamos tener una idea clara y concreta de las cosas que nos rodean, pero esto no siempre es posible.

El ser humano es limitado y no siempre puede tener una conciencia de los cambios de su entorno. Por ejemplo, no podemos contabilizar por nosotros mismos el volumen de agua que cae en un metro cuadrado durante una tormenta; tampoco podemos conocer la velocidad con la que se corrompe una madera o un metal.

Las ciencias y las ingenierías conocen este problema y han desarrollado técnicas y herramientas para reducir estas limitaciones. Una de estas tecnologías son las redes de sensores inalámbricas.

En los años 90, las redes de sensores han cambiado la forma de intercambiar información. Los últimos avances tecnológicos han permitido el desarrollo de unos dispositivos distribuidos, pequeños, de bajo coste y consumo llamados nodos sensores o *motes*. Estos dispositivos son capaces tanto de procesar información localmente como de comunicarse de forma inalámbrica.

Una red de sensores o Wireless Sensor Networks (WSN) [1] está formada por un conjunto de motes, colaborando para llevar a cabo un fin común, transmitiéndose datos y mediciones; esto permite que se pueda observar y analizar un fenómeno de forma remota.

Una red de sensores puede llegar donde el hombre no puede: por ejemplo, puede medir el clima de los lugares más inhóspitos y en parte, ser nuestros ojos.

1.1 Motivación

El propósito del proyecto es establecer un protocolo de tiempo que permita la sincronización de los nodos.

Las redes de sensores actuales pueden realizar mediciones en medios hostiles, pueden ayudar a detectar una tormenta, pueden analizar la corrosión de las vías de un tren; pero no pueden en general establecer un tiempo en sus mediciones; es decir, podemos realizar mediciones de gran importancia pero no podemos, en principio, establecer que medición fue anterior o posterior, puesto que dichas redes no disponen de un reloj absoluto.

Por ejemplo, si tenemos cinco nodos monitorizando una habitación y se produce un incendio, todos los nodos van a detectar con una diferencia de centésimas de segundo el humo, y van a lanzar los paquetes a la estación base, pero no podemos saber con seguridad qué nodo fue el primero en detectar el humo, y con casi toda seguridad, establecer en qué lugar de la habitación empezó el fuego. Esto es sólo un ejemplo, pero conocer el tiempo en el que se realizó una medición es de gran importancia en todas las áreas.

Como se mencionó anteriormente, las motes no tienen un reloj absoluto que les permita conocer la hora. Por tanto, para poder sincronizar los nodos de la red es necesario enviar el tiempo absoluto a cada uno de los nodos de la red desde un sistema externo.

La sincronización de los nodos es un problema crítico. Unas de las aplicaciones donde la sincronización de nodos tiene gran importancia es en los protocolos de acceso al medio o MAC. Los protocolos MAC se encargan en líneas generales de repartir y sincronizar el uso del medio

compartido. Por lo tanto, deben garantizar que el medio esté libre si alguno de los dispositivos que lo comparte necesita transmitir algún mensaje. Además debe evitar las colisiones debidas a la transmisión simultánea.

Los protocolos MAC en las redes de sensores inalámbricas van a permitir minimizar la energía usada en la radio ya que deben reducir el tiempo de escucha del canal. Esto implica que el tiempo para poder recibir se ve limitado, y los nodos deben enviar en tiempo para que no se produzcan pérdidas de paquetes.

La principal idea de los protocolos MAC es ahorrar energía encendiendo y apagando periódicamente las radios de los nodos, apareciendo el término ciclo de trabajo. En lugar de tener las radios siempre escuchando posibles transmisiones los nodos periódicamente activarán su radio para escuchar paquetes en el canal y posteriormente volverán a apagar la radio con el propósito de ahorrar energía.

En la Figura 1 se muestra la idea básica de los protocolos MAC de nodos sensores. Cuando los nodos reciben un mensaje dirigido a ellos entran en la fase de escucha. El resto del tiempo están durmiendo, de esta manera gracias a los protocolos MAC los nodos sensores ahorrarán energía. Los nodos también pueden dormir cuando se detecta una transmisión que se dirige a otro nodo.

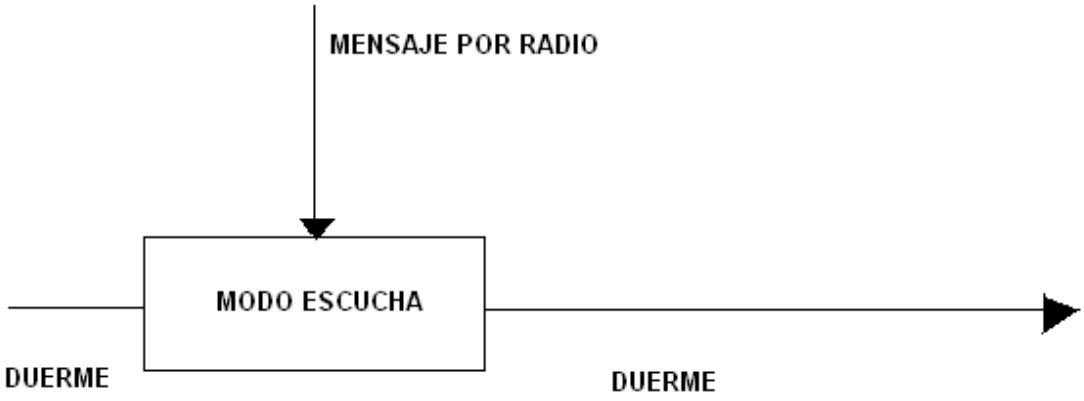

Figura 1: Ciclo de trabajo de la radio

1.2 Alcance

Como se ha comentado en el apartado anterior las redes de sensores inalámbricas o Wireless Sensor Networks (WSN) actualmente no tienen ningún mecanismo de sincronización por lo que no se puede establecer una relación temporal entre los datos que se transmiten dentro de la red. Por este motivo este proyecto pretende desarrollar un protocolo de tiempo que sincronice todos los nodos de la red.

En la siguiente ecuación se pueden observar los dispositivos que forman una WSN estándar:

WSN= Nodos sensores + Gateway + Estación Base

El primer actor que aparece en la ecuación es el *nodo sensor*. Esta variable se corresponde con los nodos inalámbricos que están distribuidos por la red y cuya labor es realizar mediciones con

los sensores que tienen integrados. Estos nodos o motes son pequeños dispositivos inalámbricos con un tiempo de vida muy limitado, que pueden variar de localización. Se encargan de realizar mediciones y enviárselas al gateway.

El segundo actor de la ecuación es el *gateway*, un nodo especial que no se encarga de realizar mediciones sino de entregar los paquetes generados en la red a un dispositivo externo o estación base. Este nodo está equipado de un puerto para la comunicación con el ordenador, que realiza la función de estación base. Los gateway reciben los paquetes que le envían los nodos sensores a través de su radio y envía dichos paquetes a la estación base.

La *estación base* es el último elemento de nuestra ecuación y se encarga de almacenar los datos que recibe desde el gateway para que los científicos puedan estudiar las mediaciones enviadas desde los nodos sensores y establecer unas conclusiones sobre el medio. Como norma general las estaciones base son ordenadores convencionales.

En este proyecto se va a desarrollar un protocolo de comunicación que permita sincronizar los nodos, es decir, va a posibilitar añadir el factor tiempo a los datos, y de esta manera se va a poder establecer en qué instante se han realizado las mediciones o se han transferido paquetes. Añadir un protocolo de sincronización a las redes inalámbricas permite poder analizar con exactitud los eventos estudiados por los nodos y poder establecer un orden en las mediciones y así incluso, poder establecer secuencias en los hechos.

La ecuación básica de una WSN se modifica en este proyecto añadiendo el protocolo de sincronización de los nodos y además, separando los nodos con función de enrutador en dos tipos: el nodo conectado a la estación base o *gateway maestro* y, el nodo o los nodos que se encargarán de enrutar el tiempo de la red a los nodos sensores pero que no están conectados a ningún equipo, los *gateways esclavos*.

La nueva ecuación que se presenta sería como sigue:

WSN= Nodos sensores + Gateway Maestro + Gateway Esclavo + Estación Base

Para poder realizar esta sincronización se ha de conseguir un reloj externo a la red que obtenga el tiempo y se propague a través de los nodos accesibles. Siempre que un nodo pueda recibir un mensaje a través de su radio de un gateway ya sea maestro o esclavo va a estar sincronizado.

En el caso que un nuevo nodo se incorporase a la red y fuese accesible para los nodos enrutadores, en la siguiente transmisión de los datos de tiempo se va a poder sincronizar.

1.3 Objetivos

El objetivo de este Proyecto Fin de Carrera es desarrollar un protocolo de tiempo y sincronización que pueda ser utilizado dentro de la red y permita tener en todo momento todos los nodos de la red sincronizados.

De esta manera las mediciones que realizan los nodos sensores podrán llevar asociado un instante de tiempo, y se podrá establecer una evolución temporal. Como se comentó anteriormente, es muy importante que las redes donde se intercambia un volumen grande de datos tengan una referencia temporal ya que permite establecer un orden cronológico en los acontecimientos y mediciones.

Se desea permitir que los nodos estén sincronizados sin necesidad de tener que consultar a una estación base cada vez que tengamos que transmitir.

En este proyecto se utilizarán dos tecnologías/soluciones para llevar a cabo este propósito: por un lado están las redes de sensores inalámbricas que permiten obtener mediciones de forma remota y sin necesidad de cables; por otro lado se tienen los protocolos de tiempo o Network Time Protocol (NTP) [2], que permiten establecer una referencia horaria en redes, principalmente redes de ordenadores.

Los protocolos de tiempo actuales utilizan un dispositivo como referencia horaria. Este dispositivo toma el nombre de servidor y propaga la hora a todos los dispositivos conectados a la red que serán denominados clientes. Esta comunicación se realiza a través de conexiones físicas y es el cliente el que va a solicitar al servidor la hora de sincronización; es decir, la comunicación entre el servidor y el cliente se realiza bajo demanda.

En las redes inalámbricas no se dispone de un medio de comunicación guiado así que se ha de adaptar la estructura del protocolo para poder transmitir a través de la radio de los nodos y con el menor consumo posible de energía. Es fundamental minimizar lo máximo posible el uso de la radio de los nodos para alargar así su tiempo de vida.

En este proyecto se pretenden unir ambas tecnologías y generar una red de sensores inalámbrica que pueda usar un protocolo de tiempo, y de esta manera ampliar la aplicación de los protocolos NTP actuales y mejorar el valor de los datos analizados de las redes WSN.

Las redes de sensores inalámbricos se pueden aplicar en un gran número de áreas como en domótica, en el estudio de entornos naturales o en estudios meteorológicos. Se pueden estudiar un gran número de fenómenos, pero estás redes no pueden establecer un tiempo en esas mediciones. Por ese motivo en este proyecto se pretende realizar un avance en las redes de sensores inalámbricas, añadiendo la función de sincronización.

1.4 Glosario de términos

A

API o interfaz de programación de aplicaciones: (del inglés Application Programming Interface) conjunto de funciones y procedimientos que ofrece cierta funcionalidad para ser utilizada por otro software como una capa de abstracción.

B

Baudios: Unidad de medida, usada en telecomunicaciones, que representa el número de símbolos transmitidos por segundo en un red analógica.

Big-Endian: Formato de almacenamiento de datos de más de un byte en una dirección de memoria donde el byte de mayor peso se almacena en la dirección más baja de memoria y el byte de menor peso en la dirección más alta.

Broadcast: Se denomina *broadcast* a la transmisión de un paquete que será recibido por todos los dispositivos en una red.

C

Cliente-servidor: Arquitectura de red en la que un sistema nominado *cliente* realiza peticiones a otro sistema denominado *servidor* que le da respuestas. Los clientes son los sistemas que inician la comunicación lanzando una o varias solicitudes a uno o varios servidores, una vez hecho esto, se queda esperando la respuesta. Por otro lado, el servidor está permanentemente esperando que lleguen peticiones, cuando recibe una la procesa y envía la respuesta al cliente que le consultó.

Cygwin: Conjunto de herramientas desarrolladas por *Cygnus Solutions* que permite simular el sistema operativo Unix en el sistema operativo Windows.

D

Domótica: Se entiende por domótica al conjunto de sistemas capaces de automatizar una vivienda, aportando servicios de gestión energética, seguridad, bienestar y comunicación, y que pueden estar integrados por medio de redes interiores y exteriores de comunicación, cableadas o inalámbricas, y cuyo control goza de cierta ubicuidad, desde dentro y fuera del hogar. Se podría definir como la integración de la tecnología en el diseño inteligente de un recinto.

E

Estación Base: Se llamará así al ordenador personal al que se conectarán los motes de la red WSN para transmitir los datos obtenidos. Este ordenador almacenará estos datos para un futuro análisis por parte de un especialista.

Enrutador: Dispositivo de hardware que permite interconectar redes de ordenadores que opera en la capa de red. Permite asegurar el enrutamiento de paquetes entre redes y determinar la ruta que debe tomar el paquete de datos.

F

Framework: Estructura conceptual y tecnológica de soporte definida normalmente con artefactos o módulos de software. Normalmente suele incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros programas, para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

G

Gateway: Pasarela que permite la interconexión entre la red de sensores y una estación base o computador y que suele disponer de una conexión directa a una red TCP/IP.

K

Kernel: En informática se denomina kernel o núcleo al software que actúa como el sistema operativa de un ordenador. Las funciones básicas del kernel es facilitar a los distintos programas un acceso seguro al hardware del equipo, además de gestionar recursos.

L

Latencia: En redes informáticas de datos se denomina latencia a la suma de retardos temporales dentro de una red. Un retardo es producido por la demora en la propagación y transmisión de paquetes dentro de la red.

Little-Endian: Formato de almacenamiento de datos de más de un byte en una dirección de memoria donde el byte de menor peso se almacena en la dirección más baja de memoria y el byte de mayor peso en la más alta.

Low Power Listening: Los protocolos Low Power Listening o protocolos de bajo consumo de escucha (LPL) permiten ahorran energía en la transmisión de paquetes en las redes de sensores inalámbricas. Cuando un nodo no esté procesando un mensaje entrará en un estado de hibernación apagando su radio, cuando el nodo detecte actividad en la red volverá a activar su radio y tomará el control de la transmisión.

M

Mote: Una mote o nodo sensor es un dispositivo inalámbrico equipado con sensores que permite realizar mediciones y transmitirlas, que colabora con otros para formar una WSN o red de sensores.

N

NesC: es un lenguaje de programación que se utiliza para crear aplicaciones que serán ejecutadas en sensores que ejecuten el sistema operativo de TinyOs.

NTP: *Network Time Protocol* o protocolo de tiempo, es un protocolo de comunicación que permite sincronizar los relojes de un sistema informático a través de enrutamiento de paquetes en redes con latencia variable. NTP fue diseñado originalmente por Dave Mills de la Universidad de Delaware y es uno de los protocolos más antiguos de internet.

P

Plugin o plug-in: (del inglés "enchufable") complemento de una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy especifica.

Protocolo de comunicación: Un protocolo de comunicación es un conjunto de reglas usadas por computadoras para comunicarse unas con otras a través de una red.

Protocolos MAC: (del inglés *Medium Access Control*, o control de acceso al medio) son un conjunto de algoritmos y métodos de comprobación encargados de regular el uso del medio físico por los distintos dispositivos que lo comparten.

R

Radio: Medio no guiado que utiliza antenas lineales para comunicar mensajes a través de ondas unidireccionales.

Red: Una red en el campo de la informática es un conjunto de equipos (ordenadores y/o dispositivos) conectados por medio de cables, señales, ondas o cualquier otro método de transporte de datos, que comparten información, recursos y servicios. En toda red van a existir tres componentes: el sistema emisor que es quien empieza la comunicación, el sistema receptor que es quien recibe el mensaje del emisor, y el canal que es el medio por donde va a viajar el paquete.

S

Sensor: Dispositivo cuyo fin es recibir estímulos y transformarlos en información, permitiendo así medir fenómenos físicos.

T

TCP/IP: Transmisión Control Protocol/Internet Protocol es el protocolo estándar de comunicaciones en red utilizado para conectar sistemas informáticos a través de Internet.

TinyOS: Es un sistema operativo de código abierto basado en componentes para redes de sensores inalámbricas. Está escrito en el lenguaje de programación nesC como un conjunto de tareas y procesos que colaboran entre sí.

Transistor o radio: El transistor es un pequeño dispositivo electrónico semiconductor que cumple diversas funciones, como pueden ser entre otras: hacer de amplificador, de oscilador o de conmutador.

U

USB: El *Universal Serial Bus* (USB) o Conductor Universal en Serie (CUS), es un puerto que sirve para conectar periféricos a un ordenador. Fue creado en 1996 por siete empresas: IBM, Intel, Northern Telecom, Compaq, Microsoft, Digital Equipment Corporation y NEC.

UART: (Del inglés *Universal Asynchronous Receiver-Transmitter*). El controlador del UART es el componente que permite manejar las interrupciones de los dispositivos conectados a un puerto serie de un equipo. Permite convertir los datos que se están transmitiendo por el puerto de un formato paralelo, a datos en formato serie, para que puedan ser transmitidos a través de los puertos y viceversa.

W

WSN: *Wireless Sensor Network* o Red de Sensores Inalámbrica. Red compuesta por un conjunto de sensores cuyo objetivo es la adquisición y tratamiento de datos con múltiples aplicaciones en distintos campos tales como entornos industriales, domótica, entornos militares, detección ambiental. Estas redes se caracterizan por su facilidad de despliegue y por ser auto configurables.

Z

ZigBee: Conjunto de protocolos de comunicación de radios digitales de alto nivel basado en el estándar IEEE 802.15.4.

2. Estado de la cuestión

En este capítulo se presenta el estado de la cuestión en Wireless Sensor Networks. Se describen los aspectos más relevantes de la tecnología hardware y software y se buscan algunos trabajos similares al propuesto como objetivo en este proyecto.

2.1 Wireless Sensor Networks (WSN)

En este capítulo se van a describir las redes de sensores inalámbricas, sus funciones y sus principales características.

2.1.1 Definición y características

Como se ha explicado anteriormente, en la última década se ha realizado la mayor parte de los avances tecnológicos en el desarrollo de los nodos sensores o *motes*. Estos dispositivos diminutos, y de bajo consumo, son capaces tanto de procesar información localmente como de comunicarse con otros dispositivos de forma inalámbrica.

Las redes de sensores inalámbricos, del inglés Wireless Sensor Networks (WSN), consisten en multitud de dispositivos autónomos que intercambian información entre sí sin necesidad de cables y mediante un protocolo de comunicación pre-establecido. En definitiva, es una red de pequeños ordenadores independientes que colaboran en una tarea común y que se comunican entre sí para intercambiar información.

Estos nodos sensores se encargan de monitorizar las condiciones físicas y ambientales, tales como la temperatura, movimiento, sonido o presión en diferentes lugares, entre otros. El desarrollo de redes de sensores inalámbricos fue originalmente ideado para aplicaciones militares, tales como la vigilancia del campo de batalla. Sin embargo, se utilizan ahora en muchas áreas de aplicación civil.

Algunas de las características únicas de una WSN son las siguientes:

- **Topología dinámica**: En una red de sensores la topología siempre es cambiante ya que los nodos pueden cambiar de localización, por lo que una WSN tiene que adaptarse a la movilidad de los nodos.

- **Heterogeneidad**: una WSN debe admitir dentro de la red nodos de distinto tipo sin que esto influya en el funcionamiento de la red.

- **Variabilidad del canal**: El canal radio es muy variable y existen una serie de fenómenos que pueden alterarle. Por ejemplo la atenuación o pérdida de la señal, o interferencias que produzcan errores en los datos.

- **Ad-hoc:** Una red de sensores no tiene la necesidad de una infraestructura de red para poder operar ya que sus nodos pueden actuar de emisores, receptores o enrutadores de la información.

- **Tolerancia a errores**: Un dispositivo sensor dentro de una WSN tiene que ser capaz de seguir funcionando a pesar de tener errores en el sistema propio.

- **Comunicaciones *broadcast***: En una red de sensores es posible enviar mensajes a un nodo específico o a todos los nodos alcanzables dentro del rango de la radio.

- **Consumo energético**: Es uno de los factores más importantes de las redes de sensores, ya que disponen de energía limitada. Un nodo sensor debe contar con un procesador y una radio de consumo ultra bajo para alargar la vida del nodo.

- **Fiabilidad**: Una WSN tiene que tener un número elevado de nodos sensores para poder obtener múltiples datos y garantizar así una alta fiabilidad.

En la Figura 2 se muestra la estructura básica de una red de sensores inalámbricos. En esta figura se pueden observar los tres actores de la red previamente definidos: la estación base donde se analizan los datos a través de programas informáticos; el gateway conectado a través de un puerto Ethernet a la estación base; y los nodos sensores que se comunican a través de la radio con el gateway para transferir las mediciones.

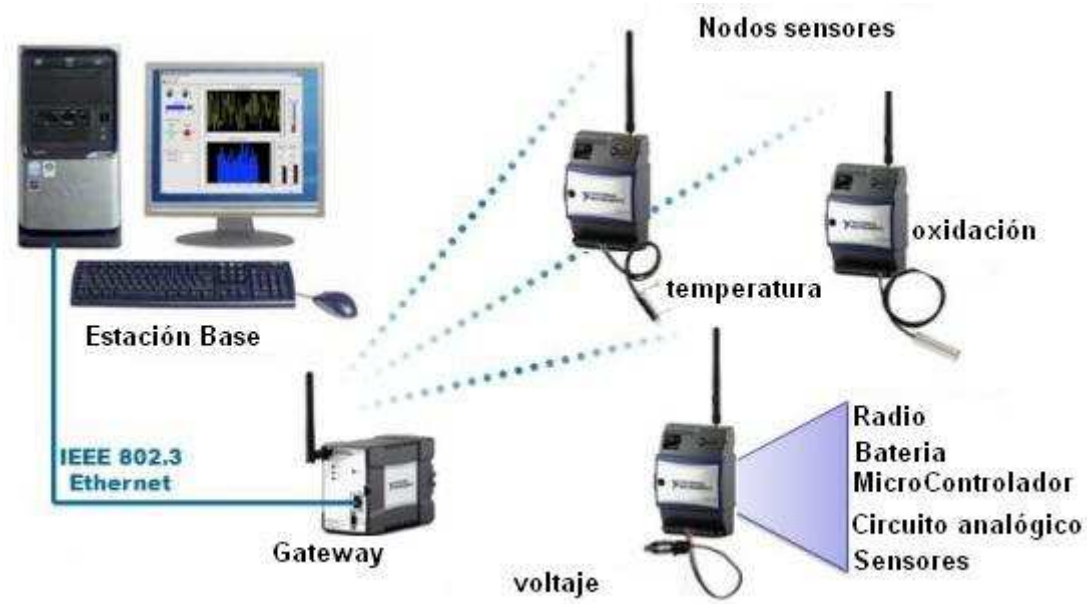

Figura 2: Arquitectura básica de una WSN

Además en la imagen se pueden observar los principales componentes de los nodos sensores: radio, batería, microcontrolador, circuito analógico y sensores. Todos estos componentes se van a estudiar en los sucesivos capítulos.

Un nodo integra varios sensores que le permiten medir distintos fenómenos medioambientales. En la red que se muestra en la figura podemos observar que uno de los nodos tiene integrado un sensor que mide la temperatura, el otro mide el voltaje y el último mide la oxidación.

2.1.2 Aplicaciones en la vida real

La primera aplicación que se dio a las redes de sensores inalámbricos fue en el ámbito militar, para permitir la identificación y seguimiento de tropas o vehículos militares, así como la detección de armas químicas y biológicas.

Actualmente gracias a los avances en las redes de sensores existen muchas más aplicaciones y se pueden encontrar en gran número de sistemas y dispositivos electrónicos. En los últimos años han surgido nuevos proyectos impulsados por varios laboratorios de investigación y multinacionales como Intel [3]. Estos avances están haciendo posible aplicar las redes de sensores a nuevas áreas de la información.

La agricultura constituye una de las áreas donde se prevé que pueda implantarse con mayor rapidez. Las redes de sensores favorecen una reducción en el consumo de agua y pesticidas, además pueden alertar sobre la llegada de heladas. Entre las aplicaciones más interesantes se encuentra el control de plagas y enfermedades, ya que a través de los nodos se monitorizan parámetros como la humedad de las hojas y la temperatura, lo cual permite adelantarse a enfermedades y situaciones adversas pudiendo realizar acciones correctivas.

En este campo se ha desarrollado el proyecto Wisewine [4] en el estado Oregón de los Estados Unidos que consiste en monitorizar la humedad del suelo, la temperatura y el nivel de luz de las grandes extensiones de viñedos. Para ello se despliegan nodos sensores con una separación de 20 metros por toda la superficie a analizar.

Las aplicaciones sociales y sanitarias permiten realizar el seguimiento de los pacientes sin interferir en su privacidad y evitando tener un cuidador permanentemente con el enfermo. Un ejemplo de ello es el proyecto CodeBlue [5], desarrollado en la Universidad de Harvard [6]. En este caso se han implementado distintos tipos de sensores para la monitorización de parámetros vitales: tasa de latidos del corazón, concentración de oxígeno en sangre, datos EKG de electrocardiograma, etc. Esta información se transmite de forma inalámbrica a una PDA u ordenador portátil para su procesamiento por parte de los médicos.

Las aplicaciones medioambientales permiten conservar parques naturales y áreas de grandes dimensiones o de difícil acceso. Estas zonas suelen tener gran variedad de especies animales y vegetales. Los sensores, de pequeño tamaño, pueden disimularse con el entorno, procesando los datos de diversos parámetros ecológicos como el crecimiento de los árboles o arbustos, el desplazamiento de las especies y los cambios climáticos; y transmitiendo la información de forma inalámbrica hasta un centro de control, para su análisis por parte de los guardias forestales. De este modo, se evita en la medida de lo posible la circulación de personas y vehículos.

Un ejemplo de esta aplicación es el realizado en Great Duck Island [7] en el estado de Maine en los Estados Unidos, este proyecto permite estudiar las especies de pájaros que habitan en la isla monitorizando la temperatura, humedad y luminosidad para poder estudiar los ciclos reproductivos de las aves. La instalación de sensores en los nidos permite el estudio sin presencia humana evitando así cualquier alteración en el comportamiento de las aves.

En la Figura 3 se puede ver uno de los nodos sensores que se utilizan en el proyecto Great Duck Island, se puede apreciar un recubrimiento para evitar que se estropeen los circuitos por las agresiones externas.

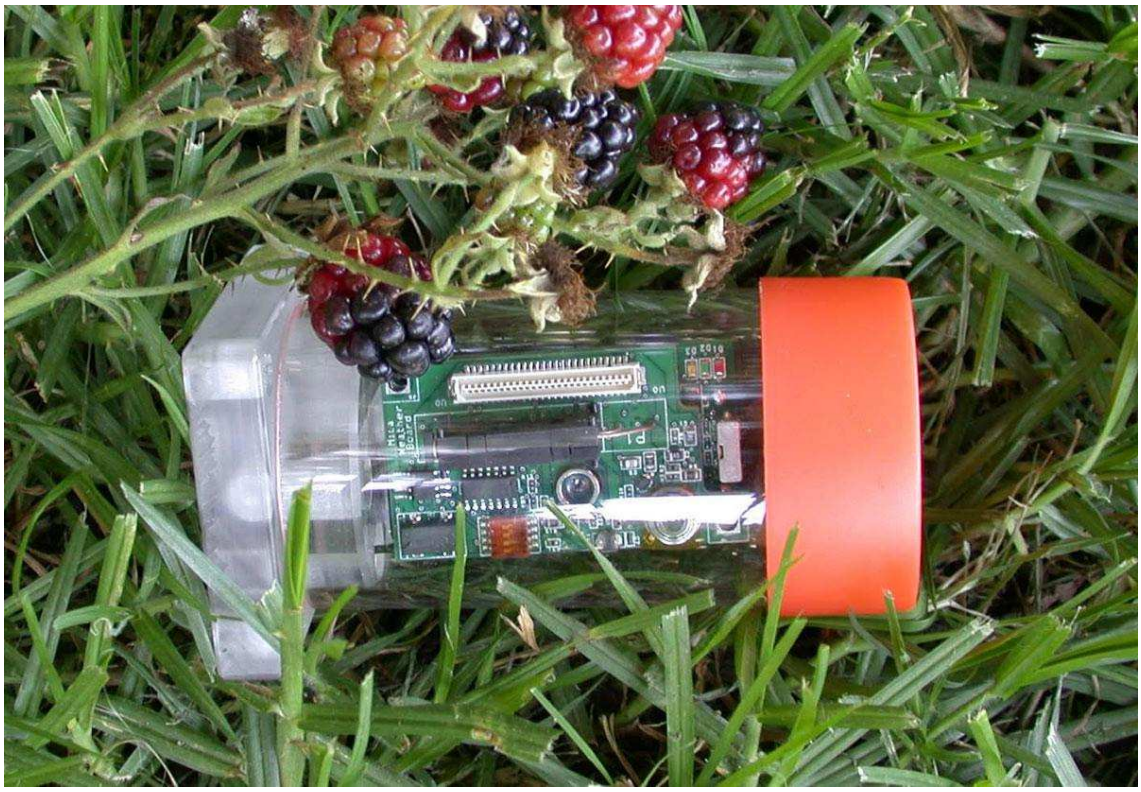

Figura 3: Un nodo Mica utilizado en el proyecto de WSN en Great Duck Island

El proyecto ARGOS [8] consiste en desplegar sensores en los océanos para medir las condiciones de temperatura y salinidad del agua, y transmite estos datos a un satélite para su estudio por parte de los científicos marinos.

La aplicación social de las redes inalámbricas se está ampliando rápidamente. Actualmente se están utilizando en los parking para la localización de plazas libres, en guarderías y residencias para la vigilancia de niños y personas mayores mediante videocámaras, en los hogares para garantizar la seguridad, en empresas para la monitorización de redes de ordenadores, además de en áreas de protección civil para la observación de costas.

La Figura 4 describe el uso de las redes de sensores inalámbricos en el área de la domótica para la detección de incendios. Cuando el nodo inalámbrico detecta el aumento de temperatura y el humo envía una alerta a una central que avisa a los equipos de emergencia.

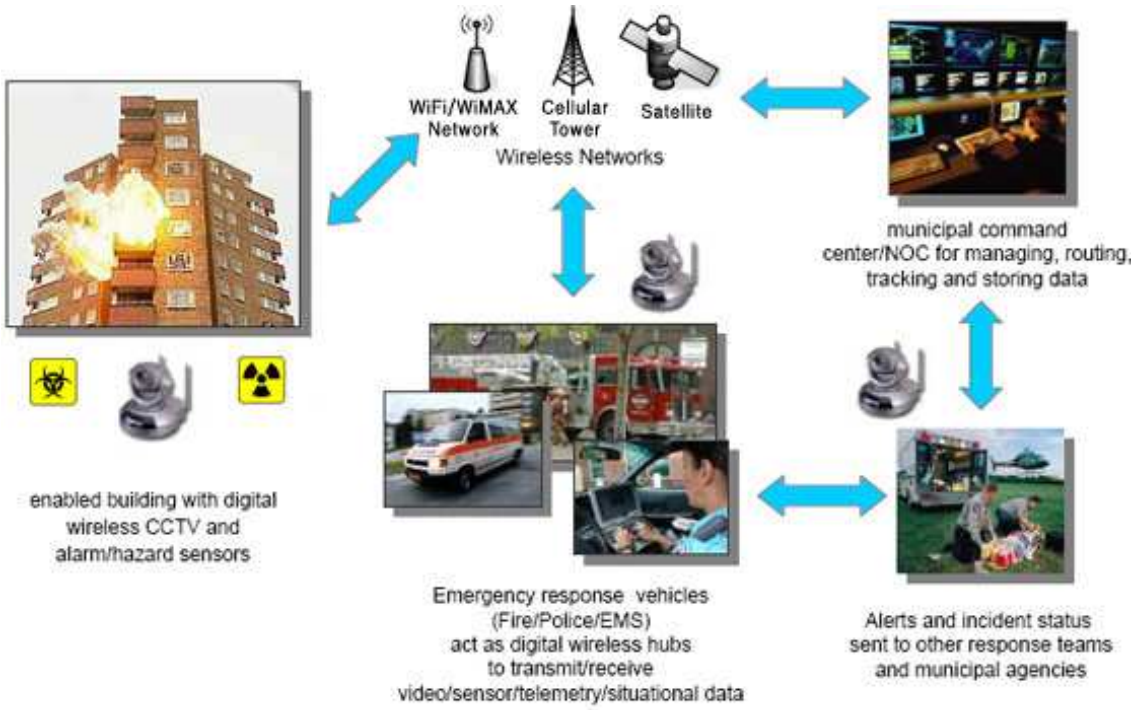

Figura 4: Uso de WSN en edificios

Cada vez con más frecuencia las redes de sensores inalámbricas están apareciendo en los entornos industriales controlando los sistemas de calidad, supervisando el estado de las cadenas de producción, e incluso se usan para garantizar la seguridad en el recinto.

Se estima que las redes de sensores ampliarán enormemente sus aplicaciones en los próximos años, incluso que pueden ser usadas como complemento a las cámaras de vigilancia de tráfico.

A continuación se presenta un listado de varias compañías que fabrican los nodos y gateways, así como el diseño e implementación de aplicaciones embebidas:

- MEMSIC [9] desarrolla circuitos integrados, sistemas electrónicos y sensores de bajo consumo.

- XSILOGY Solutions [10] es una compañía que provee WSN para las siguientes aplicaciones comerciales: organización de inventario de tanques, sistemas de distribución de flujos, edificios comerciales, monitorización medioambiental, defensa del hogar, etc.

- ENSCO [11] investiga con WSNs para aplicaciones meteorológicas.

- EMBER [12] provee soluciones con WSN para automatización industrial, defensa y edificios inteligentes.

- H9009 [13] Wireless SensorNet System (TM), el primer sistema de enrutamiento de malla inalámbrico para sensores, desarrollado por la compañía Snsicast systems. Sus aplicaciones van desde la electricidad a la seguridad del hogar

- SOFTLINX [14] desarrolla productos de seguridad perimetral basada en sensores.

- XYZ [15]: Integra redes de sensores inalámbricas para el control de entornos en el interior de edificios.

- Japan's Omron Corporation [16] ha elaborado una red de sensores para naves de carga que provee un sistema de seguridad en los puertos.

2.2. Tecnología de hardware

En esta sección se va a explicar detalladamente las diferentes tecnologías de hardware que se van a utilizar para desarrollar el protocolo de tiempo que se propone en este Proyecto Fin de Carrera.

2.2.1 Nodos sensores

Un nodo sensor o *mote* es un dispositivo autónomo que forma parte de una red de sensores inalámbricos. Los nodos son capaces de realizar la recolección de información sensorial, almacenarla temporalmente y transmitirla a otros nodos conectados en la red. Por sí sólo un sensor es capaz de procesar una limitada cantidad de datos pero cuando se coordina la información en una red con un número importante de nodos, éstos tienen la habilidad de medir un fenómeno físico dado con gran detalle.

Es necesario que los nodos funcionen con pequeñas fuentes de energía y que se comuniquen por medio de canales inalámbricos. En la Figura 5 se puede ver un conjunto de nodos sensores.

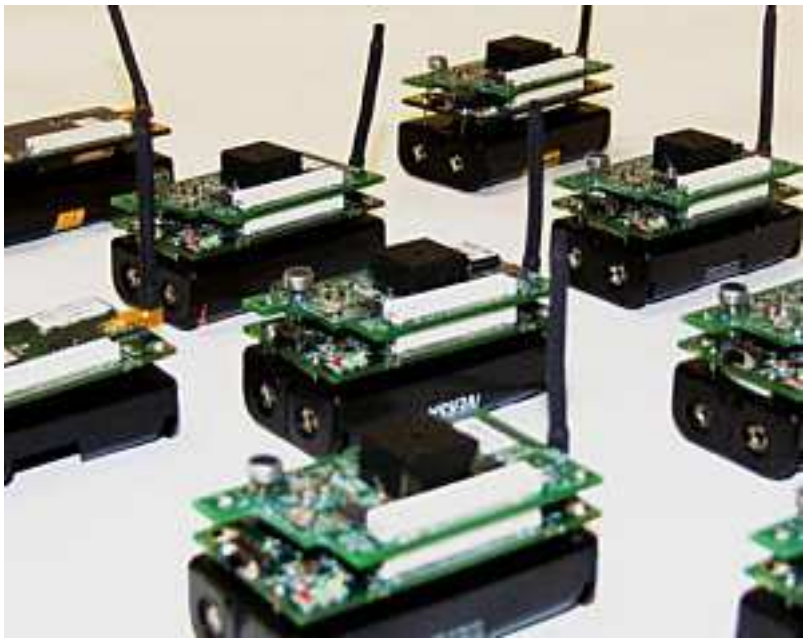

Figura 5: Nodos Sensores modelo MicaZ [17].

2.2.1.1 Componentes físicos

Los principales componentes físicos que integran los nodos sensores son:

- Microcontrolador: realiza tareas de procesamiento de datos y controla la funcionalidad de otros componentes del sensor. El microcontrolador puede contener una memoria *flash* de pequeño tamaño, un convertidor analógico-digital (ADC) y uno o varios relojes permiten manejar una hora local. Además incorporan una memoria RAM de 4KB para datos, esta memoria es muy pequeña y los nodos no pueden almacenar casi información en ella, de manera que deben retransmitir las mediciones que están realizando por radio. Debido a su flexibilidad para conectarse con otros dispositivos el consumo de energía es menor, ya que este dispositivo puede ir al estado de suspensión pero parte de la controladora puede seguir activa.

- Memoria externa. Los nodos tienen una pequeña memoria EEPROM de tipo FLASH. Esta memoria es muy limitada, típicamente 512 KB.

- Batería. La batería de un nodo sensor es muy limitada, por este motivo es necesario optimizar al máximo el gasto de energía tanto en el procesamiento de datos como en la comunicación. Actualmente las baterías son la única fuente de alimentación de los nodos. Normalmente se suele usar pilas o baterías electroquímicas, aunque se está avanzando en el desarrollo de nodos capaces de recargar su batería a través de la energía solar o de vibraciones energéticas.

- Sensores: Los sensores son dispositivos hardware que producen respuestas a un cambio en una condición física como la temperatura o la presión. Las señales analógicas percibidas por los sensores son digitalizadas y enviadas a los procesadores para su procesamiento.

- Radio: Existen varios tipos de radio para los nodos sensores pero actualmente las que más se usan son las que utilizan el estándar IEEE 802.15.4 y Zigbee por el amplio respaldo que cuentan de la industria a nivel mundial. Emplean diversas bandas de frecuencia de radio 2.4 GHz con una velocidad de transmisión de 250 Kbps, 915 MHz (banda americana) con una velocidad de transmisión de 40 Kbps, o 68 MHz (banda europea) con una velocidad de transmisión de 40 Kbps. Todas ellas tienen un alcance de entre 5 y 200 metros, siendo 50 metros lo habitual. El estándar Bluetooth y la IEEE 802.11b (WLAN) no se emplean en las redes de motes debido al alto consumo de energía que conllevan, a pesar de ofrecer velocidades de transmisión de 1 Mbps y 11 Mbps, respectivamente.

A continuación se detallan en la Tabla 1 los distintos nodos sensores que existen en el mercado y sus principales características:

Tabla 1: Listado de nodos sensores y características principales (extraído de: (http://en.wikipedia.org/wiki/List_of_wireless_sensor_nodes)

Nombre Nodo sensor	Microcontrolador	Datos de los dispositivos de comunicación	Datos del Programa Memoria	Memoria externa	Programación	Comentarios
FRIJOL	MSP430F169	CC1000 (300-1000 MHz) con 78,6 kbit / s		4 Mbit		Apoyo YATOS

Btnode	Atmel Atmega 128L (8 MHz @ 8 MIPS)	Chipcon CC1000 (433-915 MHz) y Bluetooth (2,4 GHz)	64 +180 K de RAM	128K FLASH ROM, EEPROM 4K	C y nesC Programación	BTnut TinyOS y apoyo
COTS	ATMEL Microcontrolador 916 MHz					
Punto	ATMEGA163		1K de memoria RAM	8-16K de Flash	weC	
Ojos	MSP430F149	TR1001		8 Mbit		Apoyo PeerOS
EyesIFX v1	MSP430F149	TDA5250 (868 MHz) FSK		8 Mbit		Apoyo TinyOS
EyesIFX v2	MSP430F1611	TDA5250 (868 MHz) FSK		8 Mbit		Apoyo TinyOS
Gwnode	PIC18LF8722	BIM (173 MHz) FSK	64k de memoria RAM	128k flash	C	De SO
IMote	Núcleo ARM 12 MHz	Bluetooth con el rango de 30 m	64K SRAM	Flash 512K		Apoyo TinyOS
IMote 1,0	ARM 7TDMI 12-48 MHz	Bluetooth con el rango de 30 m	64K SRAM	Flash 512K		Apoyo TinyOS
IMote 2,0	Marvel PXA271 ARM 11-400 MHz	TI CC2420 802.15.4/ZigBee compatible con la radio	32 MB de SRAM	32MB Flash		Microsoft. NET Micro, Linux, Soporte TinyOS
Iris	ATmega1281	Atmel AT86RF230 802.15.4/ZigBee compatible con la radio	8K de RAM	Flash 128K	nesC	TinyOS, MoteWorks Apoyo
KMote	TI MSP430 microcontrolador	250 kbit / s 2,4 GHz IEEE 802.15.4 Chipcon Transmisor inalámbrico	10k de memoria RAM	Flash 48k		Apoyo TinyOS y SOS
Mica	Atmel ATMEGA103 4 MHz CPU de 8 bits	RFM TR1000 radio 50 kbit / s	128 +4 K de RAM	Flash 512K	nesC Programación	Apoyo TinyOS
Mica2	Atmega 128L	Chipcon 868/916 MHz	4K de RAM	Flash 128K		TinyOS, SOS y MantisOS Apoyo
Mica2Dot	Atmega 128		4K de RAM	Flash 128K		
MicaZ	Atmega 128	TI CC2420 802.15.4/ZigBee compatible con la radio	4K de RAM	Flash 128K	nesC	TinyOS, SOS, MantisOS y Nano-RK Apoyo

Mulle	Renesas M16C	Bluetooth 2.0	31K de memoria RAM	4 K 384K Flash, 2 MB de EEPROM	Programación C	TCP / IP y los perfiles de Bluetooth: PAL, DUN, PAN y Apoyo ASPAN
Nymph	ATMEGA128L	CC1000		64 kB EEPROM		Apoyo MantisOS
René	ATMEL8535	916 MHz con ancho de banda de radio de 10 kbit / s	512 bytes de RAM	8K Flash		Apoyo TinyOS
SenseNode	MSP430F1611	Chipcon CC2420	10K de memoria RAM	48K Flash	C y NesC programación	Genos y TinyOS Apoyo
DeSunSpot	ARM 920T	802.15.4	512K de memoria RAM	4 MB de Flash	Java	Squawk la máquina virtual de J2ME
Telos	Motorola HCS08		4K de RAM			
TelosB	Texas Instruments MSP430 microcontrolador	250 kbit / s 2,4 GHz IEEE 802.15.4 Chipcon Transmisor inalámbrico	10k de memoria RAM	Flash 48k		Contiki, TinyOS, SOS y MantisOS Apoyo
T-Mote cielo	Texas Instruments MSP430 microcontrolador	250 kbit / s 2,4 GHz IEEE 802.15.4 Chipcon Transmisor inalámbrico	10k de memoria RAM	Flash 48k		Contiki, TinyOS, SOS y MantisOS Apoyo
weC	Atmel AT90S2313 AVR	RFM TR1000 RF				
XYZ	ML67 serie ARM / THUMB microcontrolador	CC2420 ZigBee compatible radio desde Chipcon	32K de memoria RAM	256 K Flash	Programación C	SOS Soporte del sistema operativo
FireFly	Atmel Atmega 1281	Chipcon CC2420	8K de RAM	128K FLASH ROM, EEPROM 4K	Programación C	Nano-RK RTOS Apoyo

2.2.2 Gateway

En una red de sensores inalámbricos el nodo *gateway* o puerta de enlace es el nodo encargado de recibir la información que han ido recopilando y transmitiendo los nodos sensores, y enviarla a un PC o estación base donde finalmente pueda ser procesada.

Para poder desarrollar esta labor los gateway disponen de una radio para poder comunicarse con los nodos sensores, y además de un puerto serie que les permite comunicarse directamente con la estación base.

El gateway permite la comunicación entre dos redes distintas, por un lado la red de sensores que se comunica a través de ondas; y la red de la estación base que se comunica a través de un medio guiado o cable.

En la Figura 6 se puede observar un gateway del modelo MIB600 [18] que utiliza un puerto Ethernet para comunicarse con la estación base.

Figura 6: Gateway MIB600 con conexión Ethernet

En la Tabla 2 se muestran algunos modelos de gateways que existen en el mercado.

Tabla 2: Listado de Gateways y sus características

	Microcontrolador	Motes conectables	Puerto de programación	Puerto de datos
SPB400	IntelPXA255	IRIS, MicaZ, Mica2	RS-232 o USB	Varias
MIB510	Atmega128	IRIS, MicaZ, Mica2,Mica-Sensor Board	Serial RS-232	Serial RS-232
MIB520	Atmega16L	IRIS, MicaZ, Mica2	Conexión USB	Conexión USB
MIB600	Atmega128L	IRIS, MicaZ, Mica2	Ethernet (10/100 Base-T)	Ethernet (10/100 Base-T)

2.3. Tecnología de software

En esta sección se va a intentar explicar detalladamente las diferentes tecnologías de software existentes para el desarrollo de aplicaciones sobre nodos sensores. Se analizará tanto el sistema operativo sobre el que se va a programar, como los lenguajes de programación y las herramientas de desarrollo que se necesitan tanto para compilar como para simular.

2.3.1 Sistemas operativos

A continuación se va a analizar algunos sistemas operativos diseñados específicamente para nodos sensores.

2.3.1.1 TinyOS

TinyOS [19] es un sistema operativo para redes de sensores inalámbricas que fue desarrollado por la Universidad de Berkeley [20].

TinyOS es un sistema operativo conducido por eventos, lo cual implica que su funcionamiento se decide mediante eventos señalizados por el hardware que llamarán a funciones o manejadores de eventos.

Está diseñado para poder responder a las necesidades de las redes de sensores, como se ha comentado antes las redes de sensores tienen unas características y limitaciones propias como pueden ser el reducido tamaño de memoria o la necesidad de un bajo consumo de energía.

El diseño del *kernel* de TinyOS está basado en una estructura de dos niveles de planificación.

- Eventos: Se producen cuando un componente de bajo nivel avisa a uno de alto nivel que ha sucedido algo pudiendo interrumpir las tareas que se están ejecutando, o cuando se produce un evento externo como la recepción de un paquete de datos.

- Tareas: Las tareas son porciones de código que se ejecutan de forma asíncrona mientras la CPU no tenga que ejecutar ningún evento. Generalmente se encargan de realizar la mayor parte del procesamiento.

Con este diseño se permite que los eventos, que son rápidamente ejecutables, puedan ser realizados inmediatamente, pudiendo interrumpir a las tareas, que tienen mayor complejidad computacional en comparación a los eventos. Esto permite un alto rendimiento en aplicaciones de concurrencia y un uso eficiente de la CPU y de la energía.

TinyOS está diseñado de manera flexible y modular para incorporar nuevas innovaciones rápidamente y para funcionar bajo las importantes restricciones de memoria que se dan en los nodos sensores.

El entorno de desarrollo de TinyOS soporta directamente la programación sobre diferentes microprocesadores y permite programar cada tipo con un único identificador para diferenciarlo, o lo que es lo mismo se puede compilar en diferentes plataformas cambiando este atributo en tiempo de compilación.

Las librerías y aplicaciones de TinyOs, se ha programado en una meta-lenguaje que deriva de C, cuyo nombre es nesC [21]. Este lenguaje es una versión de C basada en componentes que fue diseñada para programar sistemas empotrados. En nesC, los programas están compuestos por un conjunto de componentes que se enlazan para formar un programa completo.

Los componentes se enlazan a través de sus interfaces. Las interfaces son bidireccionales y especifican un conjunto de funciones que están implementadas bien por los proveedores o bien por los desarrolladores de aplicaciones. NesC esperará que el código que va a ser generado cree un programa con un ejecutable que contenga todos los elementos del mismo, así como los manejadores de las interrupciones de programas de más alto nivel.

Además en TinyOS existen varias herramientas que facilitan el desarrollo y posterior análisis de aplicaciones para las redes de sensores, que van desde aplicaciones para la obtención y manejo de datos, hasta sistemas completos de simulación.

El sistema operativo TinyOS presenta distintas implementaciones para los protocolos correspondientes a los distintos niveles de la pila de comunicaciones de un nodo sensor. En la Figura 7 se puede observar la torre de protocolos.

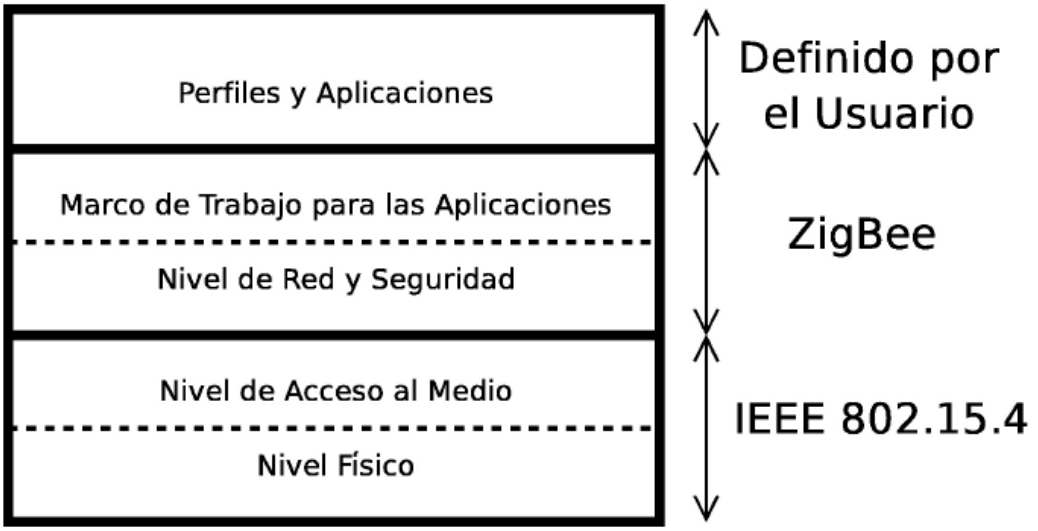

Figura 7: Torre de protocolos TinyOS

TinyOS proporciona distintos protocolos para el enrutamiento de mensajes. En el nivel físico y de enlace se utiliza el estándar de comunicaciones IEEE 802.15.4. El nivel de perfil y aplicación va a poder ser definido por el usuario y aquí será el lugar que ocupe el protocolo STP.

Como se ha comentado antes, el protocolo STP tiene definido por debajo otros protocolos y para poder interactuar con ellos sin ningún conflicto se tienen que añadir unas cabeceras a la trama. TinyOS utiliza a nivel de red el protocolo Zigbee y el propio sistema operativo define las cabeceras que se tiene que utilizar para poder invocarlo.

2.3.1.2 Otros sistemas operativos

Aunque la mayor parte de las aplicaciones para sensores inalámbricos utilizan el sistema operativo TinyOS, existen otros sistemas operativos para estos dispositivos. Estos son algunos sistemas operativos:

- Contiki [22]: Sistema operativo de libre distribución, multithread y multitarea que sólo puede utilizarse en un tipo de microcontroladores limitado.

- MANTIS [23]: *Multimodal Networks In-situ Sensors* es un sistema operativo para nodos sensores cuyo modelo de ejecución se basa en threads basados en prioridades.

- LiteOS [24]: Sistema operativo que se ideó para su uso en calculadoras.

- Bertha [25]: Se trata de una plataforma software diseñada para poder modelar, testear una red de sensores distribuida formada por múltiples nodos idénticos. Sus funciones principales se reducen a administrar procesos, manejar estructuras de datos y proporcionar un interfaz de red.

- CORMOS, *A Communication Oriented Runtime System for Sensor Networks* [26], se trata de un sistema operativo específico para redes de sensores inalámbricas.

- MagnetOS [27]: Sistema operativo distribuido utilizado en redes de sensores o ad-hoc, cuyo objetivo es ejecutar aplicaciones de red que precisen de un bajo consumo de energía.

2.3.2 Lenguajes de programación

En esta sección se van a describir los lenguajes de programación utilizados en la implementación del protocolo de tiempo.

2.3.2.1 nesC

Como se mencionó en la sección anterior nesC [22] es un lenguaje de programación basado en la sintaxis de C y orientado a componentes.

Este leguaje está muy optimizado para hacer frente a las limitaciones de memoria de los nodos sensores. El usuario crea su propio *componente* y los enlaza a su vez con otros componentes del sistema operativo y capas inferiores, de manera que todos esos componentes enlazados crean un programa completo.

Dos componentes podrán comunicarse entre sí mediante una *interfaz*, la cual definirá una serie de métodos (comandos y eventos) los cuales deberán ser implementados por el componente que use esa interfaz.

Todo componente está dividido lógicamente en dos partes: *configuración*, declaración de las interfaces utilizadas y proporcionadas por un componente y *implementación*, fichero donde se programarán las acciones que llevará a cabo el programa.

En la Figura 8 se muestra la estructura de un programa desarrollado en nesC. En esta imagen se pueden observar tres componentes: el primer fichero empezando por la izquierda es la configuración, el segundo es el módulo o implementación y el tercer fichero es un fichero de cabecera típico de C.

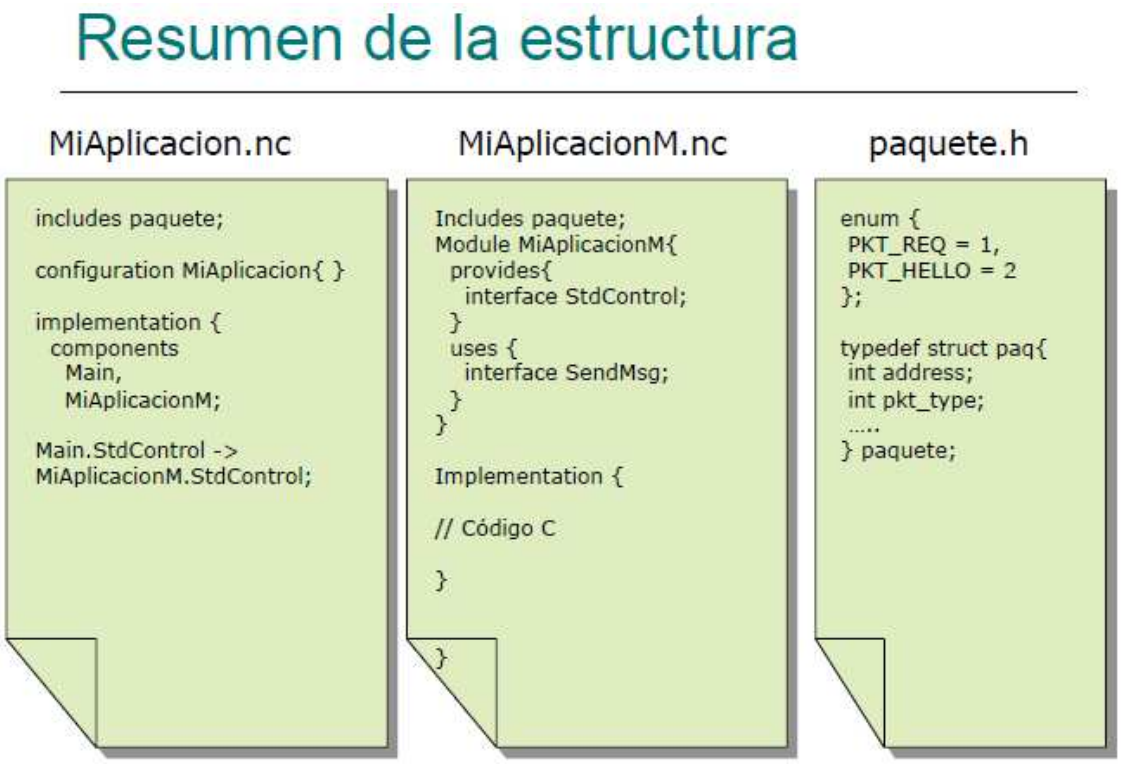

Figura 8: Estructura de un componente en NesC

Un programa realizado en nesC deberá generar una vez compilado, un programa con un ejecutable que contenga todos los elementos del mismo, incluyendo los manejadores de interrupciones de programas de más alto nivel.

Los tipos de datos que usa nesC son muy parecidos a los usados en C, por ejemplo:

- `uint16_t:` es un entero sin signo de 16 bits.
- `uint8_t:` entero de 8 bits.
- `bool:` booleano (TRUE, FALSE).
- `result_t:` es un tipo de datos que recoge el resultado de una función. Puede tomar el valor SUCCESS que implica la finalización sin fallo de la función o FAIL que implica la finalización errónea.
- Otros tipos de datos de C.

2.3.2.2 El lenguaje C

El lenguaje de programación C [28] fue desarrollado por Denis Ritchie y Ken Thompson en la década de 1970, en los Laboratorios Bell AT&T en Nueva Jersey. Los desarrolladores de UNIX [29] necesitaban un lenguaje pequeño y compacto para escribir su código UNIX, por lo tanto es un lenguaje orientado a la implementación de sistemas operativos.

Una de las grandes ventajas de este lenguaje de programación es su eficiencia ya que es posible utilizar sus características de bajo nivel para realizar implementaciones óptimas.
C es apreciado por la eficiencia del código que produce y es el lenguaje de programación más popular para crear software de sistemas, aunque también se utiliza para crear aplicaciones de propósito general.

2.3.3 Simuladores de aplicación

En esta sección se van describir las características generales de los simuladores que se usan en programas implementados para redes de sensores inalámbricas.

Los simuladores son aplicaciones que permiten emular el comportamiento de un programa. A continuación se describen brevemente dos de los más relevantes simuladores para redes de sensores: TOSSIM y Avrora.

2.3.3.1 TOSSIM

TOSSIM [30] es un simulador diseñado para el sistema operativo TinyOS ideado para facilitar el desarrollo de aplicaciones de red de sensores. TOSSIM compila el código directamente desde TinyOS de manera que los desarrolladores pueden poner a prueba no sólo de sus algoritmos, sino también sus aplicaciones.

Este simulador provee un completo entorno de simulación para redes de sensores inalámbricas, que abarca su comportamiento como red y la ejecución de la aplicación en cada nodo permitiendo la depuración de las aplicaciones que se estén programando.

La forma de invocar una simulación con TOSSIM es ejecutando la siguiente línea de comando:

```
./ejecutable [parametros] nodos
```

Donde *ejecutable* es nombre del ejecutable a simular, y *nodos* es el número de nodos para los que se va a lanzar la simulación. Cuando se desea lanzar una simulación, en el campo *parámetros*, se pueden activar una serie de opciones. En la Figura 9 se muestra cómo se usa TOSSIM y qué opciones podemos activar en una ejecución. Esta figura es la salida de la opción *help* que nos muestra la ayuda del simulador.

```
$ build/pc/main.exe --help
Usage: build/pc/main [options] num_nodes
[options] are:
-h, --help Display this message.
-gui pauses simulation waiting for GUI to connect
-a=<model> specifies ADC model (generic is default)
options: generic random
-b=<sec> motes boot over first <sec> seconds (default 10)
-e=<file> use <file> for eeprom; otherwise anonymous file is used
-l=<scale> run sim at <scale> times real time (fp constant)
-r=<model> specifies a radio model (simple is default)
options: simple lossy
-rf=<file> specifies file for lossy mode (lossy.nss is default)
implicitly selects lossy model
-s=<num> only boot <num> of nodes
-t=<sec> run simulation for <sec> virtual seconds
num_nodes number of nodes to simulate
```

Figura 9: Opciones de simulación TOSSIM

TOSSIM tiene mecanismos para el trabajo con el conversor ADC y con el *transceiver* del modelo de comunicación inalámbrica (radio model). Y puede comunicarse con otros programas externos que permitirán la interacción con la simulación, de la misma forma como se realizaría con una red inalámbrica real.

Además dispone de una herramienta gráfica, TinyViz [31], que permite visualizar e interactuar con las simulaciones. Usando un simple plug-in, los usuarios pueden desarrollar nuevas interfaces para la visualización dentro de TinyViz.

2.3.3.2 Avrora

Avrora [32] nace como un proyecto de investigación del Grupo de Compiladores de la Universidad de UCLA [33].

Este simulador es específicamente un emulador para las plataformas TinyOS/Mica2 y MicaZ. Se compone de un conjunto de herramientas de análisis y simulación de programas escritos para el microcontrolador AVR producido por Atmel [34] y nodos sensores Mica2 (en su versión original) y MicaZ. Avrora contiene un marco flexible para simular y analizar los programas, provee un completo API de Java e infraestructura para la experimentación, para añadir perfiles, y análisis.

Avrora permite analizar el ciclo de desarrollo de sistemas empotrados, permitiendo analizar los flujos de la ejecución y el diagnosticar posibles problemas del software antes de que este sea desplegado en el hardware y puesto en el entorno final. También provee un *framework* para el análisis de programas, permitiendo chequeo estático de software y desarrollando una infraestructura para futuras investigaciones en el análisis de programas.

Las principales características de Avrora son:

- El simulador que provee puede chequear los programas antes de ser desplegados en el dispositivo hardware con tiempos precisos en el ciclo de ejecución.

- La infraestructura de monitorización permite a los usuarios añadir monitorización o *check-points* online en el programa para su mejor seguimiento y pudiendo así ser optimizado fácilmente.

- Las utilidades de perfil permiten a los usuarios estudiar el comportamiento de sus programas en simulación.

- Las capacidades de instrumentación permiten una observación detallada del comportamiento del programa sin alterar la simulación y sin modificar el código fuente del simulador.

- El modo debug GDB permite depurar a nivel fuente y un desarrollo y testeo integrado.

- La herramienta gráfica de control de flujo puede crear una representación gráfica de las instrucciones del programa que permite mostrar la estructura del código.

- La herramienta de análisis puede analizar el consumo de energía de cada componente físico como la radio, CPU, memoria, etc. y ayuda a determinar el tiempo de vida de la batería del dispositivo.

- La herramienta de chequeo de pila puede ser usada para obtener el máximo tamaño de pila utilizado por el programa.

2.4 ZigBee

ZigBee [35] es un conjunto de protocolos de alto nivel de comunicación inalámbrica para su utilización con radios digitales de bajo consumo, basada en el estándar IEEE 802.15.4 de redes inalámbricas de área personal o *Wireless Personal Area Network*, (WPAN). Su objetivo son las aplicaciones que requieren comunicaciones seguras con baja tasa de envío de datos y maximización de la vida útil de sus baterías. Por ello este protocolo se utiliza en las redes de sensores inalámbricas.

ZigBee se basa en el nivel físico y el control de acceso al medio que se realiza a través de los protocolos MAC definidos en la versión de 2003 del estándar IEEE 802.15.4 [36], que desarrolla estos niveles para redes de área personal de baja tasa de transferencia (LR-WPAN's).

En la Figura 10 se muestra la estructura de protocolos de comunicaciones donde se sitúan los estándares Zigbee y IEEE 802.15.4.

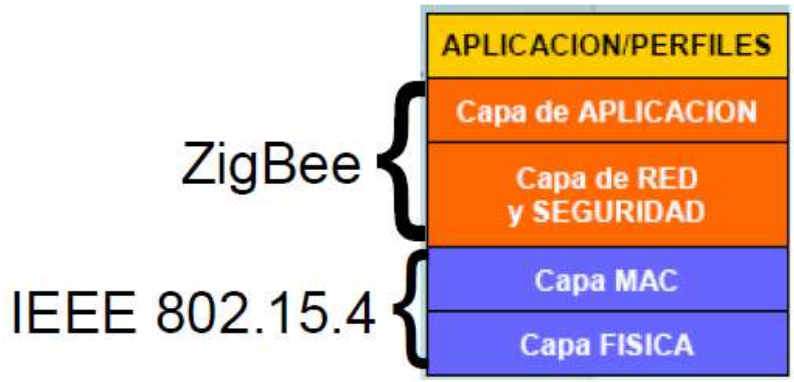

Figura 10: Capas de ZigBee 802.15.4

A través del componente *ZigBee device objects* (ZDO) que se encuentra en el nivel de aplicación del protocolo se puede realizar el mantenimiento de los roles de los dispositivos, identifica los dispositivos que se encuentra en un salto en la red y los servicios que ofrece, también realiza la gestión de peticiones de unión a una red, el descubrimiento de otros dispositivos y la seguridad.

ZigBee permite el uso de varias topologías de red: redes en estrella y árbol, así como redes en malla. Toda red necesita un dispositivo coordinador, encargado de su creación, mantenimiento básico y control de sus parámetros. En redes en estrella, el coordinador debe ser el nodo central, las redes en árbol y malla permiten el uso de enrutadores en el nivel de red.

Las principales características de Zigbee son:

- Su bajo consumo.
- Su topología de red en malla en la que cada nodo está conectado a todos los nodos.
- Su fácil integración, ya que permite fabricar nodos con muy poca electrónica.

Las redes ZigBee han sido diseñadas para conservar la potencia de manera que un nodo ZigBee reduce su consumo gracias a que puede permanecer dormido la mayor parte del tiempo, incluso varios días seguidos. Cuando se requiere su uso, el nodo ZigBee es capaz de despertar en un tiempo ínfimo, para volverse a dormir cuando deje de ser necesario. Un nodo cualquiera despierta en aproximadamente 15 milisegundos.

Los conceptos de ahorro de energía en la redes de sensores se engloban en los protocolos Low Power Listening (LPL). Con los protocolos LPL cada vez que el nodo despierta, enciende la radio y comprueba la actividad del canal. Si se detecta actividad, el nodo permanece despierto durante el tiempo necesario para recibir un paquete. Después de la recepción el nodo vuelve a dormir.

Si no se ha recibido ningún paquete se considera un "*falso positivo*" y un *timeout* fuerza que el nodo vuelva a dormir. La correcta evaluación del canal es fundamental para que se logre el objetivo de los protocolos LPL, no sólo porque permite encontrar el canal libre en la transmisión de paquetes sino también para determinar si está activo. No hay sincronización explícita de los nodos.

En una red 802.15.4 pueden funcionar dos tipos de dispositivos: dispositivos de funcionalidad completa o *full funcion device* (FFD) y dispositivos de funcionalidad reducida o *reduced funcion device* (RFD).

Los FFD pueden comunicarse tanto con ellos mismos como con otros dispositivos de funcionalidad reducida (RFD), mientras que los RFD solo pueden hacerlo con dispositivos de funcionalidad completa (FFD).

El nivel de enlace va a utilizar una topología de árbol es decir se puede definir una red descentralizada con un conjunto de redes en forma de estrella ordenadas en una jerarquía en la que cada nodo de la red va a decidir a quién reenviar lo paquetes según la conectividad de la red. En la Figura 11 se muestra una estructura genérica de una topología de tipo árbol.

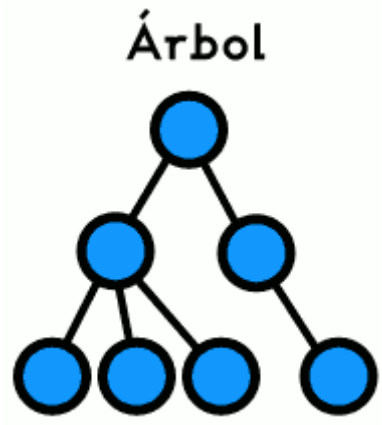

Figura 11: Topología árbol

A continuación se muestra la forma de una red con topología de estrella, en la que hay un nodo central con el que el resto de nodos se comunican. En la Figura 12 se muestra su estructura genérica.

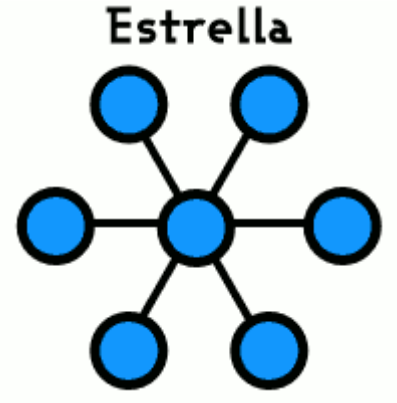

Figura 12: Topología de estrella

3. Network Time Protocol (NTP)

En este capítulo se va a explicar detalladamente el protocolo de aplicación *Network Time Protocol* (NTP), analizando pormenorizadamente su comportamiento y los usos que de él se hace en la actualidad.

3.1 Descripción y características

En un sistema distribuido, no existe ningún reloj global. Cada procesador tiene su propio reloj interno y su propia noción de tiempo. En la práctica, estos relojes fácilmente pueden surgir desfases de segundos, acumulando errores que se harán significativos con el tiempo. Debido a estos problemas surgió la necesidad de crear un protocolo de tiempo que permitiera tener un sistema distribuido con una hora común en cada uno de sus nodos. Es decir, que cada uno de los componentes de la red almacenará la misma hora.

Network Time Protocol es un protocolo de comunicación que se usa en Internet para sincronizar los relojes de sistemas informáticos. Es uno de los protocolos en uso más antiguos, ya que se desarrolló en 1985 por Dave Mills en la universidad de Delaware [37].

En las redes informáticas se producen retardos desde la emisión del paquete de datos hasta la recepción en destino. Estos retrasos se deben a una demora en la propagación y en la transmisión del paquete. A la suma de estos retrasos temporales se le denomina *latencia.*

El protocolo NTP está diseñado para poder encaminar paquetes en redes con latencia variable. Proporciona una precisión inferior a un milisegundo en LAN y un máximo de unos pocos milisegundos en WAN. Configuraciones típicas de NTP utilizan múltiples servidores redundantes y diversas rutas de red para lograr una alta precisión y fiabilidad.

El protocolo NTP puede trabajar en uno o más modos de trabajo. Uno de los más conocidos es el modo cliente/servidor, también llamado maestro/esclavo. En este modo, un cliente se sincroniza con un servidor igual que en el modo RPC convencional.

NTP soporta el modo *broadcast* por el cual muchos clientes pueden sincronizarse con uno o varios servidores, reduciendo el tráfico en la red cuando están involucrados un gran número de clientes. El *multicast* IP también puede ser usado cuando la subred abarca múltiples redes de trabajo.

La configuración puede ser un serio problema en grandes subredes. NTP usa el modo *broadcast* para soportar grandes cantidades de clientes pero para los clientes que solo escuchan es difícil calibrar el retraso y la precisión que pueden sufrir. En NTP, los clientes determinan el retraso a la vez que buscan un servidor en modo cliente/servidor y luego cambian a modo sólo escucha. Además, los clientes NTP pueden hacer un *broadcast* de un mensaje especial para solicitar respuestas de servidores cercanos y continuar en modo cliente/servidor con los que le respondan.

NTP opera en una jerarquía de niveles, donde cada nivel se le asigna un número al que se le llama estrato. La Figura 13 muestra esta jerarquía. Los sistemas del estrato 1 (nivel primario) están sincronizados con un reloj externo tal como un reloj GPS ó algún reloj de radio. Los sistemas del estrato 2 (nivel secundario) de NTP derivan su tiempo de uno ó más de los sistemas del estrato 1, y así sucesivamente.

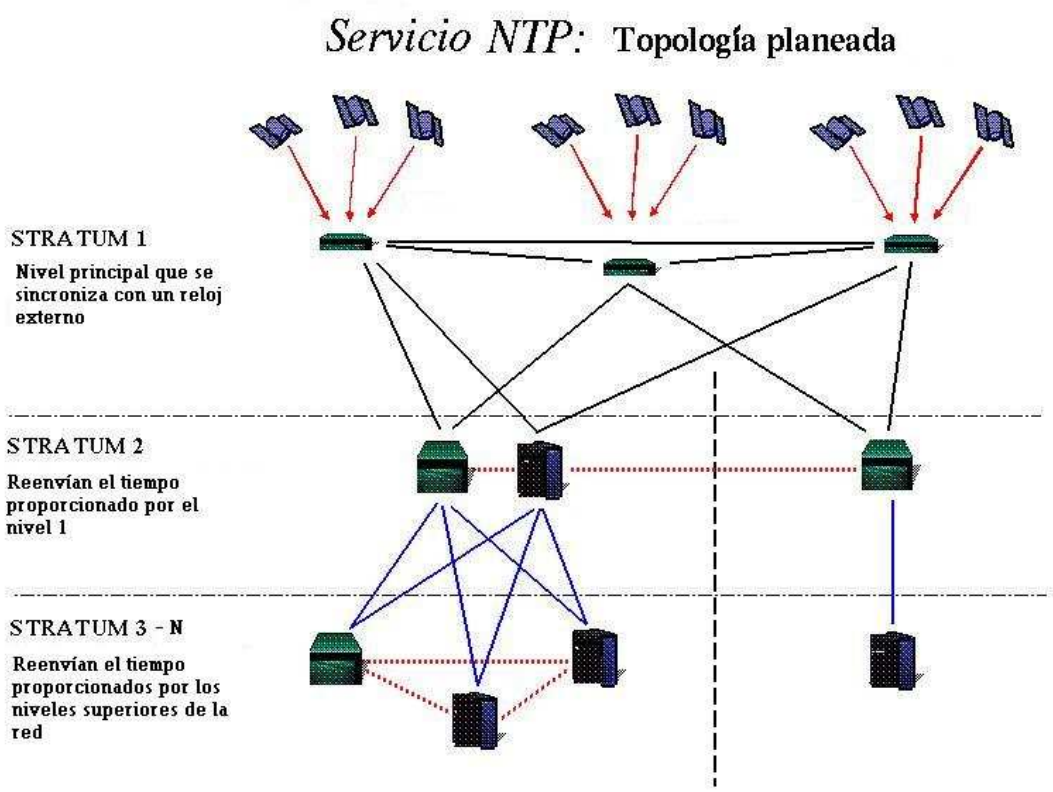

Figura 13: Arquitectura en niveles de NTP

3.1.1 Uso de NTP en Internet

El protocolo de tiempo de red es ampliamente utilizado para sincronizar un equipo con los servidores de Internet o de otras fuentes, tales como un receptor de satélite o de radio o un módem.

El servicio de tiempo NTP está ampliamente disponible en Internet. NTP tiene varios miles de servidores en la mayoría de los países y en todos los continentes del mundo, incluida la Antártida. Se estima que estos servidores sean más de 25 millones de ordenadores distribuidos en Internet.

Disponer de todos los equipos sincronizados correctamente con un servidor de tiempo proporciona muchas ventajas, como por ejemplo, fiabilidad de los mensajes de correo, gestión de la seguridad, manejo de certificados digitales, etc.

Existen muchas redes que utilizan el protocolo NTP, por ejemplo la red *Public NTP Server de Suiza.* En esta red los servidores de HOSTINET se sincronizan cada hora con un servidor suizo.

En ocasiones algunos sistemas de publicación web pueden mostrar erróneamente la fecha y hora en los artículos publicados o en diferentes módulos añadidos. Este error no se produce por culpa del servidor, que está perfectamente sincronizado, sino porque muchas aplicaciones tienen una sección para configurar la zona horaria que por defecto entiende que la hora del servidor debe ser GMT +0. Si el huso horario se modifica en esa sección por GMT+1, correspondiente a la zona peninsular de España, la hora que toma la aplicación se modifica. Deberá marcarse en ese caso GMT+0 para que sea la hora sincronizada del servidor la que se tome por defecto.

En sistemas de base de datos, el orden en el cual los procesos realizan actualizaciones sobre una base de datos es importante para asegurar una correcta visualización de la base de datos.

3.2 Protocolos de tiempo en redes de sensores

Las actuales WSN no disponen de ningún protocolo de tiempo estándar que sea utilizado masivamente y de forma general por las aplicaciones de redes de sensores. El principal problema que se detecta al plantearse un protocolo de sincronización para redes inalámbricas es que el medio de sincronización no es guiado, es decir se va a necesitar el envío de mensajes a través de las radios de los nodos para poder transmitir la hora de sincronización, y esas comunicaciones son costosas en términos de energía.

Además como los nodos no tienen reloj absoluto interno no es posible mantener la sincronización global, así que es necesario que la red se vaya sincronizando periódicamente. En su lugar, los nodos van a disponer de varios temporizadores por software que permiten sincronizar sus acciones.

La hora de sincronización tiene que ser obtenida desde el único dispositivo de la red WSN que sí dispone de reloj: la estación base. La obtención de la hora desde la estación base tiene una ventaja ya que ésta puede, a su vez, estar sincronizada con un reloj global mediante un protocolo NTP a través de Internet.

3.3 Trabajo relacionado

Esta sección describe el trabajo relacionado en protocolos de sincronización de tiempo que tienen las redes de sensores como escenario.

El principal problema que presentan los esquemas tradicionales de protocolos de tiempo como NTP, es que no se pueden aplicar directamente a redes inalámbricas ya que en ellas la energía es limitada.

Existen varios trabajos relacionados con los protocolos de tiempo y las redes de sensores inalámbricas que proponen mejoras en este campo. A continuación se van a analizar algunos de estos trabajos.

3.3.1 Time Synchronization for Wireless Sensor Networks

Un proyecto relacionado con los protocolos NTP y las redes de sensores inalámbricas es *Time Synchronization for Wireless Sensor Networks*, desarrollado por Jeremy Elson y Deborah Estrin [38].

En este trabajo se observa la criticidad de la sincronización de tiempos en los sistemas distribuidos y como consecuencia propone una ampliación de la funcionalidad de los protocolos NTP.

Los autores consideran que los sistemas de sincronización tradicionales gastan demasiada energía y tiempo en labores como el control de redundancia o la distribución del mensaje. Las redes de sensores tienen una energía muy limitada y se ha de ser conscientes del tiempo y energía que se está consumiendo. Para solucionar este problema desarrollan una sincronización post-facto que consiste en sincronizar el nodo una vez el hecho ya se ha producido y así sólo

enviar mensajes de sincronización cuando es necesario. Con este protocolo no se van a enviar mensajes de sincronización por la red, con la consiguiente pérdida de energía, si el nodo no está realizando ninguna labor que requiera esa sincronización. Los mensajes de sincronización sólo se enviará cuando un nodo requiera una hora global, así se minimiza el uso de la radio en los nodos.

En la descripción de WSN que proponen pueden existir distintos tipos de nodos sensores (red heterogénea), donde cada nodo tiene unas características distintas: los nodos más pequeños dispondrán de una radio de corto alcance mientras que los nodos más grandes dispondrán de radios de largo alcance o incluso sistemas GPS y WWVB. El sistema WWVB es una estación que controla los relojes de radio en toda América del Norte y es utilizada para sincronizar. Por estos motivos los autores elaboran un conjunto de algoritmos en los nodos sensores que les permitan ajustar el consumo y la energía a las necesidades de la red.

En la Figura 14 se puede observar la diferencia de precisión entre el método post-facto y la precisión de NTP. Cuanto más precisa sea la sincronización menos paquetes van a circular por la red y más energía se va a ahorrar.

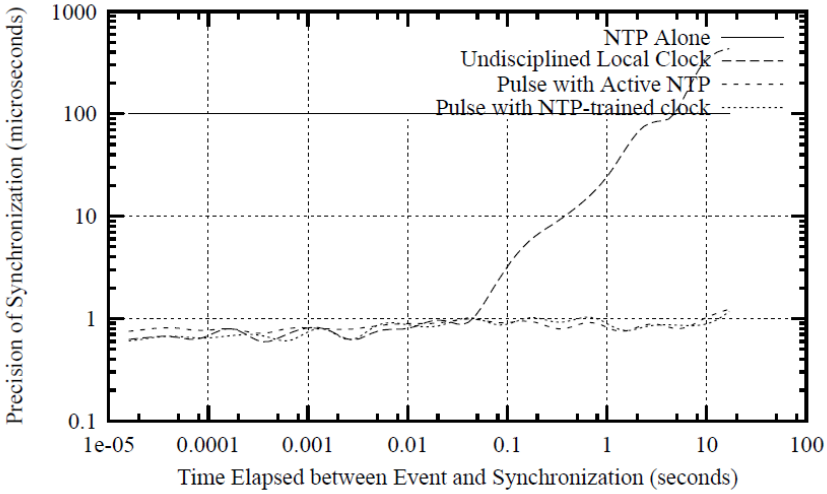

Figura 14: Comparación método post-facto y NTP. (http://www.cens.ucla.edu/Estrin/papers/timesync.pdf)

Todos los nodos de la red deben estar el mayor tiempo posible en un estado “sleep” en el cual estarán consumiendo la mínima energía posible, sólo quedará activo el micro-controlador del nodo. Este micro-controlador estará escuchando cualquier señal que reciba. Cuando le llegue un estímulo, el nodo registrará el tiempo del estímulo con respecto a su reloj local y emitirá un impulso a los nodos de su zona a través de un mensaje por radio, de ésta manera sincronizaremos los nodos de la red ahorrando la máxima energía posible.

3.3.2 Wireless Sensor Networks: A new regime for time

El proyecto “Wireless Sensor Networks: A new regime for time” [39] realiza un análisis de las redes de sensores inalámbricas y los protocolos de sincronización. En este artículo se comenta que las características físicas en un protocolo de sincronización son cruciales y que por lo tanto hay que estudiar cuidadosamente los factores de precisión de la red, alcance, coste, vida de la red y el empleo de la energía.

En los esquemas de sincronización de tiempo tradicionales los relojes son pre-sincronizados cuando un acontecimiento ocurre y es fechado. Esto puede causar problemas con la retransmisión de mensaje y puede causar un comportamiento imprevisible de la aplicación. En

este proyecto se propone la sincronización después de que se produzca un evento (post-facto). Los relojes de los nodos corren de forma independiente en cada uno de ellos. Cuando se recibe un evento se comparan las fechas de inicio de los relojes y si son diferentes se sincronizan. Esto elimina la necesidad de que los nodos estén sincronizados antes de que se produzca un evento, y por lo tanto, la energía de sincronización sólo es gastada después de un acontecimiento de interés.

3.3.3 FTSP

Flooding Time Synchronization Protocol (FTSP [40] desarrollado por la Universidad Vanderbilt [41] es un protocolo que permite la sincronización de una red inalámbrica a través de un método de inundación. Este protocolo establece jerarquías entre los nodos de la red; el nodo líder, periódicamente envía un mensaje de sincronización de manera que cada nodo que lo reciba lo retransmitirá a los otros nodos de la red que esté accesible para él. El tiempo local de cada nodo será definido mediante la utilización de marcas de tiempo de los mensajes de radio en las capas bajas de la radio, permitiendo que el error de sincronización de este protocolo sea del orden de unos pocos milisegundos.

FTSP utiliza ancho de banda de comunicación relativamente baja utilizando un solo sentido de tráfico para la sincronización, esto es posible porque los mensajes se transmiten desde un nodo líder hasta el resto de nodos y no se realiza transmisión desde los nodos esclavos hacia el líder.

Este protocolo tiene un gran poder ya que con mantener el nodo que se selecciona para ser el líder siempre sincronizado, se puede realizar una difusión mundial de la hora de sincronización.

El protocolo FTSP es muy similar al que se pretende desarrolla en este proyecto pero tiene algunas diferencias como por ejemplo: en FTSP el nodo líder envía de forma periódica mensajes de sincronización y esto tiene una desventaja que en este proyecto se quiere evitar que es saturar el canal de comunicación para evitar que se produzcan cuellos de botella y perdida de paquetes, y un uso excesivo de la energía de los nodos. Por otro lado FTSP utiliza un único sentido en las comunicaciones, es decir el mensaje se crea en el nodo líder y va descendiendo en cadena; además este nodo líder es el responsable de ir manteniendo la hora de sincronización en la red, el resto de nodos no mantienen la hora de forma autónoma.

La gran ventaja de este protocolo es que permite sincronizar una red con sólo tener actualizada la hora de un nodo elegido como nodo líder.

4. Análisis de requisitos

En este capítulo se va a presentar la Especificación de Requisitos de Software. Se recogen los requisitos funcionales y no funcionales que se deberán tener en cuenta a la hora de desarrollar el protocolo así como el hardware y software de los dispositivos que se comunicarán con los nuevos motes.

En el proyecto se desea realizar un protocolo de tiempo llamado *Synchronized Time Protocol* (STP) que posibilitará la sincronización de los nodos de una red de sensores inalámbrica. Para poder desarrollar este protocolo es necesario analizar los requisitos que se van a desear cubrir y los medios con los que se van a contar.

4.1 Arquitectura

Se desea desarrollar un protocolo de tiempo denominado Synchronized Time Protocol (STP). El análisis de la arquitectura del protocolo STP va a permitir definir la funcionalidad de cada uno de los elementos de la red.

4.1.1 Arquitectura de STP

El protocolo STP tiene como principal función la sincronización de una red de sensores inalámbricos. Para poder llevar a cabo esta funcionalidad es necesario obtener la hora de un sistema externo, como se realiza en los protocolos NTP, para ello un nodo de la red solicitará a un sistema externo la hora a transmitir por la red, esa hora se recibirá y se propagará a toda la red donde se almacenará y actualizará.

El protocolo STP define tres tipos de nodos:

1. Gateway Maestro: El gateway Maestro es un dispositivo gateway que va a estar conectado físicamente a través de un puerto USB a un ordenador del que podrá obtener la hora de dicho sistema para almacenarla y propagarla a los nodos que realizan la función de gateways esclavos en la red.

2. Gateway Esclavo: Son nodos sensores “especiales” que actúan como servidores de tiempo. Estos nodos son físicamente iguales que los nodos sensores clientes pero van a tener una funcionalidad distinta, ya que van a tener la labor de retransmitir los paquetes de sincronización que obtienen del gateway maestro a los nodos sensores clientes.

3. Nodo sensor cliente: Son los nodos sensores que se encargan de recibir la hora de sincronización, almacenarla y mantenerla. Estos nodos sensores pueden estar programados para realizar tareas de medición, pero esa labor es independiente del protocolo STP. Los nodos sensores clientes están situados en el nivel más bajo de la estructura de STP.

4.1.2 Arquitectura del sistema

El modelo de enrutamiento que se va a usar en el proyecto es una adaptación del modelo multi-salto ya que no se va a realizar una comunicación directa entre los nodos de la red en el momento de la sincronización.

En el protocolo STP es la estación base el origen de los datos y no el destino como suele ser en las aplicaciones de redes de sensores. En este protocolo los datos salen de la estación base, se envían al gateway maestro y éste se comunica con los gateways esclavos. Como se puede ver, en esta parte de la comunicación el mensaje de datos se trasmite en n saltos, pudiendo pasar por varios nodos para llegar a los gateways esclavos, y a esto se le denomina un protocolo de comunicación multisalto. Cuando los gateways esclavos ya tienen la fecha de sincronización envían un mensaje con este dato a todos los nodos que están accesibles para él; por lo tanto se realiza otro salto en la transmisión del mensaje.

Toda WSN tiene una arquitectura en tres niveles:

- Un primer nivel lo define la estación base. Esta estación base es un ordenador sobre el que se van volcando los datos recogidos por los nodos sensores a través de una aplicación. Un experto será el encargado de analizarlos para poder realizar un estudio y extraer las conclusiones.

- El segundo nivel de las redes inalámbricas lo compone un gateway, este componente es una mote que dispone de un puerto USB de manera que se puede conectar a la estación base. Este nodo tiene dos funciones básicas: En primer lugar tiene que recibir los paquetes que le envían los nodos sensores a través de la radio y en segundo lugar tiene que transmitir dicha información a la estación base a través del puerto USB.

- El último eslabón de la jerarquía lo compone los nodos sensores, o *motes,* distribuidas por la red y encargadas de obtener datos de luminosidad, corrosión de metales, temperaturas, etc. y enviarlas vía radio al gateway.

Como se ha visto en capítulos anteriores, los protocolos NTP tienen una arquitectura de 3 niveles, los cuales se van a adaptar para permitir el uso de un protocolo de tiempo. A continuación presentaremos en qué consiste tal adaptación.

- En el estrato 1 se dispone de un nodo que llamaremos *gateway maestro* (GM). Este nodo tendrá como función básica recuperar la hora del sistema, almacenarla y trasmitirla. Se corresponden en los protocolos NTP con los servidores de tiempo.

- En el estrato 2 se dispone de un conjunto de nodos denominados *gateway esclavos.* Estos nodos reciben desde el estrato 1 la hora. Los gateway esclavos son los encargados de enviar la hora a todos los nodos sensores que sean accesibles por él. Se considera que un nodo sensor es accesible cuando las ondas de radio que emite el gateway puedan ser escuchadas por la radio de algún nodo sensores.

- En el estrato 3 están todos los nodos que no son gateways, ya sean maestros o esclavos, es decir, los nodos sensores que actúan como clientes en el protocolo NTP. Estos nodos reciben la hora, la almacenan y la mantienen.

STP usa los protocolos de red multisalto de por defecto para conseguir la transmisión eficiente de mensajes entre los distintos nodos. En un protocolo de sincronización aplicado a redes de sensores inalámbricos es muy importante evitar la pérdida de paquetes y el reenvío de los mismos, ya que cada mensaje se trasmite a través de la radio del nodo, y la radio es el componente del nodo que más energía gasta.

De tal manera que una red WSN estándar tendría una apariencia similar a la de la Figura 15:

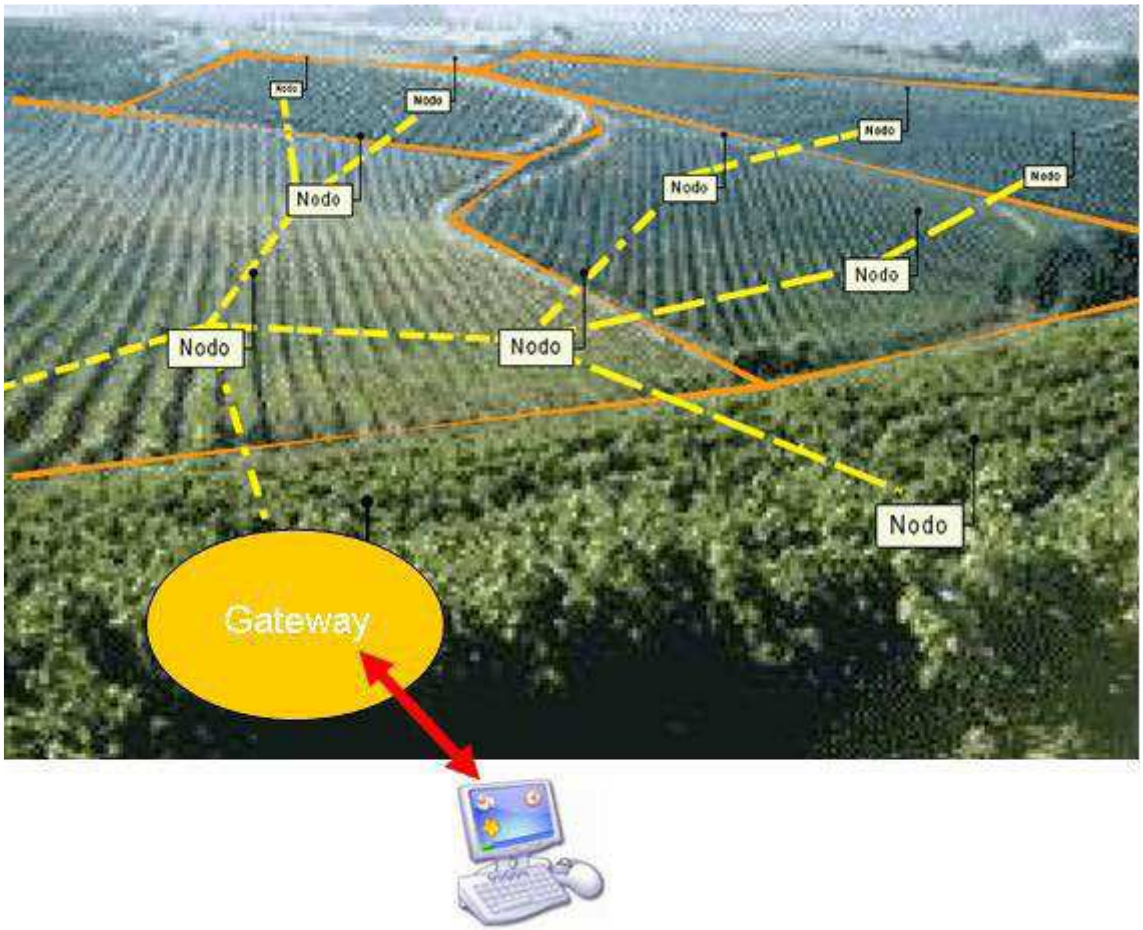

Figura 15: Una WSN típica sobre un campo de viñedos

En la red que se va a definir para poder implementar un protocolo de tiempo existirán algunas variaciones sobre la red estándar. En primer lugar va a aparecer un cuarto nivel, ya que se va a necesitar nodos que realicen las funciones de enrutadores de paquetes para evitar un cuello de botella en el gateway maestro, de tal manera que los niveles quedan así:

- En el primer nivel se va a tener la estación base, que además de recibir información del gateway que tiene conectado por USB (gateway maestro en nuestra red), se encargará de enviarle parámetros de configuración de forma que la comunicación va a ser bidireccional.

- En el segundo nivel se va a tener el *gateway maestro* (GM) que va a recibir de la estación base los parámetros de configuración, y además va a enviar a la estación base los datos obtenidos desde la red. El *gateway maestro* va a enviar mensajes de sincronización a los *gateways esclavos*.

- En el tercer nivel se encuentran los *gateway esclavos* (GS). Estos nodos van a tener la labor de encaminar paquetes de sincronización por la red, son principalmente nodos enrutadores.

- En el cuarto nivel van a estar los nodos sensores. Para el protocolo STP estos nodos pueden recibir paquetes desde cualquier otro, van a almacenar la hora y mantenerla. Cuando estos nodos reciban un mensaje de sincronización lo transmitirán a toda la red.

La Figura 16 muestra la estructura que se va a utilizar en el protocolo STP. La mayor diferencia que se encuentra con una red WSN estándar es que se va a incluir un nuevo actor: el gateway maestro, que va a ser la mote encargada de comunicarse con el resto de sensores de la red a través de la radio; y además, de comunicarse con la estación base a través de un puerto USB.

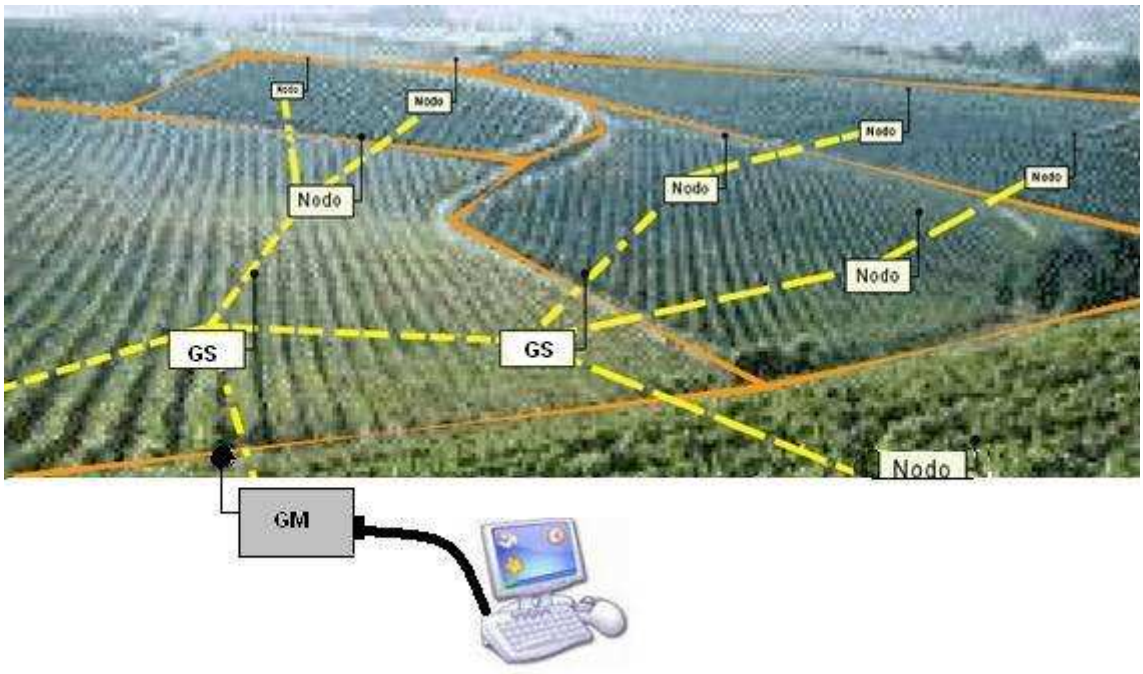

Figura 16: Una WSN usando el protocolo STP

Cuando se analizan las redes ya existentes se contemplan 3 actores principales: la estación base que se encarga de recibir los paquetes que le envía el *gateway* y a través de una interfaz humana se los presenta a un especialista o experto en el tema; el *gateway* también se encarga de recibir por radio paquetes de los nodos sensores y se los envía a través de un puerto USB a la estación base. Los nodos sensores o *motes* encargadas de realizar mediciones y enviárselas al *gateway*. Para poder realizar la sincronización de red esta estructura es incompleta por lo que se va a tener que definir un nuevo modelo de red añadiendo un actor nuevo en la comunicación. Este nuevo elemento es el *gateway esclavo*, nodo que va a comunicarse con el *gateway* maestro a través de la radio y va a obtener la hora de la red, estos nodos tendrán como función enrutar dicha hora a los nodos sensores; su labor va a ser fundamental para sincronizar la red.

Los *gateways esclavos* van a permitir que si un nuevo nodo sensor se incorpora a la red pueda sincronizarse, ya que los gateways envían la hora a la dirección de broadcast, es decir, a todos los nodos que escuchen con su radio en el perímetro de alcance de la radio del *gateway*. Además, el protocolo es tolerante a fallos, ya que si un nodo sensores se queda sin batería o se estropea, y sale de la red, el resto de nodos no tendrán ninguna consecuencia por la baja. Si es un *gateway* esclavo el que sufre una avería parte de la red podría quedar desincronizada siempre y cuando ninguno de los otros *gateway* esclavo pueda enviarle paquetes de sincronización por no ser alcanzable por su radio.

Si el *gateway maestro* se desconecta de la red, ésta seguiría sincronizada aunque no podría enviar paquetes hacia la estación base. Si el fallo se propagase en el tiempo la red se iría poco a poco desincronizando por la latencia de la red e incluso por el ruido que pueda influir en la comunicación.

Es un requisito crítico el conseguir que todos los nodos sensores de la red, pese a no tener reloj propio, puedan mantener la sincronización tras haber recibido de otro nodo una hora, y de esta manera, que ellos mismos puedan enrutar paquetes teniendo un tiempo dado como referencia.

4.2 Requisitos funcionales

A continuación se describen los requisitos funcionales del protocolo STP:

- Debe existir un único *gateway maestro* que estará conectado a la estación base. Su dirección por convenio será la dirección 0x00.

- En la estación base se deberá capturar la hora local desde el reloj del sistema. Con esta información y el número de gateways esclavos disponibles en la red, se compondrá un paquete de datos que transmitirá al gateway maestro. La aplicación programada en la estación base se ha de programar en lenguaje C. No existe petición explícita del gateway maestro.

- Cuando el gateway maestro se conecta a través del puerto a la estación base quedará a la espera hasta que la estación base genere los paquetes de datos con la hora del sistema y la dirección de los gateways esclavos que hay en la red. La estación base va a generar tantos paquetes como gateways esclavos se definan en la red.

- Cuando el gateway maestro recibe el número de gateways esclavos que tiene la red realiza una asociación de cada uno de esos gateways esclavos a una dirección en la red. Por convenio, se van a usar las direcciones más bajas de la red para los gateways esclavos. De manera que si el usuario ha indicado que tienen que existir 5 gateways esclavos, el gateway maestro recibirá un paquete con las direcciones de los GS 0x01, 0x02, 0x03, 0x04 y 0x05.

- El número de gateways esclavos que van a existir en la red tiene que ser siempre mayor que 0 y menor que 255, que es la dirección más alta de red. La trama del mensaje debe contener al menos:

    - dirección de origen del gateway esclavo o maestro que emite el mensaje;
    - la dirección de destino que es la dirección del nodo cliente o broadcast;
    - un booleano que indica si el mensaje es una petición de sincronización o no; es decir, en el caso que el booleano estuviera a falso indicaría que ese mensaje es una respuesta;
    - y finalmente, una estructura capaz de almacenar la hora que se va a usar para sincronizar los nodos sensores de la red.

- El formato de la hora debe ser: hh:mm:ss, donde “hh” es la hora (00..23), “mm” son los minutos (00..59) y “ss” son los segundos (00..59).

- Un gateway esclavo debe enviar un mensaje al gateway maestro de la red que está en la dirección 0x00 para poder sincronizarse. Cuando el servidor reciba la petición el gateway esclavo recogerá la hora del sistema, generará una trama y responderá al mensaje. Si el mensaje de sincronización es recibido por el GS, éste queda sincronizado. Si por el contrario el servidor al que se consulta no está disponible o no contesta a tiempo, entonces el gateway esclavo, vuelve a enviar la petición cada período de tiempo configurado o timeout.

- Los nodos sensores cliente pueden solicitar la hora de sincronización a cualquier nodo que está accesible incluyendo los gateways esclavos y maestro de la red. Para ello enviarán un mensaje de tipo petición a *broadcast*. Si el nodo no recibe contestación en un tiempo configurado o *timeout*, volverá a reenviarla petición de sincronización.

- Los nodos sensores cliente que reciban la hora de sincronización enviarán un mensaje a broadcast para ayudar a la sincronización de la red ya que es posible que no todos los nodos sensores sean accesibles por lo gateways.

- Se dice que la red está sincronizada cuando todos los nodos de la red han recibido la hora obtenida desde la estación base, teniendo en cuenta que podría haber un desfase entre el envío y la recepción. Para mantener la hora actualizada, los nodos tienen que aprovechar los relojes locales y temporizadores que usa la aplicación. Este proceso que va actualizando la hora en el nodo puede tener algún pequeño desfase, por lo que el siguiente mensaje de tiempo que se reciba debería sustituir a la hora actual del nodo.

- Se deberá adaptar el formato de ordenación de bytes de la red y los de la estación base dependiendo de la arquitectura de ordenación de la estación base. Se han de analizar los formatos con los que almacena más de un byte cada dispositivo. Sabemos que los motes son dispositivos *little endian* mientras que el PC que se va a usar en este proyecto como estación base es *big endian*.

- El nodo cliente que se conecte a la red tiene que enviar un mensaje *broadcast* solicitando la hora. El nodo cliente también podría recibir un mensaje de sincronización sin haberla solicitado.

- El nodo cliente que se desconecte de la red no tiene efecto sobre el resto de la red.

- El *gateway* maestro es un punto crítico en la arquitectura de la red, y por tanto debe procurarse una disponibilidad alta de este nodo.

- Por convenio, la dirección de *broadcast* es 0xffff.

- Por convenio, la dirección del puerto serie USB es 0x007e.

4.3 Requisitos de sistema

A continuación se describen los requisitos de sistema del protocolo STP:

- El protocolo de comunicación debe ser lo más genérico y flexible posible de manera que se adapte a las posibles arquitectura de las redes, independientemente del número de nodos que formen la red. Para el protocolo STP tiene que ser transparente si la red de sensores tiene 5 nodos en forma de estrella o 500 nodos con estructura de árbol.

- Los nodos que se van a utilizar en nuestra red van a ser MicaZ. Los motes MicaZ, módulo MPR2400CA son una plataforma hardware para el desarrollo de sistemas de medición, control de acceso, etc. Cada mote MicaZ incorpora un microprocesador ATMEL Mega128L de 7 Mhz. Dicho procesador incorpora una EEPROM de 128 Kilobytes de espacio para los programas y una memoria RAM de 4 Kb para datos. El procesador está conectado a través del puerto UART 2 a una memoria flash externa de 512 KB de capacidad. Dicha memoria está al servicio de las aplicaciones que requieran guardar datos de manera permanente. Ofrecen una tasa de transferencia de datos vía

radio, gracias a la interfaz 802.15.4 y por tanto, compatibles con ZigBee, de hasta 250 Kbps. Permite comunicaciones inalámbricas con otros motes, que dan cobertura a funciones de enrutamiento, como si de un *router* se tratara, ofreciendo la posibilidad de crear redes con topología estrella o multisalto. La Figura 17 muestra un nodo MicaZ

Figura 17 : Nodo sensor MicaZ

- El gateway que se va a usar en es el MIB520CB que proporciona conectividad USB con la estación base y soporta todos los nodos de la familia MICA. El MIB520CB ofrece dos puertos separados: uno dedicado para programar la mote y un segundo para la comunicación de datos con la estación base. Este gateway tiene una velocidad de transmisión en baudios de 57.6 Kbps. La Figura 18 muestra un gateway MIB520CB.

Figura 18: Gateway MIB520CB

- El medio por el que se van a transmitir paquetes entre los nodos sensores es un medio no guiado, por lo que se va a utilizar una radio para transmisión en el vacío de los mensajes. Se van a poder definir dos maneras de enrutar un paquete a través de la radio: una manera es punto a punto, conociendo la dirección del nodo destino y enviando el paquete dirigido a eso nodo; otra forma es enviar un mensaje a todos los nodos que puedan escuchar con sus radios dicho mensaje, esto de denomina un envío a *broadcast*. Se debe minimizar lo máximo posible el consumo de energía de los nodos por lo que el uso de la radio tiene que ser el mínimo posible.

- Para la comunicación entre el gateway maestro de la red y la estación base se va a utilizar un puerto USB.

- El protocolo de comunicación debe estar implementado en nesC sobre el sistema operativo TinyOS 1.1.15. Para la comunicación por radio entre nodos de la red se va a utilizar el protocolo de comunicación estándar que TinyOS y la mote MicaZ soportan, Zigbee.

- La estación base debe ser un ordenador personal con un sistema de almacenamiento *big endian.*

4.4 Limitaciones del proyecto

El protocolo STP presenta algunas limitaciones. En primer lugar, es necesario tener siempre presente que tanto los gateways como los nodos clientes tienen una vida limitada por la duración de las pilas, lo cual implica que cuando la energía de esas baterías se agote, ese mote quedará inutilizado. Además, la sustitución de la pila o el mote en caso de avería puede ser compleja dependiendo del escenario, ya que en algunos escenarios las redes de sensores inalámbricos se distribuyen en lugares de difícil acceso, donde otras tecnologías no pueden llegar.

En el caso de que un mensaje no llegue al nodo destino, el nodo origen no va a tener notificación del fallo y por lo tanto no va a reenviar el mensaje. Es decir, no se ha implementado mensajes de ACK explícitos. Por ese motivo los nodos que envían peticiones de sincronización lo hacen a todos los nodos accesibles, así la probabilidad de que ningún nodo le conteste es muy baja. Si el mensaje que contiene la hora de sincronización se pierde, el nodo que todavía no está sincronizado insistirá enviando mensajes de petición de sincronización.

Si el gateway no puede atender varias peticiones simultáneamente, es posible que se pierda la información de las peticiones de los nodos que no han podido ser atendidas.

5. Diseño

En este capítulo se abordará el diseño de las aplicaciones que conformarán el protocolo Synchronized Time Protocol (STP), basándose para ello en la especificación de requisitos que fue presentada en el capítulo anterior.

5.1 Arquitectura del sistema

El protocolo STP está pensado para poder sincronizar las motes de una red inalámbrica y de esta madera poder conocer con qué fecha y hora las motes están realizando mediciones.

Es este protocolo hay cuatro elementos básicos: el primer elemento es el PC del cual se va a obtener la hora, el segundo elemento es el gateway maestro (GM), encargado de intermediar entre el ordenador y los gateways esclavos de la red inalámbrica. Los gateways esclavos (GS) corresponden al tercer elemento y deben comunicarse con el gateway maestro, entre ellos mismos y con los nodos clientes. El último elemento son los nodos clientes o las motes que se encargan de hacer las mediciones.

En la Figura 19 se muestra de forma visual una disposición de una WSN y cada uno de los actores que intervienen en el protocolo STP.

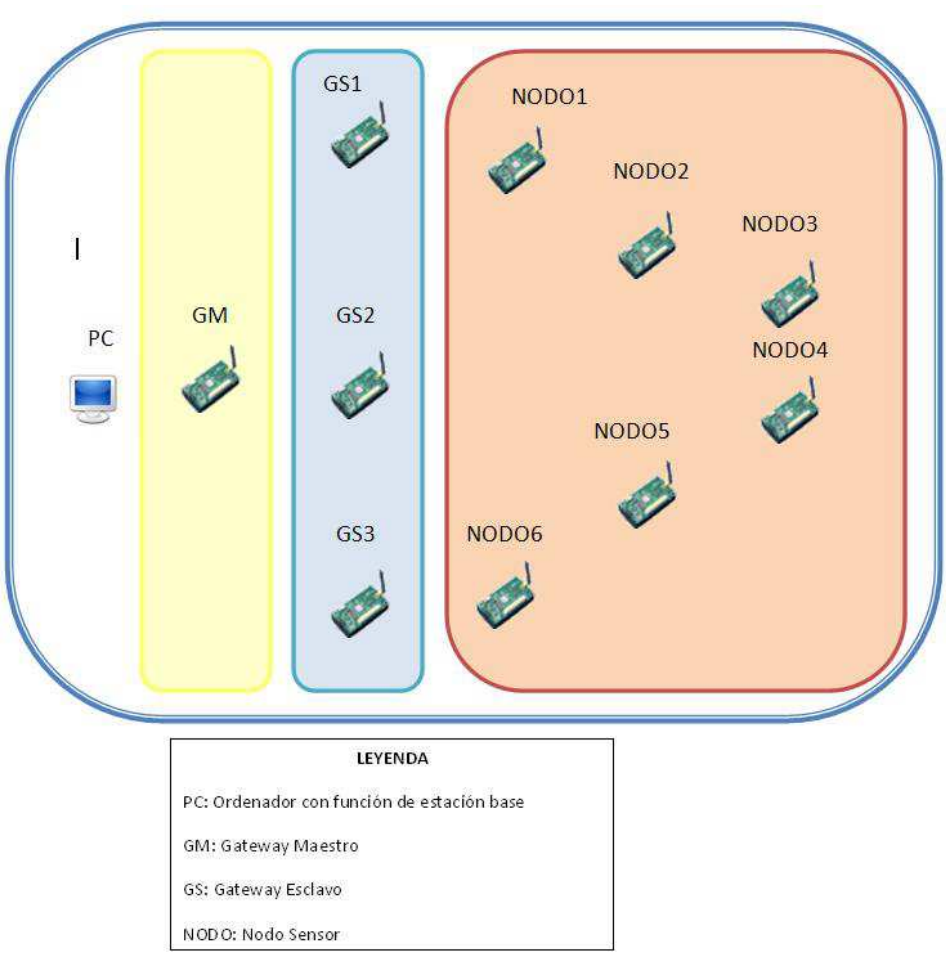

Figura 19: Jerarquía del protocolo

En el protocolo STP cada nodo de la red tiene que saber en cada momento que función tiene en la red. Por este motivo los nodos van a almacenar un rol. Los posibles roles son los siguientes:

- GM_MASTER si el nodo es el gateway maestro,
- SLAVE si el nodo va a realizar las funciones de gateway esclavo o
- NODE si el nodo va a ser un nodo sensor.

Además todos los nodos de la red van a almacenar hora, minuto y segundo en el formato hh:mm:ss

Para que en todo momento los nodos estén sincronizados va ser necesario emitir la hora con una frecuencia determinada. En el protocolo STP se va a transmitir la hora siempre que se reciba una petición de cualquier nodo de la red, en ese instante se contestará al mensaje de tipo "petición" con otro mensaje de tipo "respuesta" con la hora que el nodo tenga almacenada.

5.2 Gateway Maestro

El gateway maestro (GM) es el nombre que se atribuye al mote conectado con el PC, su función es actuar como pasarela entre ZigBee y TCP/IP. Se puede apreciar en la Figura 20 la apariencia de un gateway, además de ver que tiene incorporado un puerto USB.

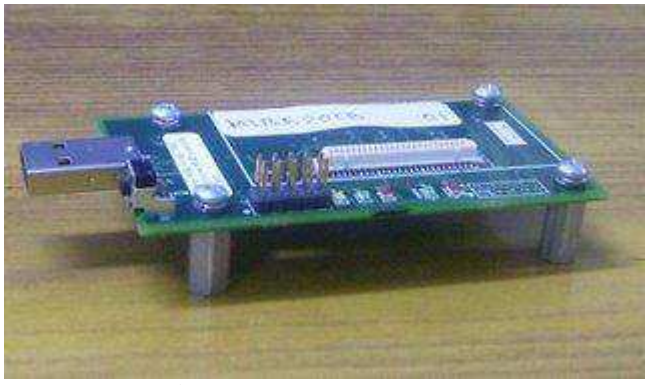

Figura 20: Gateway Maestro

Actualmente el protocolo STP está diseñado para que exista un único gateway maestro y siempre va a tener asignada la dirección 0x00 de la red de sensores. El GM debe conectarse con el PC a través de un puerto USB, y a través de la radio con los gateways esclavos.

El GM tiene dos funciones: la primera función básica del GM es comunicarse con el ordenador para obtener dos parámetros de configuración (el tiempo y el número de esclavos), y la segunda función es transmitir paquetes a los gateways esclavos.

Para poder realizar la transferencia entre el PC y el GM se va a necesitar una aplicación de bajo nivel que permita enviar a través de un puerto UBS de un ordenador personal los parámetros de configuración.

En la Figura 21 se puede ver un esquema de la comunicación entre el PC (estación base) y el gateway maestro: se puede observar como el PC y el gateway se comunican a través de un puerto USB, permitiendo que la comunicación sea bidireccional.

Dentro del gateway se han representado las dos funciones básicas de este dispositivo: La comunicación a través del puerto y la comunicación a través de la radio.

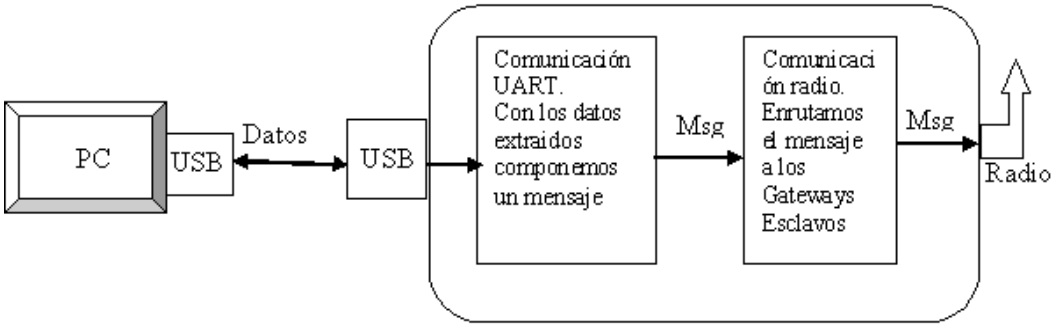

Figura 21: Comunicación PC-Gateway

El gateway maestro va a recibir de la estación base un parámetro que le indica el número de gateways esclavos que van a existir en la red y va a almacenar en una estructura interna las direcciones que van a tener estos gateways esclavos. De tal manera que si el usuario considera que hay 7 gateways esclavos en la red, el gateway maestro va a almacenar en un array la asociación de las direcciones como se muestra en la Tabla 3.

Tabla 3: Asociación GS-dirección en la red

Gateway Esclavo	GS-1	GS-2	GS-3	GS-4	GS-5	GS-6	GS-7
Dirección asociada	0x01	0x02	0x03	0x04	0x05	0x06	0x07

5.2.1 Comunicación vía UART

Se denomina comunicación UART a la transferencia de información que se realiza a través de un puerto serie de un PC. En el protocolo STP la comunicación UART se va a realizar a través de un puerto USB.

El principal objetivo de esta comunicación es poder informar al gateway maestro dos datos dinámicos:

- El primer parámetro es el número de gateways esclavos (GS) que van a existir en nuestra red, de manera que el usuario de STP puede adaptar el protocolo de acuerdo al número de motes de las que dispone y, por lo tanto, a sus necesidades. Para que el protocolo funcione correctamente es necesario que este parámetro sea igual o mayor que 1.

  En el paquete de envío al GM se va a dimensionar este parámetro a 1 byte por lo que el usuario no va poder definir este dato con un valor mayor de 255.

  Es importante realizar un buen análisis del valor de este dato, ya que si ponemos un número demasiado pequeño podemos ocasionar un cuello de botella en la comunicación entre los nodos clientes y los gateways esclavos, y esto puede producir un aumento de la latencia.

- El segundo parámetro es la hora del sistema. El GM debe recibir la hora que tiene el PC al que está conectado. Esta hora se transmitirá a todos los nodos de la red para conseguir una sincronización.

5.2.2 Comunicación vía Radio

Toda la comunicación entre los nodos sensores y los gateways esclavos se va a realizar a través de la radio mediante unos mensajes de petición y respuesta. Estos mensajes van a permitir que un nodo que no esté sincronizado pueda acceder a la red y obtener la hora del sistema.

La comunicación por radio del gateway maestro tiene la función de propagar la hora que se ha obtenido a través de la comunicación UART a todos los nodos de la red.

5.3 Gateway Esclavo

El gateway esclavo (GS) es el nombre que se atribuye al mote que se encarga de dirigir los mensajes de tiempo a los nodos sensores. Para realizar esta función los GS tienen que estar siempre escuchando en la red (modo promiscuo). En el instante en el que reciban un mensaje deben ser capaces de interpretar que deben hacer. Existen dos posibilidades:

- En el caso que reciba un mensaje proveniente del GM debe almacenar la hora almacenada en el paquete y reenviársela a todos los nodos de la red que sean alcanzables.

- En el caso que recibiera un mensaje de un nodo sensor cliente debe contestar al mensaje indicándole la hora que tiene almacenada o bien solicitarla al GM.

5.4 Nodos sensores clientes

Los nodos sensores son dispositivos autónomos que se distribuyen geográficamente alrededor de un fenómeno para poder monitorizarlo. Tienen capacidad para realizar mediciones, almacenar datos temporalmente y para comunicar información en una red conectada sin cables a través de una radio. Los nodos sensores tienen integrada una pequeña aplicación que les permite realizar tales funciones.

En una red los nodos sensores están diseminados de una forma más o menos aleatoria cubriendo la zona de estudio. Cada nodo se comunica con los dispositivos que tiene dentro de su radio de transmisión, haciendo que la información pueda acabar llegando al gateway, el cual a su vez entregará la información al dispositivo encargado de almacenar los datos, normalmente un ordenador personal.

En la Figura 22 se puede observar el aspecto de un nodo sensor. Se puede destacar en la parte inferior la batería, la cual alimenta de corriente la placa del nodo. Además se observa en la parte superior derecha la radio del mote.

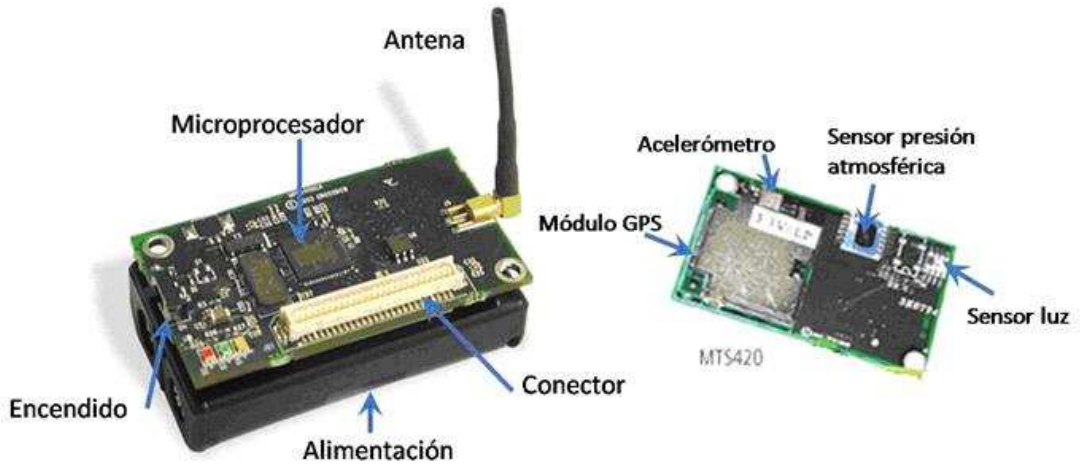

Figura 22: Nodo sensor y sus componentes

En el protocolo STP estos nodos actúan como clientes de un protocolo NTP y van a solicitar periódicamente la hora de sincronización a cualquier gateway esclavo que sea accesible por su radio. Los nodos sensores recibirán como respuesta un mensaje de sincronización desde los *gateway esclavos* a través de sus radios, por lo que cada vez que el nodo detecte un paquete en la red activará su radio, escuchará el mensaje y almacenará la hora recibida.

Cuando reciben el paquete de sincronización activa un cronómetro o temporizador que va contando marcha atrás hasta llegar a 0, de esta manera se podrá definir cada cuanto tiempo se quiere que el nodo solicite una nueva sincronización. Además para evitar que otros nodos sensores que no estén accesibles por gateways esclavos el nodo sensor que reciba un mensaje de sincronización lo va a reenviar a toda la red que pueda escucharle.

Para poder simular un reloj interno se va a usar un módulo que proporciona TinyOS el cual simula otro cronómetro cuenta atrás.

5.5 Comunicación PC-estación base

Una estación base es un equipo con mayores capacidades, normalmente un ordenador personal, aunque también puede ser un sistema embebido, que se encargará de recoger los datos que el gateway recogerá de la red de sensores.

En este caso, la estación base es un PC convencional que ejecutará una aplicación C desde la que se introducirán los parámetros de configuración del protocolo STP. Esta aplicación permitirá la comunicación a través de un puerto USB con el gateway maestro.

Para poder conseguir la hora del sistema se va a utilizar la librería de C `time.h` que permite usar funciones que devuelve en una estructura la hora local del sistema.

Para recibir por parámetro el número de gateway esclavos de la red se va a diseñar una interfaz que permite al analista que maneja la estación base introducir a través de un teclado el número de gateway esclavos que van a existir en la red. Una vez se recibe el número de gateways esclavos deseados en la red la estación base va a tener que generar y enviar un mensaje por cada uno de los gateways esclavos que van a existir en la red.

Tanto la hora como la dirección del gateway esclavo de la red se van a enviar a través del puerto USB del ordenador al USB del gateway maestro, para ello será necesario que el programa C

conozca los puertos que existen en la estación base, componga el paquete que se tiene que enviar al gateway, abra el puerto USB y escriba en él los mensajes. Una vez enviada la trama se debe cerrar el puerto USB para evitar problemas en futuros accesos.

TinyOs incluye una herramienta que realiza una comunicación entre un gateway conectado a un puerto USB y una estación base pero la comunicación se realiza en sentido contrario a la que se necesita en este desarrollo. Esa aplicación es serialForwarder [42] y permite el envío de información desde un nodo a la estación base.

5.5.1 SerialForwarder

El programa SerialForwarder se encarga para leer los datos de los paquetes que se reciben en un puerto serie de un ordenador para volcarlos sobre una conexión de Internet. Esto permite poder capturar los paquetes que envían los gateways a la estación base e interpretarlos con algún software que esté escuchando en la conexión TCP.

Este programa abre el puerto serie que especifica el usuario, y definir la velocidad de transmisión baudios; ambos puntos son necesarios para el correcto funcionamiento de serialSTP, la diferencia esencial entre ambos programas es que SerialForwarder escucha paquetes que el gateway envía.

SerialForwarder es un programa que se instala sobre TinyOS y se encuentra en la ruta `tinyOS/tools/java/net/tinyos/sf` como se muestra en la Figura 23.

Figura 23: Localización de SerialForwarder

En la imagen se puede apreciar que este software está implementado en Java y si se desea ejecutar se puede lanzar desde la ruta `tinyOS/tools/java` el siguiente comando:

```
java net.tinyos.sf.SerialForwarder
```

Cuando se ejecuta el comando se abre un interfaz gráfico como el que se muestra en la Figura 24.

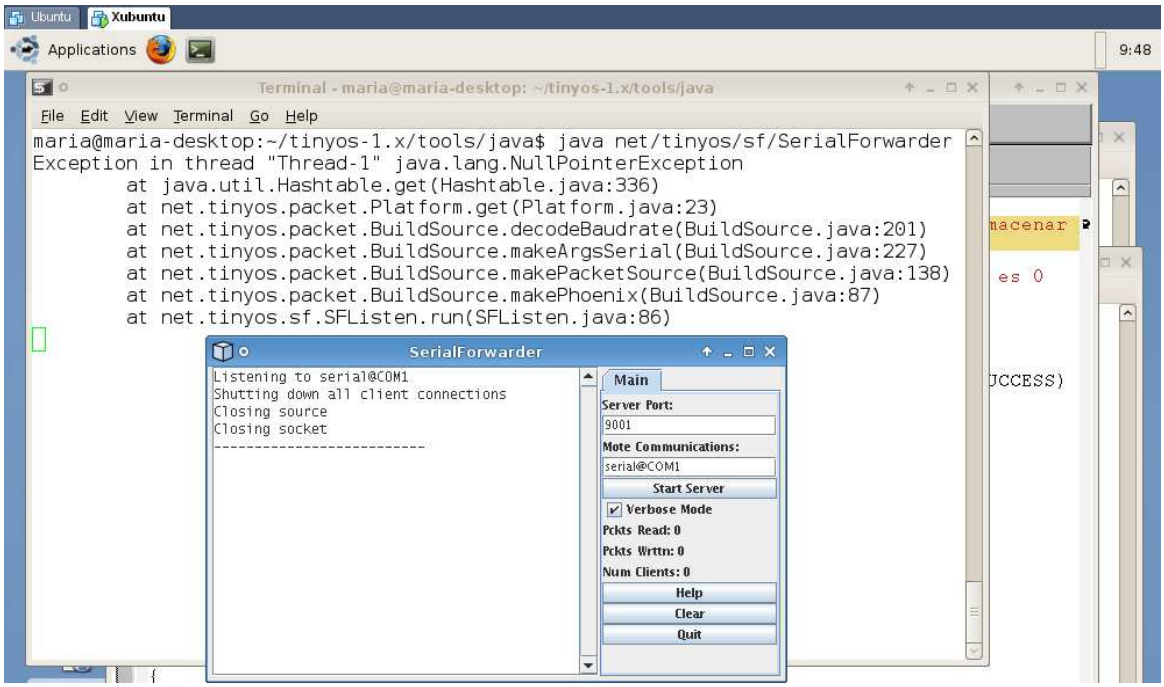

Figura 24: Ejecución SerialForwarder

SerialForwarder va a servir como referencia para la comunicación que permita el envío de información desde la estación base y el gateway maestro.

5.6 Descripción del protocolo STP

La sincronización de los nodos se realiza a través de mensajes de petición del tiempo y de respuesta.

La sincronización del gateway maestro se realizará cada que el usuario lo solicite, por otro lado cada nodo va solicitando periódicamente la hora de sincronización. Por último, cada nodo va a mantener la hora de sincronización que dispone.

En la Figura 25 se puede ver como al iniciarse un nodo se realiza una comprobación de su dirección en la red, si el nodo dispone de la dirección 0x00 se sabe que es el gateway maestro, si no se marca provisionalmente como nodo sensor hasta poder definir si es un nodo sensor o un gateway esclavo.

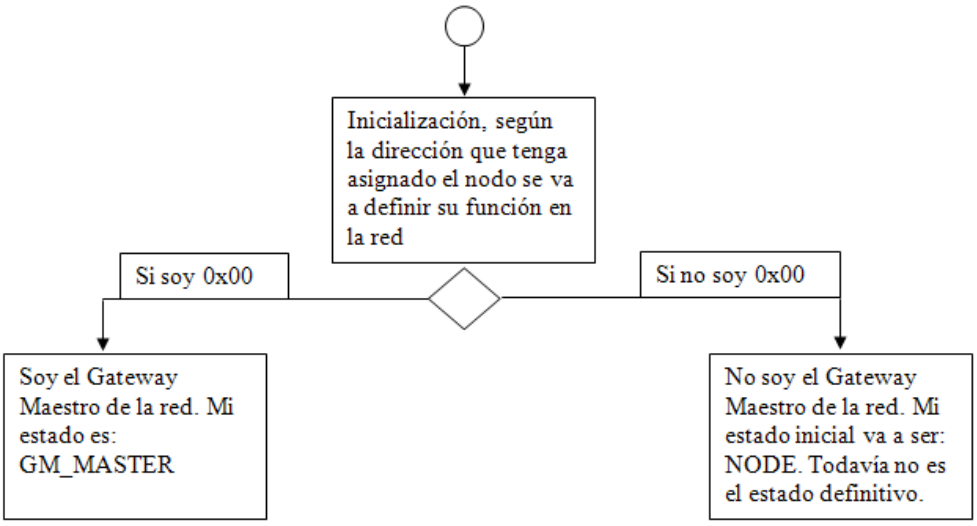

Figura 25: Inicialización de los nodos

5.6.1 Comportamiento de un gateway maestro

El gateway maestro de la red va a tener dos funciones en la sincronización de la red, en primer lugar debe identificar a los gateways esclavo de la red una vez haya recibido los parámetros de configuración desde UART, y debe atender a los mensajes que reciba desde otros nodos.

En la Figura 26 se muestra el diagrama de comunicación que realiza un gateway maestro según va recibiendo mensajes. Se hace distinción entre los paquetes recibidos por radio o por el puerto USB.

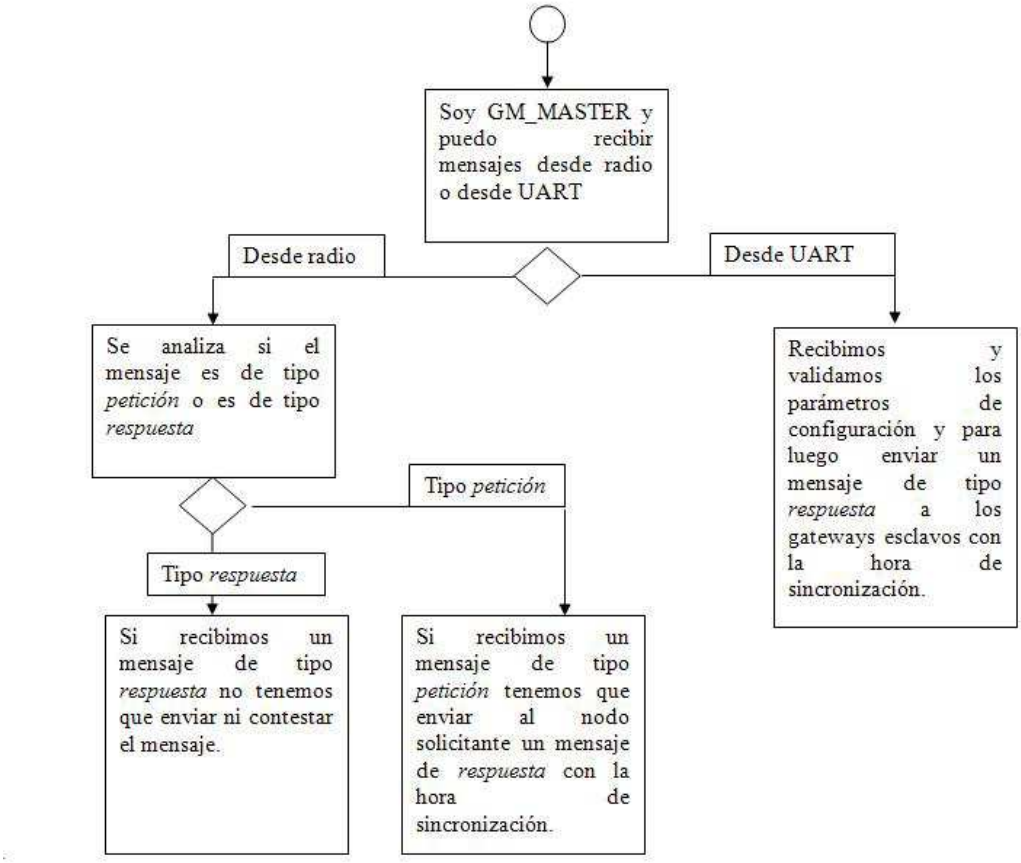

Figura 26: Diagrama de comunicación del GM

5.6.2 Comportamiento de un gateway esclavo

Una vez identificado un gateway maestro se pueden identificar los gateway esclavos, ya que son los nodos que reciban un mensaje con la dirección de origen 0x00 y que sea de tipo respuesta.

En la Figura 27 se muestra el diagrama de comunicación que realiza un gateway esclavo según va recibiendo mensajes.

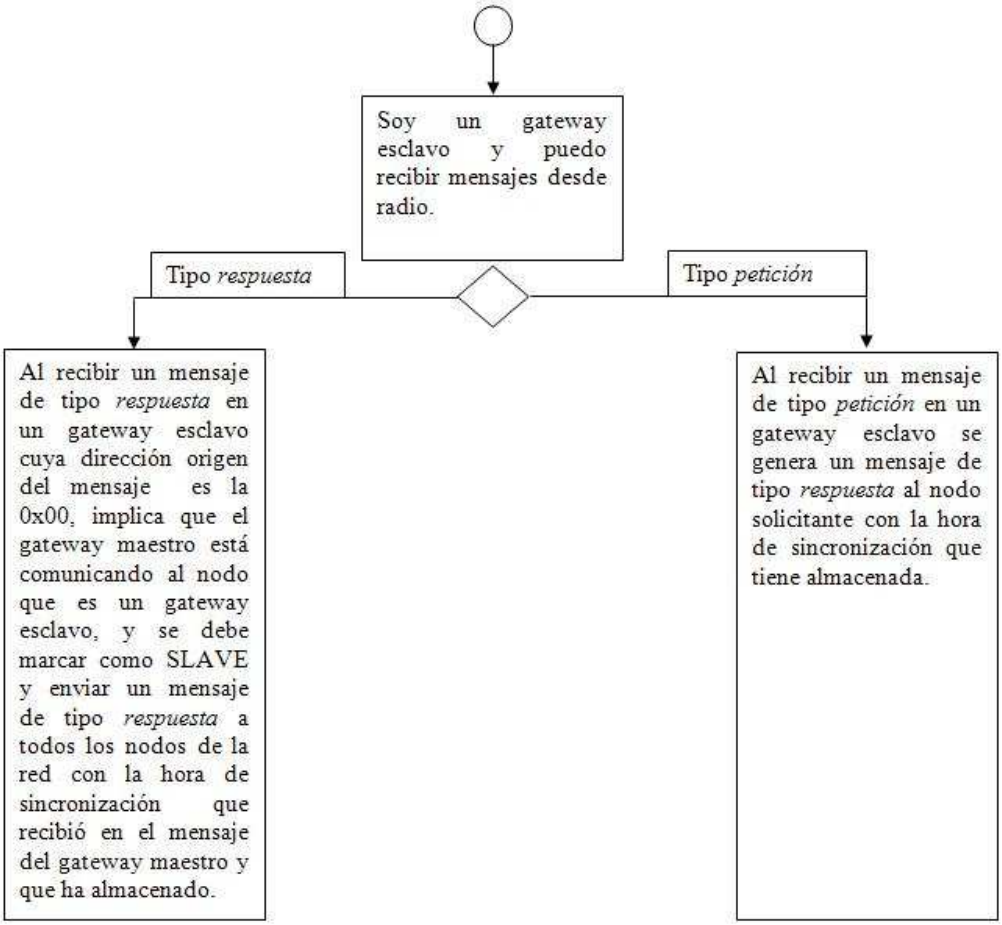

Figura 27: Diagrama comunicación de GS

5.6.3 Comportamiento de un nodo sensor

Los nodos sensores son los únicos nodos de la red que pueden enviar mensajes de tipo petición sin necesidad de haber habido un mensaje anterior; estos nodos deben solicitar la hora de sincronización de forma periódica para estar actualizados.

En la Figura 28 se muestra el funcionamiento de los nodos sensores en la solicitud de sincronización:

Figura 28: Diagrama de solicitud de sincronización de un nodo sensor

Para garantizar que todos los nodos sensores están sincronizados, cada nodo reenvía el mensaje de sincronización recibido.

En la Figura 29 se muestra el diagrama de comunicación que realiza un nodo sensor según el tipo de mensaje que recibe.

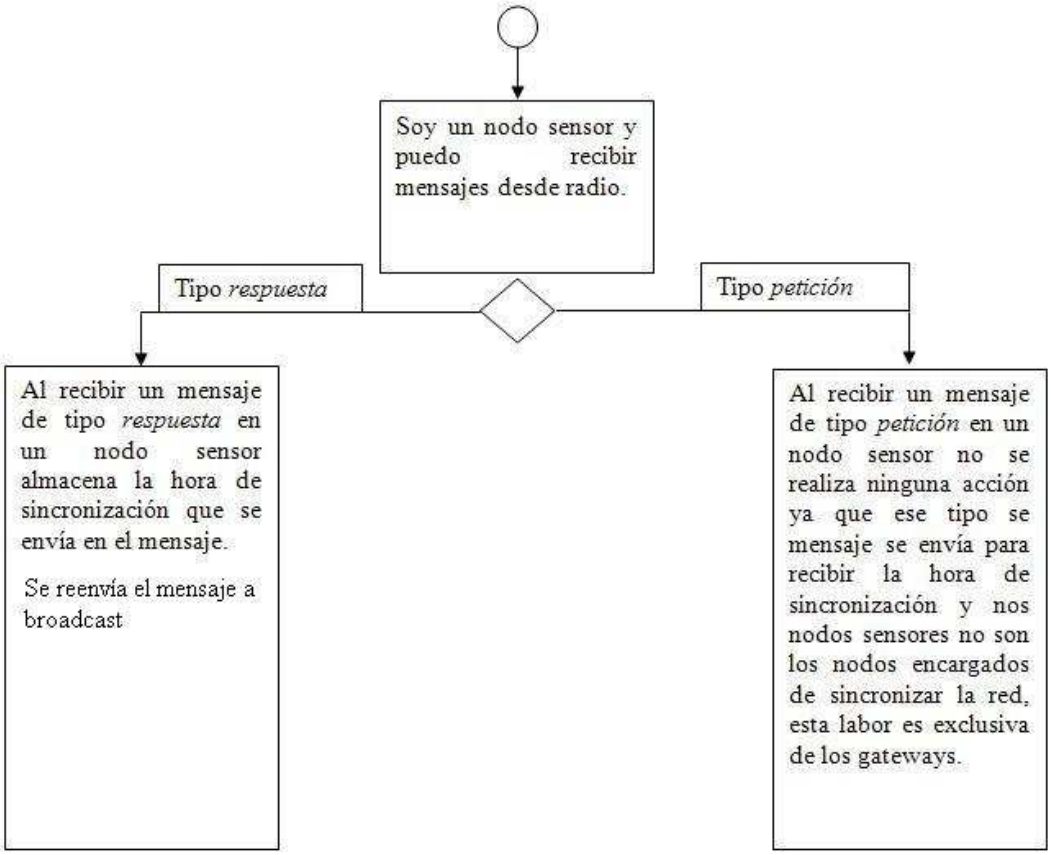

Figura 29: Diagrama de comunicación de un nodo

5.6.4 Envío

Si se desea ver un esquema general de cómo interactúan todos los nodos de la red se puede analizar por un lado el envío y por otro la recepción de mensajes. A continuación se muestra un esquema de los envíos de paquetes en la red.

Todos los nodos de la red envían paquetes para solicitar la hora del sistema, por lo que todos esos mensajes tienen la función de peticiones.

En la Figura 30 se muestra el diagrama de las interacciones que se producen entre los distintos participantes cuando un nodo cliente solicita un mensaje de sincronización.

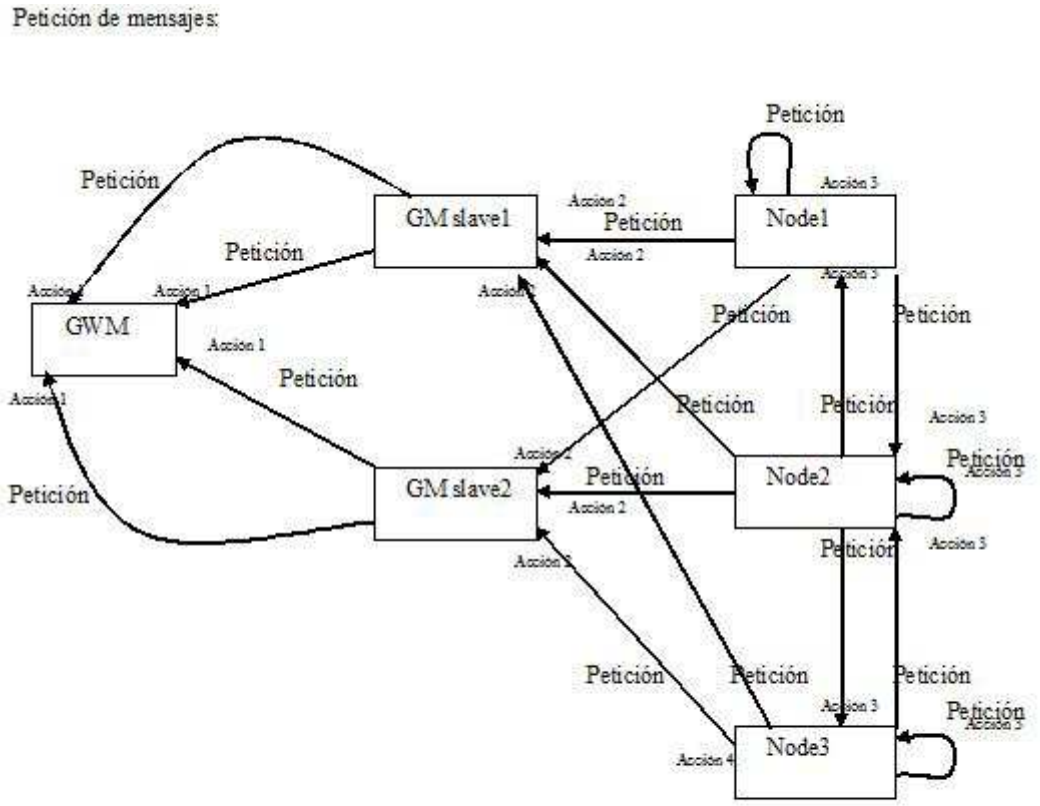

Figura 30: Petición de mensajes en STP

A continuación se describe en qué consiste cada una de las acciones que se muestran en el diagrama de envío del protocolo STP:

- Acción 1: Se origina cuando el gateway maestro recibe una petición de un gateway esclavo, en consecuencia responde con la hora que tiene almacenada.

- Acción 2: Se origina cuando un gateway esclavo recibe una petición de un nodo que no es gateway maestro ni esclavo, en ese momento el gateway envía una respuesta con la hora que tiene almacenada.

- Acción 3: Se origina cuando un nodo sensor recibe un mensaje de tipo petición de otro nodo sensor, el nodo debe ignorar el mensaje ya que es un nodo sensor y no un gateway esclavo y no debe enviar la hora de sincronización.

Lo primero que debería pasar es que el cliente solicita petición (acción 1). El GS mira a ver si la tiene y sino la solicita al GM (acción 2), etc.

5.6.5 Recepción

En la Figura 31 se muestra el diagrama de comunicación que se producen cuando cada uno de los actores de la red recibe un mensaje.

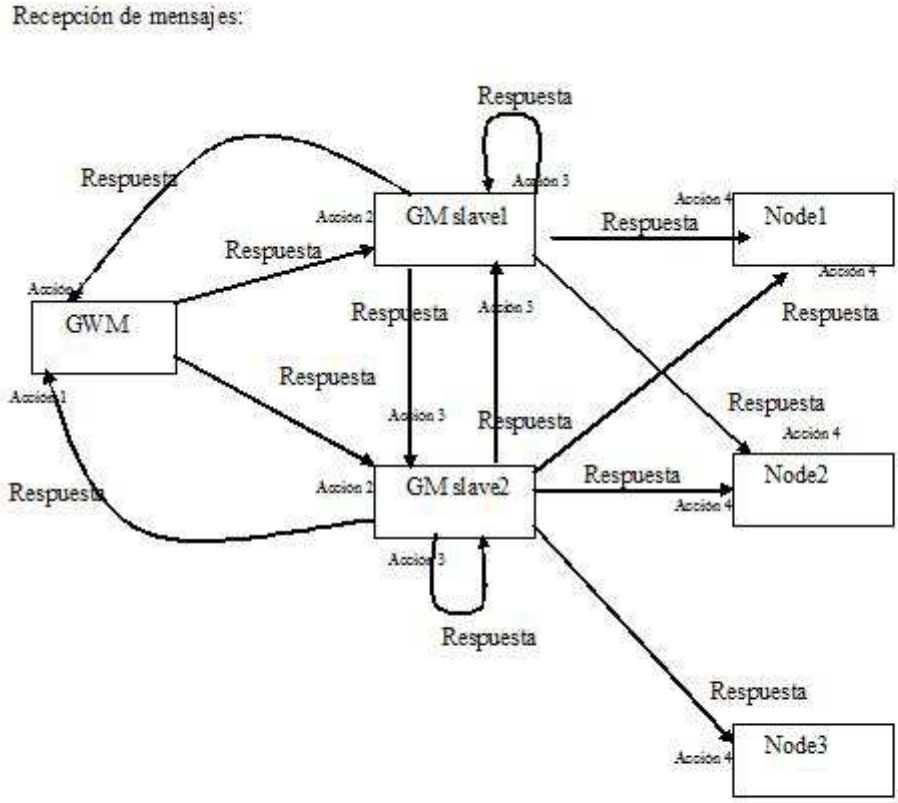

Figura 31: Recepción de mensajes en STP

A continuación se describe en qué consiste cada una de las acciones que se muestran en el diagrama de recepción del protocolo STP:

- Acción 1: Se origina cuando el gateway maestro recibe un mensaje de respuesta de un nodo, cuando ocurra esto el gateway maestro debe ignorar el mensaje.

- Acción 2: Se origina cuando un gateway esclavo recibe un mensaje del gateway maestro, en ese momento el nodo se reconoce como enrutador, almacena la hora que el gateway maestro le comunica y envía un mensaje a broadcast con esa información.

- Acción 3: Se origina cuando un gateway esclavo recibe un mensaje de otro gateway esclavo incluido recibir un mensaje de él mismo, cuando ocurra esto el gateway debe ignorar el mensaje.

- Acción 4: Se origina cuando un nodo que no es gateway maestro ni esclavo recibe un mensaje de un gateway esclavo cuando ocurra esto el nodo debe marcarse como nodo y almacenar la hora que le han enviado y reenviarla a broadcast con un mensaje de tipo respuesta.

5.7 Componentes que conforman STP

TinyOS incluye muchos componentes a la disposición del programador. En la Figura 32 se muestran los componentes contenidos en la versión 1.0 de TinyOS:

TinyOS 1.0 System Components	
BapM.nc	Beaconless ad-hoc routing protocol
Counter.nc	Continually incrementing counter
IdentC.nc	Component to retrieve and set node identity
IntToLedsM.nc	Display an integer value on the LEDs
IntToRfmM.nc	Report an integer value over the radio
OscopeM.nc	Component that continually samples and reports and ADC channel
ResetC.nc	System reset component
RfmToIntM.nc	Receive and integer over the radio
Route	Multi-hop routing suite
SenseToInt.nc	Generate and iteger valued sensor reading from the ADC
TinyAlloc.nc	Memory allocator
TinyDB	Database application for TinyOS
ADCM.nc	Interface to the ADC
AMStandard.nc	Active Messages implemetnation
ByteEEPROM.nc	EEProm access componet
CRCPacket.nc	CRC packet calculator
ClockC.nc	Timing componet
CrcFilter.nc	CRC packet filter
GenericComm.nc	Generic communication stack for general use
I2CPacketC.nc	I2C protocol implementation
LedsC.nc	LED interface
LoggerM.nc	Interface to log data to the off-chip flash
NoCRCPacket.nc	Packet componet without CRC calculation
PotM.nc	Signal strength control
RFM.nc	TR1000 Radio interface
RadioCRCPacket.nc	TR1000 Packet interface with CRC
RadioNoCRCPacket.nc	TR1000 Packet interface without CRC
RandomLFSR.nc	Random Number Generator
SecDedRadioByteSignal.nc	Forward error correction component
TimerM.nc	Multi-application timer module
UARTComm.nc	UART communicaitons stack
UART.nc	UART interface module
UARTNoCRCPacket.nc	UART communication stack without CRC
VoltageM.nc	Battery voltage measurement component

Figura 32: Componentes TinyOS 1.x

A continuación se van a describir los componentes e interfaces de TinyOs 1.x que se van a utilizar en la implementación del protocolo Synchronized Time Protocol.

5.7.1 Interfaz StdControl

La interfaz *StdControl* es una interfaz especial que debe ser incluida en toda implementación para TinyOS. Permite el suministro de energía a los componentes del sensor. Esta interfaz obliga a implementar los siguientes métodos:

- *command result_t init():* Este método se va a invocar cuando el nodo sensor arranque, esto es, cuando reciba energía.

- *command result_t start():* Este método se va a invocar después de la invocación del método *init()* y cuando el sensor pase de estado apagado (off) a encendido (on).

- *command result_t stop():* Este método se va a invocar cuando el sensor pase de encendido (on) a estado apagado (off) .

5.7.2 Interfaz SendMsg

Esta interfaz proporciona mecanismos para poder enviar mensajes a otros sensores o al puerto serie del ordenador. Obliga a implementar los siguientes métodos:

- *command result_t send (uint16_t address, uint8_t length, TOS_MsgPtr msg):*
  Este método realiza el envío de un mensaje a la dirección *address*; dicho mensaje es *msg* de tipo TOS_MsgPtr, el cual es un puntero a una estructura TOS_Msg, y deberá de ser global, ya que si se declara local, la duración de ese mensaje solo será hasta que la función termine y cuando esto ocurre, todavía no se ha enviado el mensaje).

El componente que utilice esta interfaz deberá de implementar los siguientes eventos:

- *event result_t sendDone (TOS_MsgPtr msg, result_t success)*
  Este evento se va a producir en el caso de que se produzca el envío de un mensaje a través de la radio. Si el envío ha sido correcto se devuelve un SUCCESS y si ha fallado se devuelve un FAIL.

5.7.3 Interfaz ReceiveMsg

Esta interfaz proporciona los mecanismos necesarios para poder recibir un paquete en el componente que la utilice. No obliga a implementar ningún método, sin embargo cualquier componente que haga uso de dicha interfaz, se verá obligado a implementar los siguientes eventos:

- *event TOS_MsgPtr receive (TOS_MsgPtr m)*
  Este evento se va a producir ante la llegada de un paquete y ha de devolver al final de su invocación el mismo paquete que se ha recibido para poder pasarlo a capas de nivel superior.

5.7.4 Interfaz Timer

Esta interfaz proporciona mecanismos de temporización. Para ello obliga a implementar los siguientes métodos:

- *command result_t start(char type, uint32_t interval)*
  Este método produce la inicialización o el arranque de la temporización. El parámetro type sirve para determinar si es una temporización continua, es decir, que se repite de forma indefinida cada intervalo de segundos (tomará el valor TIMER_REPEAT), o por el contrario si es un timer eventual que sólo se invocará una vez, cuando haya transcurrido el tiempo (tomará el valor TIMER_ONE_SHOT).

- *command result_t stop()*
  Este método produce la parada de la temporización del timer del reloj.
  El componente que haga uso de esta interfaz, está obligado a implementar el evento *fired*

- *event result_t fired()*

Este evento se producirá cuando haya transcurrido el intervalo especificado en el parámetro del *timer*.

5.7.5 Componente Main

Este es un componente especial que representa el cuerpo *main* al estilo de un programa escrito en C y que necesita una interfaz StdControl para su funcionamiento; dicha interfaz será proporcionada por el componente que quiera convertirse en una aplicación.

5.7.6 Componente GenericCommPromiscuous

Este componente proporciona mecanismos para poder enviar y recibir paquetes vía radio y vía puerto serie. Utiliza el modelo de *Active Messages (AM)* estándar de TinyOS, por lo que se basa en el envío de paquetes TOSMsg; estos paquetes poseen la siguiente estructura:

- uint16_t addr: Es la dirección a la que va destinado el paquete, existen ciertas direcciones especiales:

  TOS_BCAST_ADDR: Es la dirección de broadcast.
  TOS_LOCAL_ADDRESS: Es la dirección local (localhost).
  TOS_UART_ADDR: Es la dirección del puerto serie.

- uint8_t type: Indica el tipo de paquete que se está enviando.
- uint8_t group: Es el grupo al que pertenece el sensor que está enviando el paquete, de manera que sólo lo reciban aquellos sensores que pertenezcan al mismo grupo.
- uint8_t length: Indica la longitud total del paquete.
- int8_t data [TOSH_DATA_LENGTH]: Datos o payload del paquete.
- uint16_t crc: Código que se usa para validar la integridad del paquete.

5.7.7 Componente QueuedSend

En TinyOs cuando se envía un mensaje a través de la radio, no se hace de manera directa, sino que se almacena en una cola de tipo FIFO. Este componente proporciona los mecanismos necesarios para el control de cola de los mensajes a enviar, todo ello de manera transparente al desarrollador. Para ello, provee las interfaces:

- *StdControl*
- *SendMsg*
- *QueueControl*

En la Figura 33, muestra el gráfico de componentes de QueuedSend.

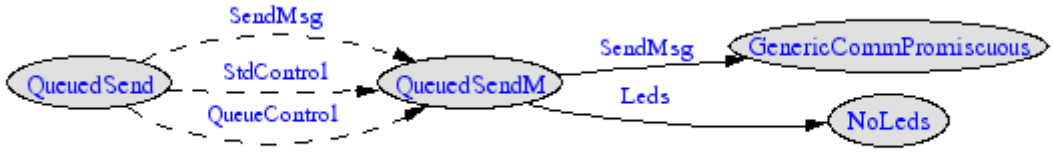

Figura 33: Componente QueuesSendM

5.7.8 Componente TimerC

Este componente proporciona los mecanismos necesarios para poder llevar a cabo funciones de temporización de hasta 10 timer diferentes. Para ello proporciona las siguientes interfaces:

- Timer[id]: Esta interfaz permite tener múltiples timers diferentes, y para ello se utiliza el parámetro id. Para hacer el wiring de esta interfaz se suele utilizar la función que proporciona el compilador *unique*, la cual proporciona un identificador único.

- StdControl: En la interfaz *StdControl* que proporcione el componente que estamos desarrollando, tendremos que incluir llamadas a las correspondientes funciones de esta interfaz para que la aplicación TimerC funcione correctamente.

5.8 Estructura de los mensajes del protocolo

A continuación se va a definir las estructuras de los mensajes del protocolo STP. Se van a definir dos estructuras debido a que se van a utilizar dos tipos de mensajes: UART y radio. La primera comunicación que se va a analizar es la que depende del envío de los mensajes a través de las radios de las motes.

La Figura 34 muestra la estructura estándar de los mensajes que se van a enrutar en nuestra red a través de la radio.

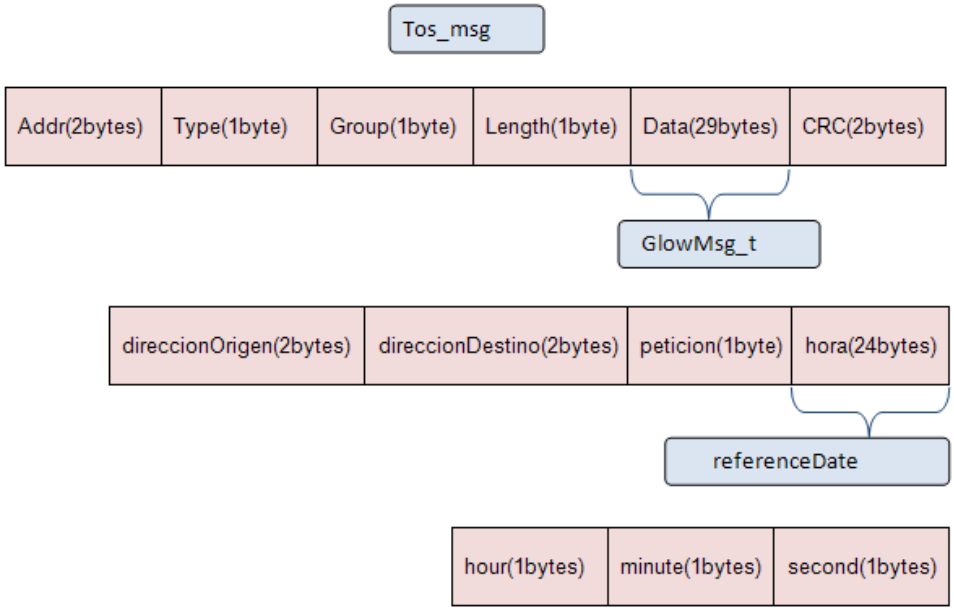

Figura 34: Estructura de la trama del paquete de la WSN

Dicha estructura contiene los datos específicos de cabecera y encapsulado de los protocolos de TinyOS añadiendo los datos que se necesitan en la aplicación. El primer bloque que aparece en la parte superior de la figura es la cabecera de Tos_msg El significado de cada uno de los campos es el siguiente:

- *addr*: contiene la dirección de destino del paquete. Existen tres posibilidades:
    - Dirección Broadcast (0xFFFF): indica que se debe transmitir a todos los nodos.
    - Dirección UART (0x007e): indica que se debe transmitir desde un nodo al puerto serie del gateway.
    - Dirección de nodo: se utiliza en el caso de querer enviar a un único nodo.
- *type*: es un número que identifica el tipo de mensaje. Típicamente cada aplicación tendrá su propio tipo de mensaje activo. Esto posibilita que el canal pueda compartirse por distintas aplicaciones identificadas mediante *type*.
- *group*: se trata de un identificador único para todos los motes que participan en la red. El valor por defecto es 125 (0x7d). Sólo los motes con el mismo número de grupo podrán comunicarse entre sí.
- *length*: indica la longitud en bytes del *payload*.
- data (*payload*): son los datos específicos de la aplicación. Aquí es donde se van a insertar los datos de sincronización.
- *CRC*: este campo identifica la integridad del mensaje.

En el campo *data* es donde se tiene que definir la estructura concreta del protocolo STP, que consta de los siguientes campos

- *direccionOrigen*: Para poder transmitir paquetes por la red sin tener que consultar las cabeceras de TinyOS se va a definir la dirección del nodo donde se originó el mensaje.

- *direccionDestino*: En este campo se va a almacenar la dirección del nodo destinatario, es decir, el nodo que tiene que recibir el mensaje.

- *Petición*: Este campo es un booleano, por lo que sólo puede tomar los valores verdadero o falso. Si toma valor 0 o falso implica que el mensaje no es una petición sino una respuesta. Si toma el valor 1 o verdadero implica que el mensaje es una petición o solicitud.

- *Hora*: estructura donde se va a almacenar el tiempo del sistema y que contiene los siguientes tres campos:

    - *Hour*: campo entero donde se almacena la hora.
    - *Minute*: campo entero donde se almacenan los minutos.
    - *Second*: campo entero donde se almacenan los segundos.

Además se tiene que tener en cuenta que todos los mensajes de TinyOS empiezan y terminan con un byte de sincronización. A ese byte se le denomina *sync* y siempre va a tomar el valor hexadecimal 0x7E.

Además de esta estructura, se ha de definir otra estructura de mensaje para la comunicación que se va a realizar entre la estación base y el gateway maestro. Este mensaje se va a componer en la estación base y luego se va a enviar al gateway maestro conectado por USB.

Para que es GM pueda entender los datos que se le envían se tiene que definir un mensaje con las cabeceras de TinyOS, ya que si no fuera así la mote no podría establecer donde empieza y termina el *payload* .

Si tanto el sistema que envía la trama como el sistema que la recibe tuvieran configurado el mismo formato al almacenar más de un byte, la transferencia podría hacerse directamente. Pero como se vio en apartados anteriores esto no se cumple, ya que en este caso las motes MicaZ son dispositivos *little endian* mientras que los Pc's convencionales Intel son *big endian.* Por lo tanto, si se quisiera trasmitir el valor hexadecimal 0x4A3B2C1D en el sistema *little-endian* el valor se codificaría como {1D, 2C, 3B, 4A} y en cambio en el sistema *big-endian* se codificaría en memoria en la secuencia {4A, 3B, 2C, 1D}; por este motivo en todos las cabeceras y datos que se vayan a enviar con más de un byte se va a tener que establecer el orden correcto en la trasferencia para que los dispositivos obtengan el mismo valor.

Por este motivo se va a definir en la trama la parte alta de las variables de más de un byte. Quedando la estructura de la trama como se muestra en la Figura 35.

Sync	Address (Parte Alta)	Address (Parte Baja)	Type	Group	Lenght	Param (Hex)	Hora (Hex)	Minuto (Hex)	Segundo (Hex)	CRC (Parte Alta)	CRC (Parte Baja)	Sync

Figura 35: Estructura de la trama de la comunicación PC-UART

En la figura podemos apreciar en color rosado los campos estándar del paquete en TinyOS y en color amarillo podemos apreciar los campos que se han definido para este proyecto en el *payload*. Cada uno de los campos mostrados representa un byte. El paquete siempre empieza y

acaba con los bytes de sincronización o sync que van a tomar en valor 0x7E. A continuación se describen cada uno de ellos.

- El campo *Address* ha sido dividido para enviar la parte alta de la variable y la parte baja. Como la dirección para las comunicaciones UART es la 0x007E, en la variable correspondiente a la parte alta se va a enviar 0x00 y en la variable correspondiente a la parte baja se va a enviar 0x7E.

- El campo *Type* va a tomar el valor 0x42 al igual que en el paquete TinyOS y el campo Group el valor 0x7D.

- El campo *Length* va a tomar el valor 0x04 ya que es la longitud del payload. El payload del paquete está formado por el campo *param*, *hora*, *minuto* y *segundo* que tiene un tamaño de un byte y que sumado definen el tamaño del payload a 4 bytes.

- El valor máximo que puede almacenar el campo *Param* es 0xFF ya que es el máximo valor que se puede representar con un byte y el valor mínimo es 0x01 ya que siempre debe haber al menos un gateway esclavo en la red.

- El campo *Hora* puede tomar valores entre 0x00 y 0x17, ya que el campo hora sólo puede oscilar entre las 0 horas y las 23 horas.

- Los campos *Minuto* y *Segundo* pueden tomar valores entre el 0x00 y el 0x3B, ya que ambos datos sólo pueden tomar valores reales entre el 0 y el 59.

- El campo *CRC* también debe ser divido para enviar la parte alta y posteriormente la parte Baja de la variable.

6. Implementación

En este capítulo se aborda la implementación del protocolo Synchronized Time Protocol, basándose para ello en la especificación de requisitos que fue presentada en el capítulo cuarto de esta memoria.

Se tiene que realizar una implementación específica para cada uno de los actores del protocolo.

Para la estación base es necesario implementar un programa que permita establecer una interfaz a través del cual el usuario introducirá el número de gateways esclavos que van a existir en la red, además este software permitirá obtener la hora del sistema y enviar estos datos al gateway maestro que deberá estar conectado al PC a través de un puerto USB.

Para el gateway maestro se van a necesitar dos funcionalidades: una que permita escuchar desde el puerto USB de la mote los datos que la estación base le está transmitiendo; y otra funcionalidad que permita la comunicación entre el GM y el resto de motes de la red para poder propagar la hora de sincronización. Además el gateway maestro necesita almacenar es un listado los gateways esclavos que están accesibles para él, de esta manera el gateway maestro siempre va a saber que nodos son gateways esclavos y por lo tanto son nodos que deberían estar siempre sincronizados.

En los gateways esclavos es necesario una implementación que permita almacenar la hora que le hayan enviado el GM y que permita enviar dicha hora través de la radio de la mote al resto de nodos de la red.

Los nodos esclavos necesitan almacenar la hora que le enrutan los gateways de la red.

Para poder cubrir con toda esta funcionalidad ha sido necesario realizar dos software que sean independientes pero que puedan colaborar entre sí. Un programa se encargará de la comunicación entre la estación base y el gateway maestro y otro programa será el protocolos de comunicación de la red.

6.1 Comunicación entre el PC y la estación base

Para poder comunicar el gateway maestro con la estación base, se necesita conectar el PC al gateway a través del puerto USB.

Para poder realizar esta labor va a ser necesario implementar un programa en C que sea ejecutado por el usuario a través de un intérprete de comandos. Este intérprete de comandos puede ser una versión del programa Shell en el sistema operativo Linux, o bien un emulador de Linux como Cygwin si se usa el sistema operativo Windows. El objetivo del programa será establecer una interfaz de comunicación entre el PC y la estación base, a través de un puerto del PC enviar los datos que requiere la red WSN para operar.

6.1.1 Argumentos que se envían al gateway maestro

Como ya se ha comentado la estación base propaga su hora del sistema a los gateways maestros. Para poder obtener la hora del sistema de la estación base se va a utilizar la función `localtime`

que está incluida en la librería `time.h` de C. Esta función permite almacenar en una estructura "tm" la hora actual del sistema.

`struct tm* localtime ( const time_t* timer ): Convertir un valor de tiempo de time_t a una estructura tm como hora local.`

La ventaja que da la estructura `tm` es que permite almacenar de forma independiente los datos del reloj, y de esta manera, permite manejar fácilmente los valores que se necesitan para la sincronización de la red.

En la Tabla 4 se muestran todos los campos que almacena la estructura `tm`. En la implementación del protocolo sólo se va a usar la hora, los minutos y los segundos, ya que el resto de datos no se van a transmitir a través de la red de nodos. La transferencia de datos a través de la radio de los nodos es muy costosa energéticamente hablando por lo que se pretende minimizar el volumen de mensajes de la red y el tamaño de los mismo. En los requisitos del protocolo se requería la sincronización a través de la hora, minuto y segundo por lo que de esta estructura se va a utilizar los tres primeros campos.

Tabla 4: Estructura tm

Miembros	Significado	Rango
tm_sec tm_sec	Segundos	0-59 0-59
tm_min tm_min	Minutos	0-59 0-59
tm_hour tm_hour	Horas (formato 24 horas)	0-23 0-23
tm_mday tm_mday	Día del mes	1-31 1-31
tm_mon tm_mon	Meses desde enero de	0-11 0-11
tm_year tm_year	Años desde 1900	
tm_wday tm_wday	Días desde el domingo	0-6 0-6
tm_yday tm_yday	Días desde Enero 1	0-365 0-365
tm_isdst tm_isdst	Marca para el horario de verano	

Por otro lado, existe otro requisito del proyecto que permite al usuario de la aplicación definir de forma dinámica el número de gateways esclavos que van a existir en la red. Para poder atender a ese requisito se tiene que definir una interfaz a través de código para que el usuario de forma intuitiva pueda proporcionar ese dato y configurar la red.

6.1.2 SerialSTP

Para realizar el programa que permite enviar datos desde un ordenador personal a través de un puerto USB a un gateway se ha tomado como referencia el programa SerialForwarder [42], que viene incluido en el sistema operativo TinyOS y su funcionamiento se comentó en el capítulo anterior.

Dicho programa se ha modificado para adaptarlo a los requerimientos ya que originalmente su función es recoger en la estación base la información que envía un gateway conectado por USB a dicha estación base. Pero la funcionalidad que se necesita para este proyecto es una comunicación en sentido inverso, es la estación base la que ha de enviar información al gateway maestro.

El programa que se ha implementado para cubrir la funcionalidad descrita anteriormente se denomina *serialSTP*.

Este software está compuesto de tres componentes:

- serial.c: Se encarga de la implementación de los módulos que va a utilizar el programa principal.
- serial.h: Se encarga de la definición de librería y funciones.
- serialSTP.c: Es el programa principal, gestiona las llamadas a los módulos, la recepción de parámetros por pantalla y el control de errores.

Y cubre las siguientes funcionalidades:

- Apertura del puerto definido por el usuario, si no es posible abrir el puerto indica un error por pantalla. Esta función se define de la siguiente manera:

```
int  serial_open(char *serial_name, speed_t baud);
```

- Escritura de un mensaje en un puerto ya abierto. Si la escritura no es posible se devuelve un error. Esta función se define de la siguiente manera:

```
void serial_send(int serial_fd, char *data, int size);
```

- Cierra el puerto que previamente ha sido abierto por la función serial_open. Esta función se define de la siguiente manera:

```
void serial_close(int fd);
```

- Control de la velocidad para la transferencia de datos a través de un puerto. Si los baudios introducidos en el programa no son correctos se devuelve un error. Esta función se define de la siguiente manera:

```
static tcflag_t parse_baudrate(int requested);
```

- Cálculo del código de seguridad CRC (comprobación de redundancia cíclica) que permite garantizar que los datos no se han alterado en la transferencia.

```
int calcrc(int ptr, int count);
int calcrc_paquet(char *ptr, int index, int count);
```

- Escritura de un log del programa.

```
void escritorLog(char *Buffer, int size);
```

Cuando se ejecuta el programa SerialSTP se solicitan como parámetro los siguientes datos: la ruta y el nombre del puerto que se quiere abrir y al que tiene que estar conectado el gateway maestro, además del ratio de datos (baudios) de transmisión.

Como se muestra en la Figura 36 este programa está documentado y explica en su interior cual es su funcionalidad, el modo de uso y el nombre de los dispositivos de puerto series.

```
/*****************************************************************************/
/* serialSTP.c                                                               */
/*---------------------------------------------------------------------------*/
/* Serial communications in Linux whith a wireless                           */
/*---------------------------------------------------------------------------*/
/* This program sends a paquet to the serial port.                           */
/* The serial device name should be passed as a parameter                    */
/* The serial port speed is configured by the user                           */
/*---------------------------------------------------------------------------*/
/* Example of use:                                                           */
/*                                                                           */
/*   ./serialSTP /dev/ttyUSB0 57600                                          */
/*                                                                           */
/*---------------------------------------------------------------------------*/
/*  In linux, the serial devices names are:                                  */
/*                                                                           */
/*    /dev/ttyS0  --> First native serial port                               */
/*    /dev/ttyS1  --> Second native serial port                              */
/*    ...                                                                    */
/*    /dev/ttyUSB0  --> First USB-RS232 converter                            */
/*    /dev/ttyUSB1  --> Second USB-RS232 converter                           */
/*    ...                                                                    */
/*****************************************************************************/
```

Figura 36 :SerialSTP.c

Un ejemplo de ejecución es esta:

```
./serialSTP /dev/ttyUSB0 57600
```

Donde “serialSTP” es el nombre el programa, “/dev/ttyUSB0” es el nombre del puerto que se desea abrir y el último parámetro: “57600” es la velocidad de transmisión que requiere la mota expresada en baudios.

Los baudios es la velocidad de modulación que requiere en una transmisión de información, y depende del componente al que se envía la información. Un grupo de bits enviados en un segundo se denomina baudio. Las diferentes motes tienen definido una velocidad por defecto, tal y como se muestra en la Tabla 5:

Tabla 5: Relación motes-velocidad

Plataforma	Velocidad (baud)
Telos	115200
telosb	115200
Tmote	115200
micaz	57600
mica2	57600
iris	57600
mica2dot	19200
Eyes	115200
intelmote2	115200

Cada vez que se ejecuta el programa en C aparece un mensaje en el que se solicita el número de gateway esclavos que van a existir en la red, el programa se va a quedar esperando hasta que el usuario introduzca un número comprendido entre el 1 y el 255, en el caso de que se recibiese un valor incorrecto se volvería a solicitar el parámetro. Para poder enrutar este dato se tiene que transformar a hexadecimal.

Una vez obtenidos los datos de configuración se tiene que formar al paquete que se va a transferir a través de puerto USB al gateway maestro que está conectado.

Tal y como se comentó en el diseño del protocolo se va a definir en el paquete las cabeceras de TinyOS, para que cuando lo reciba el gateway maestro sea capaz de interpretarlo. Lo más recomendable es definir un buffer en el que ir añadiendo los bytes de las cabeceras y así ir generando el paquete de datos de la Figura 20. De acuerdo a esa especificación, los datos que se almacenan en el buffer son los siguientes:

- El primer byte a asignar es el byte de inicio del paquete o *sync*:
```
WriteBuffer[0] = (unsigned char)0x7E;
```

- El segundo byte a asignar es el byte correspondiente a la parte alta de la variable *Address*:
```
WriteBuffer[1] = (Address &0xFF00) >> 8;
```

- El tercer byte a asignar es el byte correspondiente a la parte baja de la variable *Address*:
```
WriteBuffer[2] = (Address &0x00FF);
```

- El cuarto byte a asignar es el byte de la cabecera *type*:
```
WriteBuffer[3] = 0x42;
```

- El quinto byte a asignar es el byte de la cabecera *group*:
```
WriteBuffer[4] = 0x7d;
```

- El sexto byte a asignar es el byte donde se define la longitud (*length*) del paquete ignorando las cabeceras:
```
WriteBuffer[5] = 0x04;
```

A continuación se muestra cómo se van asignando a las posiciones de el buffer los bytes correspondientes a las cabeceras de TinyOS; del byte 7 al 10 se van a incluir los datos del *payload*, el número de gateways esclavos del sistema y la fecha del sistema que se expresa como hora, minutos y segundos.

- El séptimo byte a asignar ya pertenece al *payload* y se corresponde con la dirección del gateway esclavos de la red, este dato es especialmente delicado porque si el usuario considerase que el número de gateway esclavos que tienen que existir en la red son 126 estaríamos transmitiendo el valor 0x7E que es la marca de fin de mensaje por ese motivo cuando el usuario introduzca el valor 126 tenemos que enviar el valor 0x7D&0x5E.

```
WriteBuffer[6] = (unsigned char)&hexa;

O para el valor 126:
WriteBuffer[6] = (unsigned char)(0x7D&0x5E);
```

- El octavo byte a asignar es el byte de la hora:
```
WriteBuffer[7] = (unsigned char)&hora_hex;
```

- El noveno byte a asignar es el byte de los minutos:
```
WriteBuffer[8] = (unsigned char)&minuto_hex;
```

- El decimo byte a asignar es el byte de los segundos:
```
WriteBuffer[9] = (unsigned char)&segundo_hex;
```

- El undécimo byte a asignar es el byte correspondiente a la parte alta de la variable *CRC*:
  ```
  WriteBuffer[10] = (CRC & 0xFF00) >> 8;
  ```

- El duodécimo a asignar es el byte correspondiente a la parte baja de la variable *CRC*:
  ```
  WriteBuffer[11] = (CRC & 0x00FF);
  ```

- El decimo tercero byte a asignar es el byte de fin del paquete o *sync*:
  ```
  WriteBuffer[12] = 0x7E;
  ```

Cuando ya está cargado todo el mensaje se va a abrir el puerto UBS a través de la orden:

```
serial_fd=serial_open(argv[1],baud);
```

Si ha habido algún problema al abrir el puerto el programa devuelve por pantalla un error, si no, se lanzará de forma automática la orden de escritura en el puerto:

```
serial_send(serial_fd, WriteBuffer, CMD_LEN);
```

Como se ha comentado en apartados anteriores, la estación base va a generar varios mensajes que se enviarán al gateway maestro. Si el usuario del programa serialSTP indica que en la red va a haber 5 nodos con función de gateway esclavo el programa va a generar 5 mensajes, cada uno de ellos llevará en la posición 6 del payload la dirección de un gateway esclavo, sabiendo que estos nodos van a tener siempre las primeras posiciones de la red.

6.2 Comunicación para la sincronización de la red

En este apartado se va a explicar la implementación que se ha realizado para el protocolo de sincronización de la red de sensores inalámbrica.

El programa que se ha implementado para cubrir la funcionalidad descrita anteriormente se ha escrito en el lenguaje nesC sobre el sistema operativo TinyOS y está compuesto de tres componentes:

- libreriaSTP.h: Es una librería definida para poder establecer estructuras y enumerados fuera del programa principal.

- Cliente-ServidorC.nc: Se encarga de la definición de que componentes e interfaces van a ser utilizas en el programa principal y como van a ser llamadas. Su módulo principal es "*configuration*".

- Cliente-ServidorM.nc: Es el programa principal, gestiona las llamadas a las interfaces y componentes. Su módulo principal es "*implementation*".

A continuación se definirán los módulos e interfaces que se han requerido para el desarrollo de esta aplicación y posteriormente se estudiará la funcionalidad para cada uno de los actores de la red de sensores.

6.2.1 Interfaces y componentes de STP

En la Figura 37 se muestra una vista simplificada de una aplicación implementada con TinyOS. Cada nodo representa una componente, y las flechas representan las uniones de las interfaces. Cada flecha figura con el nombre de la interfaz usada.

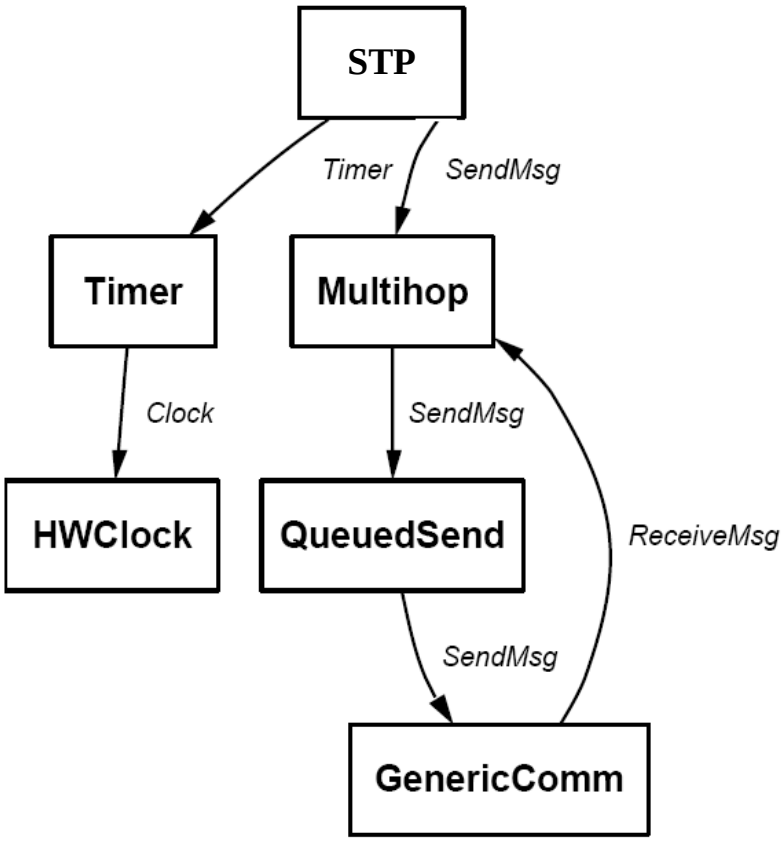

Figura 37: Diagrama de comunicación de los componentes de TinyOS

6.2.2 Gateway Maestro

Para poder marcar un nodo como gateway maestro o esclavo o como nodo (cliente) es necesario generar una estructura de tal manera que cada nodo de la red almacene en su memoria un entero. Si el entero es 0, ese nodo tomará las funciones de gateway maestro, si el entero toma el valor 1, el nodo tomará las funciones de gateway esclavo; si, finalmente, tomase el valor 2, el nodo actuará como un nodo sensor:

```
enum {
        GW_MASTER=0,
        SLAVE=1,
        NODE=2
};
```

Todo protocolo de comunicación tiene como función básica establecer unas reglas con las que se permita la comunicación entre ordenadores. Para poder establecer el protocolo STP se tiene

que definir una trama o una estructura de datos en la que se puedan enmarcar todos los mensajes que se envían en la red.

El protocolo que se está desarrollando es un protocolo de tiempo, por lo que en la estructura se tiene que almacenar la hora. Para poder enrutar los paquetes por la red es necesario que almacenar en el propio paquete la dirección de origen del paquete y la dirección de destino. Por último, para poder identificar el objetivo del paquete y no inundar la red de paquetes innecesarios se ha añadido una variable booleana con la cual se puede saber si el paquete tiene como objetivo requerir algún dato del nodo destino, o por el contrario quiere informar a dicho nodo de un dato. Ese campo es fundamental para evitar cuellos de botellas en la red, además de permitir el ahorro de energía, ya que cada nodo sabrá si tiene que atender la petición o por el contrario puede ignorar el paquete. La estructura que maneja el protocolo STP se define como sigue:

```
typedef struct
{
  uint16_t direccionOrigen;
  uint16_t direccionDestino;
  bool peticion;
  referenceDate hora;
}GlowMsg_t ;
```

La hora es en sí misma otra estructura de datos donde se almacena la hora, minuto y segundo que se hayan transferido.

```
typedef struct
 {
  uint8_t hour;
  uint8_t minute;
  uint8_t second;
}referenceDate;
```

Una vez definido el paquete se va a definir la labor de cada nodo dentro de la red. El gateway maestro siempre va ser el nodo que está conectado a través de un puerto USB, pero esa distinción física no es suficiente para identificarlo en el software del protocolo sino que es necesario identificar ese nodo de forma unívoca. Por ello se definió en los requisitos del sistema que el gateway maestro siempre ha de ocupar la dirección 0 en la red.

Cuando se inicia el gateway maestro, éste va a recibir un paquete por cada uno de los gateways esclavos de la red y la hora del ordenador al que está conectado. Para ello se va a utilizar la interfaz *ReceiveMsg*, de manera que cuando la mote detecte que ha recibido un paquete a través del puerto USB, se señaliza el evento *UARTReceive.receive* que encapsulará el mensaje e irá asociando a las variables los datos enviados desde la estación base, de esta manera:

```
event TOS_MsgPtr  UARTReceive.receive(TOS_MsgPtr msg )
{
        PaquetePC* datoReci=(PaquetePC*)m_msg.data;
         gs_notificado = datoReci-> gs_notificadoPC;
         horaPC.hour=datoReci->hourPC;
           horaPC.minute=datoReci->minutePC;
           horaPC.second=datoReci->secondPC;
         call Timer.start(TIMER_REPEAT, timeRefresh);
        return msg;
}
```

Una vez se hayan almacenado los datos, el gateway maestro va a enviar al gateway esclavo un paquete de tipo respuesta con la hora que ha obtenido de la estación base, iniciando así la sincronización.

Siempre que la dirección de nodo sea la 0x00 y se acabe de capturar un paquete, se va a remitir una respuesta a todos los nodos que estén accesibles la hora que tiene almacenada.

```
  if((datosRec->direccionDestino==0x00) && (datosRec->peticion==TRUE))
  {
    datos->direccionOrigen= TOS_LOCAL_ADDRESS
    datos->direccionDestino= datosRec->direccionOrigen;
    datos->peticion=FALSE;
    datos->hora= horaPC;
  if  (call  SendMsg.send(TOS_BCAST_ADDR,  sizeof(GlowMsg_t),  &m_msg
)==SUCCESS);
 }
```

6.2.3 Gateway Esclavo

El número de gateways esclavos que van a existir en la red es una variable configurable por el usuario del protocolo STP. La obtención de esta variable se va a realizar a través del programa SerialSTP de la manera que se ha explicado en el apartado 6.1.2. Estos gateways van a ser nodos dedicados a enrutar paquetes con el objetivo de sincronizar toda la red.

Los gateways esclavos obtienen siempre la hora del gateway maestro, ya que es el único que está conectado al PC, por ello van a tener que enviar una paquete de tipo petición a las dirección 0x00.

```
if (state==SLAVE)
  {
   datos->direccionOrigen= TOS_LOCAL_ADDRESS;
   datos->direccionDestino=0x00;
   datos->peticion=TRUE;
   datos->hora= horaPC;
    if (call SendMsg.send(0x00, sizeof(GlowMsg_t), &m_msg )==SUCCESS);
 }
```

Cuando los gateways esclavos reciban la contestación del gateway maestro van a tener que almacenar la hora que envía en el paquete y van a construir otro mensaje para reenviarlo a todos los nodos que estén accesibles (envío a broadcast), de la siguiente manera:

```
if  ((datosRec->direccionOrigen==0x00)  &&  (datosRec->direccionDestino
== TOS_LOCAL_ADDRESS))
 {
   state= SLAVE;
   datos->direccionOrigen= TOS_LOCAL_ADDRESS;
   datos->direccionDestino= TOS_BCAST_ADDR;
   datos->peticion=FALSE;
   datos->hora= datosRec->hora;
   horaPC=datosRec->hora;
   call SendMsg.send( TOS_BCAST_ADDR, sizeof(GlowMsg_t), &m_msg);
 }
```

En el caso en el que un gateway esclavo reciba un mensaje de un nodo que no sea el gateway maestro va a generar un paquete de tipo respuesta con la hora que él tiene almacenada y se la va a enviar al nodo que le envió el mensaje.

```
       datos->direccionOrigen= TOS_LOCAL_ADDRESS;
       datos->direccionDestino= datosRec->direccionOrigen
       datos->peticion=FALSE;
       datos->hora= horaPC;
       call SendMsg.send(datosRec->direccionOrigen, sizeof(GlowMsg_t),
&m_msg);
```

6.2.4 Nodos sensores cliente

Siempre que un nodo sensores se active dentro de la red va a emitir un mensaje de tipo petición a toda la red (envío a broadcast), de manera que el mensaje es recibido por un gateway maestro o un gateway esclavo esté va a generar otro mensaje reenviándole la hora.

```
if (state == NODE)
 {
   //Enviamos peticion a BROADCAST
   datos->direccionOrigen= TOS_LOCAL_ADDRESS;
   datos->direccionDestino= TOS_BCAST_ADDR;
   datos->peticion=TRUE;
   datos->hora= horaPC;
   if (call SendMsg.send(TOS_BCAST_ADDR, sizeof(GlowMsg_t), &m_msg
)==SUCCESS);
 }
  return SUCCESS;
}
```

Los nodos sensores cada vez que reciban un mensaje, independientemente de quien se lo mande (gateway maestro o esclavo) van a almacenar la hora del paquete. Además para ayudar a la sincronización los nodos sensores van a reenviar la hora de sincronización a broadcast para intentar sincronizar algún nodo que no sea accesible por un gateway de la red, este mensaje es de tipo respuesta.

```
    if (datosRec->peticion==FALSE ) //si recibo un msg de alguien
distinto al GWM y es una respuesta
    {
      if(state!=SLAVE) //y no soy un GW
        {
          dbg(DBG_USR1,"Receive:Soy un nodo: %d \n",TOS_LOCAL_ADDRESS);
          state=NODE;
          horaPC=datosRec->hora;
          if (datosRec->direccionOrigen!=TOS_LOCAL_ADDRESS)
            {
              call SendMsg.send( TOS_BCAST_ADDR, sizeof(GlowMsg_t),
&m_msg);
              dbg(DBG_USR1,"Receive:Soy un nodo: %d reenvio la hora
\n",TOS_LOCAL_ADDRESS);
            }
```

6.2.5 Mantenimiento de la hora en los nodos

En el protocolo STP se dispone de dos mecanismos que permite la actualización de la hora en los nodos.

Por un lado se puede actualizar la hora de la red de sensores inalámbricos siempre que el usuario lo desee, conectando el gateway maestro a la estación base y ejecutando el programa *serialSTP*. De esta manera se volverá a enviar una nueva hora de sincronización a toda la red. Esta opción podría ser útil si la red se desplaza a una zona con otra franja horario o si se en el país donde está la red se realizan cambios horarios en verano e invierno.

Por otro lado cada nodo tiene definido un mecanismo que le permite ir actualizando la hora que ya tiene almacenada. Para realizar esta labor es necesario definir una variable que indique con qué frecuencia se va a realizar esta actualización. Se tiene que tener en cuenta que cuanto más actualizaciones se hagan más energía se consumirá. Por ese motivo se define el tiempo de reinicio en 10000 milisegundos, que se corresponden a 10 segundos.

Para realizar estas acciones se van a necesitar:

1. Iniciar un disparador que cada vez que llegue a un tiempo definido realice la activación de la hora.
2. Un proceso que calcula y almacena la hora en cada nodo.

De esta manera se obtienen dos métodos de actualización de la hora de la red, una actualización a través de un sistema externo que en este caso es la estación base y por otro lado una actualización interna de cada uno de los nodos que se realiza cada 1000 milisegundos y que permite poder obtener una referencial horaria relativa, pero no exacta. En el supuesto en el que gateway maestro fallara y se desconectara de la estación base, la red podrá ir actualizando la hora local de cada nodo.

A nivel de código la actualización interna de la hora realizada por los nodos es la siguiente:
En el módulo *StdControl.start* se realiza una llamada al *timer* con tiempo de disparo *timeRefresh*. El módulo *Timer.start* funciona como un cronómetro hacia atrás por lo que una vez se ejecuta empezará una cuenta hacia atrás y una vez se finalice la cuenta se realizará la funcionalidad implementada dentro del módulo.

```
command result_t StdControl.start()

       call Timer.start(TIMER_REPEAT, timeRefresh);
```

Posteriormente el evento de ese disparo realiza una llamada a la función *actualizaHora*:

```
event result_t Timer.fired()
{
  GlowMsg_t* datos=(GlowMsg_t*)m_msg.data;

  actualizaHora ();
…
```

La función *actualizaHora* suma a la hora del sistema los 1000 milisegundos que tiene definido el *timeRefresh* y *actualize* la hora que tiene almacenada el nodo.
Para realizar el cálculo de la nueva hora se van a necesitar tres tipos de variables:

- Unas variables en las que el nodo ya tiene una hora almacenada. Se trata de la hora inicial
- Unas variables auxiliares que nos van a servir para hacer el cálculo y en las que se van a almacenar datos como, por ejemplo, el resto de las divisiones enteras.
- Unas variables en la que se va a ir almacenando la nueva hora calculada. Estos valores serán los que se carguen en el nodo, ya que se tratará de la nueva hora del sistema.

La función *actualizaHora* va a realizar el cálculo de la suma de la hora que ya tiene el nodo y los 10 segundos que se han definido entre actualizaciones.
En este cálculo se tiene que tener en cuenta que los minutos y los segundos no pueden ser mayores de 59 y que no hay más de 24 horas en un día.

```
void actualizaHora()
{
    uint8_t hourP;
    uint8_t minuteP;
    uint8_t secondP;
    uint16_t restoSec;
    uint16_t restoMin;
    uint16_t salto;

    hourP=0;
    minuteP=0;
    secondP=0;
    restoSec=0;
    restoMin=0;
    salto=60;

   secondP=horaPC.second + (timeRefresh/1000);
   minuteP=horaPC.minute;
   hourP= horaPC.hour;
   horaPC.second= secondP % salto;
   restoSec= secondP / salto;
   minuteP= horaPC.minute + restoSec;
   horaPC.minute= minuteP % salto;
   restoMin= minuteP / salto;
   hourP= horaPC.hour + restoMin;
   horaPC.hour= hourP % 24;
}
```

Todo procesamiento de datos tarda un tiempo en realizarse y esto puede provocar una desviación entre el tiempo real y el tiempo calculado. El tiempo de procesamiento para las motes MicaZ deben ser similares pero en cualquier caso el analista puede modificar la variable INITIAL_RATE_TIME para minimizar esta desviación.

En las simulaciones de STP se va a realizar esta optimización para conseguir una desviación mínima.

7. Evaluación

En este capítulo se abordará la compilación y las pruebas realizadas al protocolo STP, evaluando el correcto funcionamiento del protocolo.

7.1 Entorno de simulación

Para comprobar el funcionamiento de la aplicación se dispone de una herramienta de simulación denominada TOSSIM, de esta manera se podrá observar el comportamiento de la aplicación sobre cada mote. También es necesario saber si los paquetes que se envían llegan correctamente a la estación base y si los datos son consistentes. Para ello nos serviremos de la herramienta TinyViz, interfaz gráfica del simulador TOSSIM, además del depurador DBG.

Un paso previo a la simulación es la compilación del código, y a continuación vamos a explicar muy brevemente en qué consiste este paso intermedio y cómo se ha realizado en este proyecto.

Para la compilación del protocolo se va a utilizar la compilación con *make*. Para ello hay que situarse en el directorio donde se encuentran los archivos de la aplicación y escribimos la siguiente instrucción:

```
$make pc
```

La Figura 38 muestra el proceso de compilación y si éste ha sido satisfactorio o no.

```
maria@maria-desktop:~/tinyos-1.x/apps/STP$ make pc
mkdir -p build/pc
    compiling Cliente_ServidorC to a pc binary
ncc -o build/pc/main.exe -g -O0 -pthread -fnesc-nido-tosnodes=1000 -fnesc-simula
te -Wall -Wshadow -DDEF_TOS_AM_GROUP=0x7d -Wnesc-all -target=pc -fnesc-cfile=bui
ld/pc/app.c -board=micasb -DIDENT_PROGRAM_NAME=\"Cliente_Servido\" -DIDENT_USER_
ID=\"maria\" -DIDENT_HOSTNAME=\"maria-desktop\" -DIDENT_USER_HASH=0xfd2c668aL -D
IDENT_UNIX_TIME=0x4b046bccL -DIDENT_UID_HASH=0xb26a5c60L Cliente_ServidorC.nc -l
m
/opt/tinyos-1.x/tos/platform/pc/external_comm.c: In function ‘acceptConnection’:
/opt/tinyos-1.x/tos/platform/pc/external_comm.c:158: warning: pointer targets in
 passing argument 3 of ‘accept’ differ in signedness
/opt/tinyos-1.x/tos/platform/pc/PowerStateM.nc: In function ‘__nesc_nido_initial
ise’:
/opt/tinyos-1.x/tos/platform/pc/PowerStateM.nc:778: warning: passing argument 1
of ‘memset’ discards qualifiers from pointer target type
    compiled Cliente_ServidorC to build/pc/main.exe
```

Figura 38: Compilación

En este caso, se puede observar que la aplicación ha sido compilada correctamente ya que aparece la frase:”`compiled Cliente_ServidorC to build/pc/main.exe`”.

Una vez verificado el proceso de compilación, el usuario se tiene que situar en el directorio que ha creado el compilador con el fichero ejecutable:

```
$cd build/pc
```

En esta ruta se encuentra el fichero *main.*exe que es el ejecutable de nuestra aplicación y ya se puede empezar a simular.

7.2 Descripción del entorno de simulación

Una vez compilada la aplicación y obtenido el fichero ejecutable, tan sólo queda depurarla y simularla.

El lenguaje NesC permite agregar órdenes en el código de la aplicación a través del depurador DBG con el cual se pueden obtener diferentes trazas de la aplicación. A continuación se muestra un ejemplo de estas instrucciones:

```
$dbg (dbg(DBG_<variable reservada>, "<Texto>");)
```

En las simulaciones de STP se van a usar estas instrucciones para poder seguir el comportamiento de cada nodo de la red.

Para poder volcar por pantalla los mensajes de la instrucción *dbg* se debe exportar el tipo que se emplea en el *dbg*.

```
dbg(DBG_<tipo>, "”);
```

Donde tipo puede ser cualquiera de los modos debug por defecto, que son: `all, boot, clock, task, sched, sensor, led, route, am, crc, packet, encode, radio, logger, adc, i2c, uart, prog, sim, queue, simradio, hardware, simmem, usr1, usr2, usr3, temp, error, none`.

En esta simulación se va a usar el tipo *usr1* ya que es un tipo reservado para los usuarios. De tal manera que al inicio de la simulación ejecutamos la línea:

```
export DBG=usr1
```

Una vez se ha validado el código se puede simular ejecutándolo de forma paralela a TinyViz. Para ello es necesario tener dos terminales abiertos, uno de ellos ejecuta la aplicación utilizando la opción `–gui`, con lo que la aplicación se inicia y queda a la espera de arrancar la interfaz gráfica TinyViz. En el segundo terminal se arrancará TinyViz con lo que se deberá posicionar en el directorio `/opt/tinyos-1.x/tools/java/net/tinyos/sim`, que es donde se encuentra el fichero ejecutable de la herramienta y se invocará a TinyViz.

En la Figura 39 se muestra la aplicación ejecutándole con la opción `–gui`, lo que nos permite arrancar el interfaz gráfico.

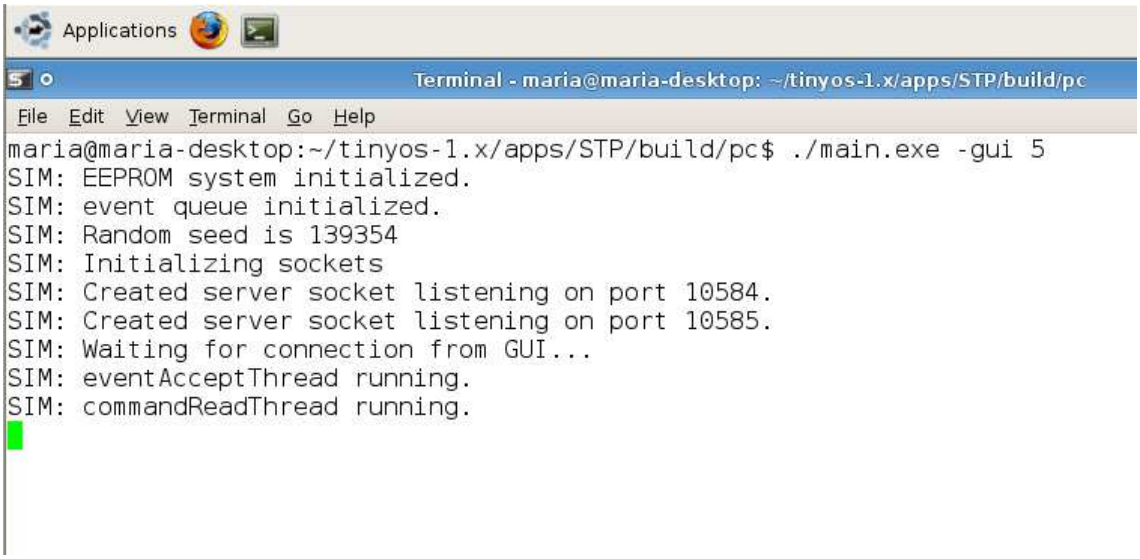

Figura 39: Ejecución de la aplicación con la opción -gui

En la Figura 40 se observa el otro terminal con el que se ha arrancado TinyViz.

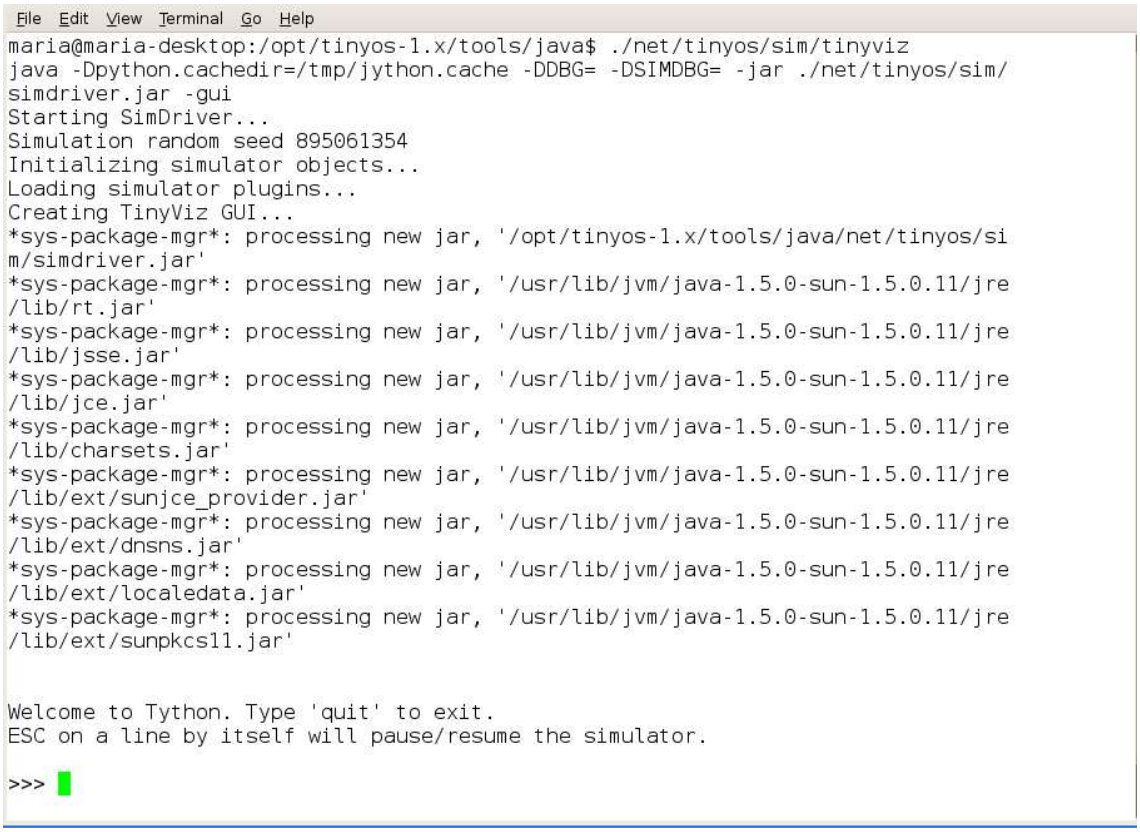

Figura 40: Arranque de TinyViz

Tras realizar estas acciones se abrirá el interfaz gráfico de TinyViz donde se tendrán que activar las opciones que se necesiten para la simulación.

A continuación, activamos los plug-ins que consideremos oportunos para depurar la aplicación. En nuestra opinión, los más interesantes o importantes son *Debug Messages*, *Sent radio packets*,

Radio links y Radio model para ello se han de seleccionar estos literales en el desplegable que aparece al pinchar en la pestaña *plugins* de la parte superior izquierda de la pantalla.

A continuación se puede iniciar la simulación pulsando el botón de inicio tal y como se hace en la Figura 41; donde se observa cómo se empiezan a emitir mensajes entre los nodos.

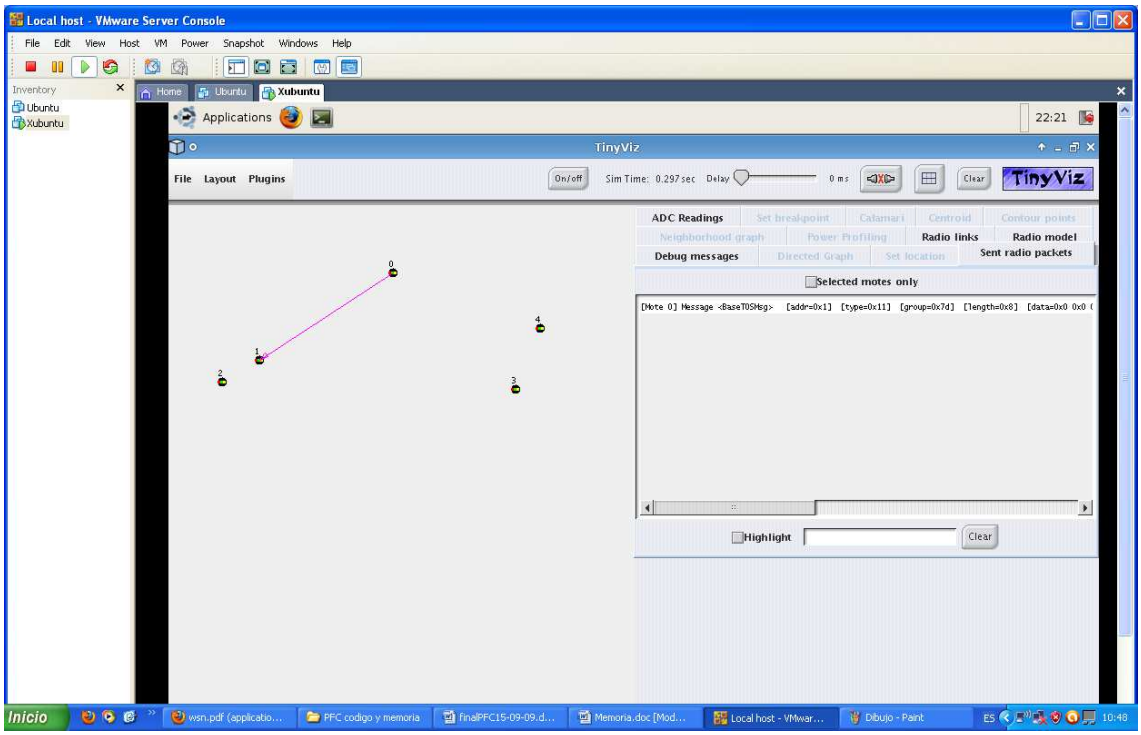

Figura 41: Simulación TinyViz

En la Figura 42 se puede observar como las líneas de color rosa son los mensajes que está enviando el GM (situado en la mitad de la imagen) a los tres nodos esclavos de la red. Por otro lado con un mensaje de color amarillo el gateway esclavo con dirección 3 encamina un mensaje con la hora al nodo sensor con dirección 4.

El nodo 4 cuando ha iniciado su funcionamiento ha detectado que se ha superado el tiempo establecido para espera un mensaje y ha solicitado la hora de sincronización a los nodos cercanos.

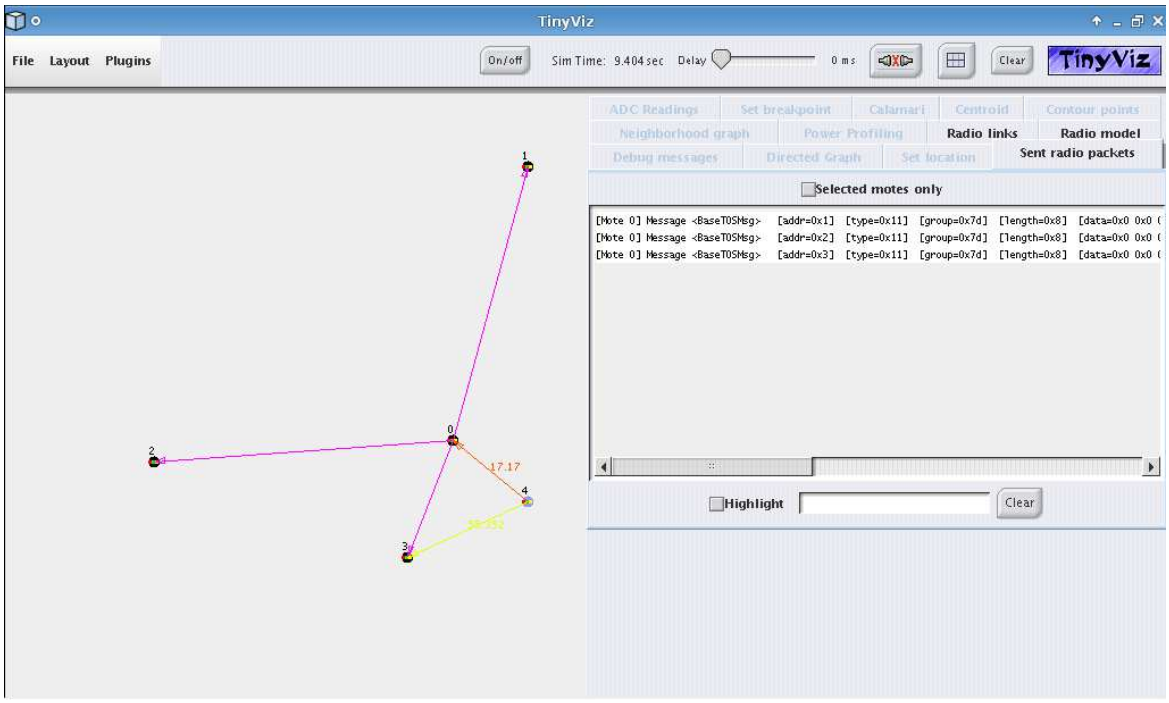

Figura 42: Simulación completa

En la simulación de la Figura 43 se ve otro tipo de ejecución donde el nodo sensor 4 recibe el mensaje de sincronización de los 3 gateways esclavos. Además al activar el plug-in *Debug message* podemos ver en Tinyviz los mensajes del protocolo además de la trama del mensaje que se envía.

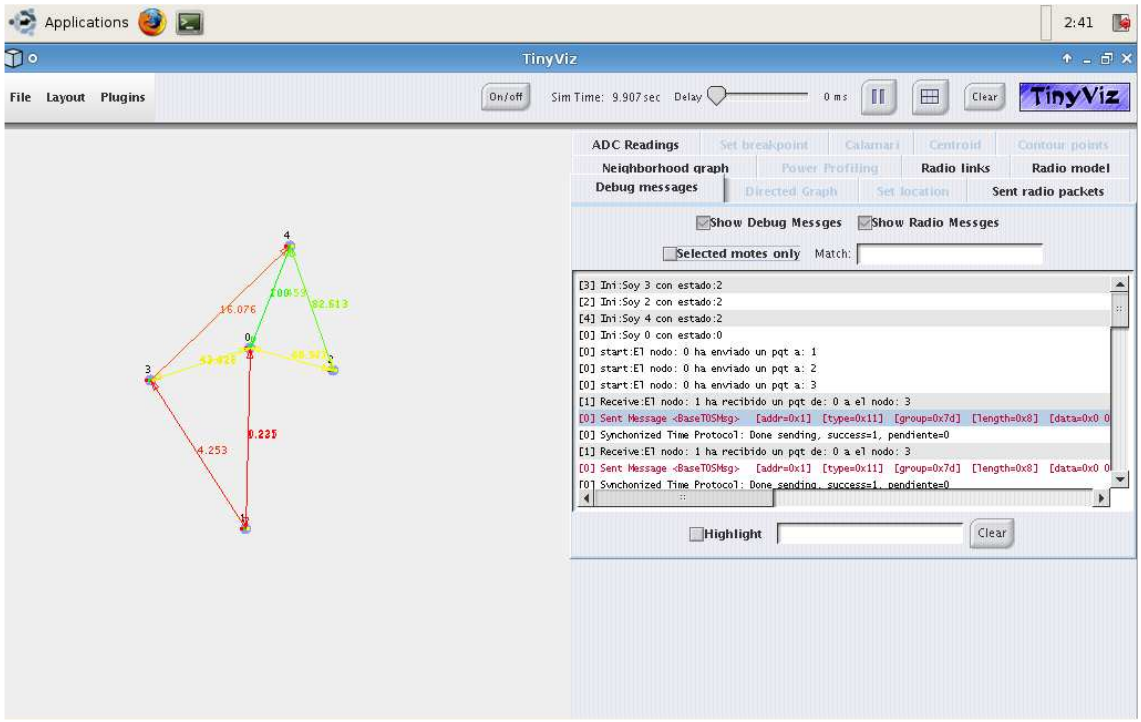

Figura 43: Simulación en modo debug

7.2 Simulación del mantenimiento del tiempo

En este apartado se va a realizar una simulación y su correspondiente análisis de los tiempos de sincronización de la red, además de comprobar la desviación del tiempo que almacena el nodo frente al tiempo real.

La aplicación Tinyviz tiene un delay que es muy útil para la simulación de aplicaciones con un número reducido de motes, ajustando este delay entre los 500 y los 0 ms es posible realizar la simulación en tiempo real o acelerarlo.

Por otro lado ha sido necesario suprimir gran volumen de los mensajes debug de STP ya que para simulaciones de larga duración se llena el buffer del plug-in *DebugMessa*ge de Tinyviz y el simulador se paraba.

Para el análisis del manteniendo del tiempo de STP se han tenido como referencia la hora real y la hora de los nodos transcurridos 1 minuto, 10 minutos, 60 minutos y un día de simulación. Además se tiene que tener en cuenta que el periodo de actualización del tiempo que tiene el nodo está configurado por defecto a 10.000 milisegundos, aunque este valor podría ser modificado en caso de necesitarlo a través del fichero *libreriaSTP.h* en el parámetro *INITIAL_TIMER_RATE.*

Como se comentó en el capitulo anterior el evento de la función *timer* se lanza con lo configurado en la variable t*imeRefresh.* El valor que toma la variable *timeRefresh* dentro del programa es la que se defina en la librería *libreriaSTP.h* en el parámetro *INITIAL_TIMER_RATE.*

```
command result_t StdControl.start()
       call Timer.start(TIMER_REPEAT, timeRefresh);
```

Para realizar el seguimiento dejamos activo el mensaje de *debug* de componente *timer*, de esta manera se puede comparar la hora que ha actualizado el nodo y la hora de simulación. En la Figura 44 se muestra una simulación de 1 hora.

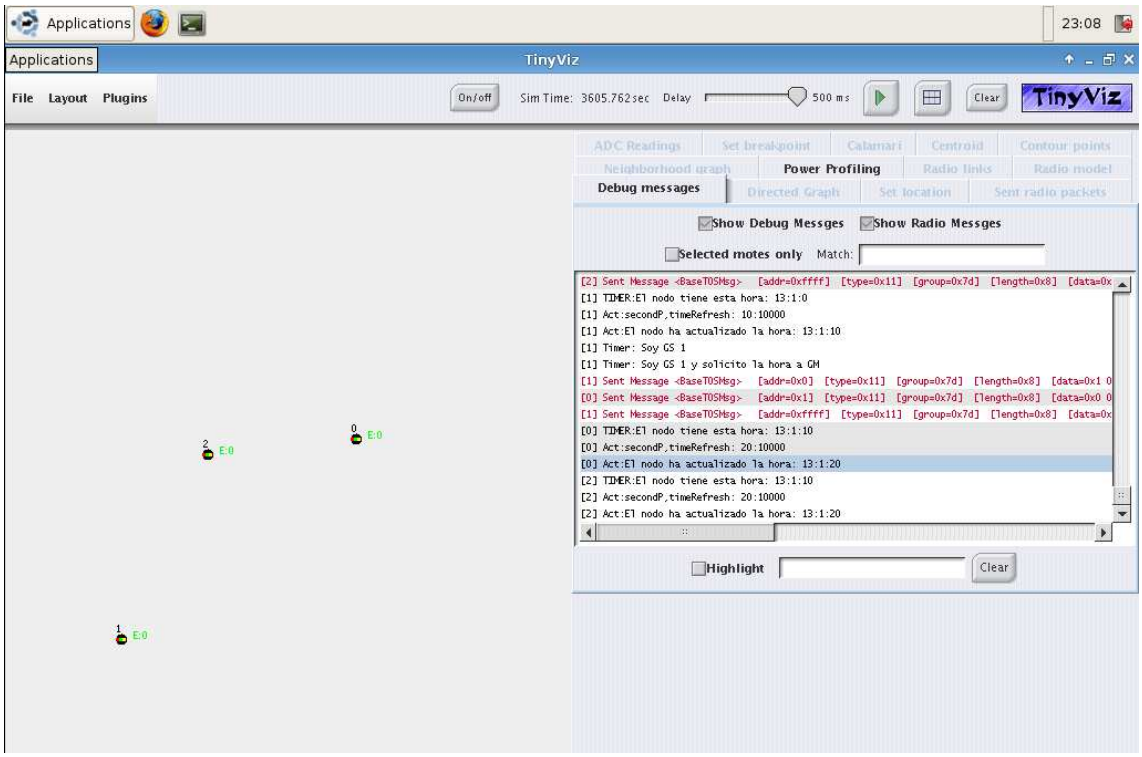

Figura 44: Simulación de una hora

En la Tabla 6 se muestra la relación entre el tiempo real en el momento de la simulación y el tiempo de sincronización que tenía los 5 nodos de la red. Comparando ambos valores se calcula la desviación del tiempo en el nodo.

Esta desviación se calcula de la siguiente manera:

```
Desviación= |tiempo_real – tiempo_del_nodo|
```

La desviación está en valor absoluto ya que se considera que aunque el reloj se adelante o atrase el resultado sigue estando desviado.

Tabla 6: Datos de simulación de tiempos

Hora inicio (HH:MM:SS)	Duración de simulación (segundos)	Nodo	Tiempo del nodo (segundos)	Desviación del tiempo de simulación(segundos)
12:00:00	64	Nodo 0	50	14
13:00:00	120	Nodo 4	120	0
12:00:00	601	Nodo 3	600	1
12:00:00	3605	Nodo 2	3680	75
12:00:00	86403	Nodo 1	88460	2057

En la Figura 45 se muestran los segundos de diferencia que existen entre el nodo y el tiempo real teniendo configurado el parámetro *INITAL_TIME_RATE* a 10.000 milisegundos.

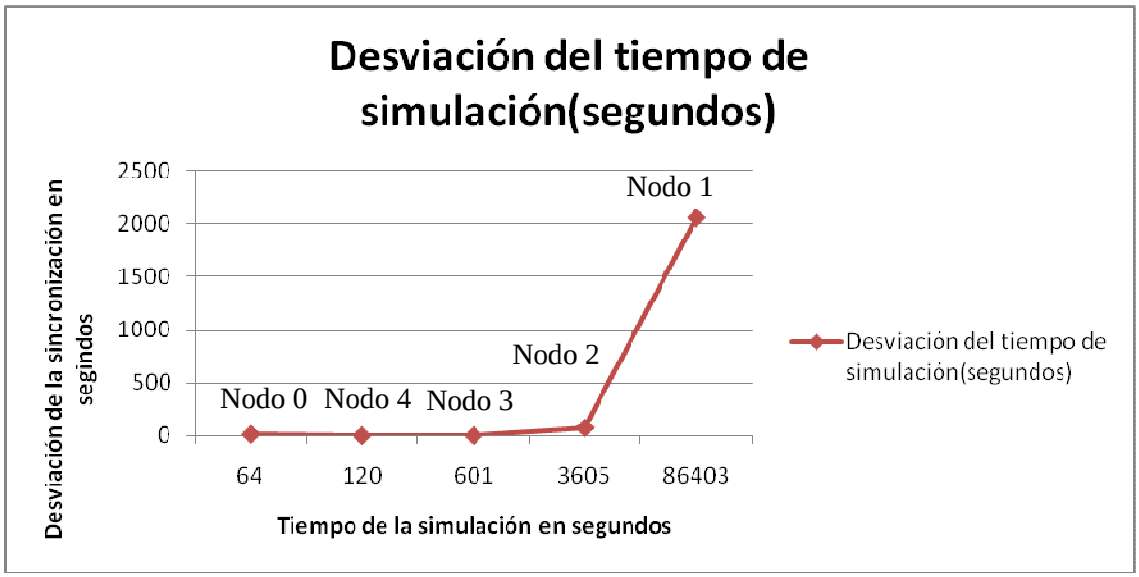

Figura 45: Desviación del tiempo en la simulación

En el gráfico se ve que el tiempo se va degradando progresivamente, esto es debido al tiempo de procesamiento que requiere el cálculo de la hora en STP. Por ello hay que ajustar periódicamente el tiempo para que las desviaciones sean mínimas. En particular, de esta evaluación obtenemos que a partir de 3600 segundos (10 minutos de simulación) es conveniente volver a sincronizar el nodo.

En la Tabla 7 se muestra un conjunto de simulación en las que se ha ido ajustando el parámetro *INITAL_TIME_RATE* para conseguir la mínima desviación posible.

Tabla 7: Optimización del parámetro INITIAL_TIME_RATE

Nodo	INITAL_TIME_RATE (milisegundos)	Tiempo de simulación (segundos)	Tiempo del nodo (segundos)	Desviación del tiempo de simulación(segundos)
Nodo 0	10.000	14.402	14.730	328
Nodo 0	10.010	14.425	14.760	335
Nodo 0	10.100	14.425	14.620	195
Nodo 0	10.200	14.421	14.470	49
Nodo 0	10.250	14.423	14.400	23
Nodo 0	10.245	14.412	14.400	12
Nodo 0	10.240	14.419	14420	1

En esta tabla se puede ver que la configuración más óptima es *INITAL_TIME_RATE* a 10.240. Esto implica que en el cálculo de la hora y en la actualización de la misma que realiza el nodo se tardan 240 milisegundos.

En la Figura 46 se puede observar como se ha ido corrigiendo la desviación en segundos según se ha ido calibrando el parámetro *INITIAL_TIME_RATE*.

Tras la optimización del parámetro INITIAL_TIME_RATE se ha conseguido que la desviación en la sincronización sea mínima lo que permite que la red se sincronice con precisión.
Además en todas las simulaciones realizadas se ha comprobado que los nodos se sincronizan a la par y no se ha obtenido en ningún caso que dos nodos actualizaran horas distintas, y esto da solidez a la sincronización de la red.

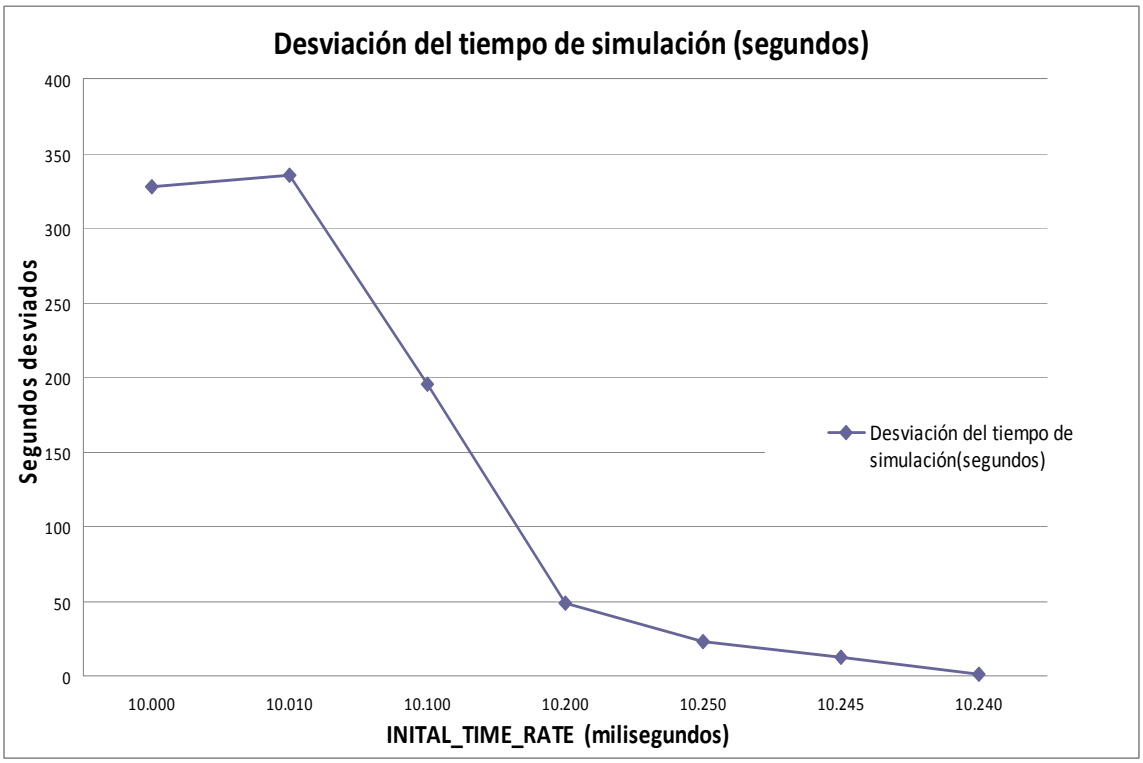

Figura 46: Desviación del tiempo en la optimización de INITIAL_TIME_RATE

En la Tabla 8 se pueden observar los tiempos de sincronización de una red de 8 nodos con el parámetro INITIAL_TIME_RATE optimizado.

Tabla 8: Simulación de tiempo de los nodos de una red

Nodo / Rol	INITAL_TIME_RATE (milisegundos)	Tiempo de simulación (segundos)	Tiempo del nodo (segundos)	Desviación del tiempo de simulación (segundos)
Nodo 0/Gateway maestro	10.240	3.600	3.600	0
Nodo 1/Gateway esclavo	10.240	3.600	3.590	10
Nodo 2 Gateway esclavo	10.240	3.600	3.590	10
Nodo 3/Gateway esclavo	10.240	3.600	3.600	0
Nodo 4/Nodo sensor	10.240	3.600	3.590	10
Nodo 5/Nodo sensor	10.240	3.600	3.600	0
Nodo 6/Nodo sensor	10.240	3.600	3.600	0
Nodo 7 Nodo sensor	10.240	3.600	3.600	0

Como se puede observar en la Figura 47 la desviación de todos los nodos es similar y varía en un valor máximo de 10 segundos que es el periodo con el que se actualiza internamente la hora en los nodos.

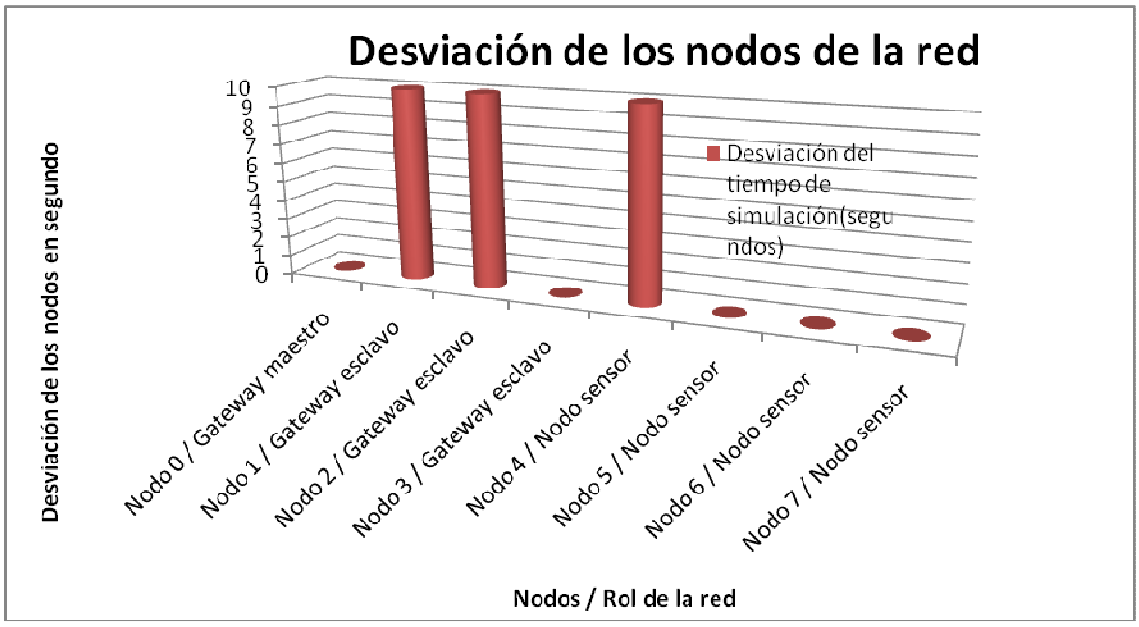

Figura 47: Desviación del tiempo de los nodos de una red

En esta gráfico se puede observar que la hora de los nodos no difiere más que el periodo de actualización de la hora y esto implica que la efectividad de STP no depende del rol del nodo.

A continuación en la Tabla 9 se muestran los resultados de dos simulaciones de larga duración, una simulación de 4 horas y otra de un día completo, esto nos va a permitir ver si la desviación sigue manteniéndose en un margen asumible transcurrido un día.

Tabla 9: Simulaciones con los parámetros optimizados

Nodo / Rol	INITAL_TIME_RATE (milisegundos)	Tiempo de simulación (segundos)	Tiempo del nodo (segundos)	Desviación del tiempo de simulación(segundos)
Nodo 0 / Gateway maestro	10.240	14.419	14420	1
Nodo 0 / Gateway maestro	10.240	86.414	86420	6

En la simulación que se ha realizado con duración de un día sólo se ha desviado la hora almacenada el nodo en 6 segundos.

En la Figura 48 se puede apreciar que la desviación del tiempo de simulación inferior a un 1% tanto para simulaciones de 4 horas como para simulaciones de 1 día.

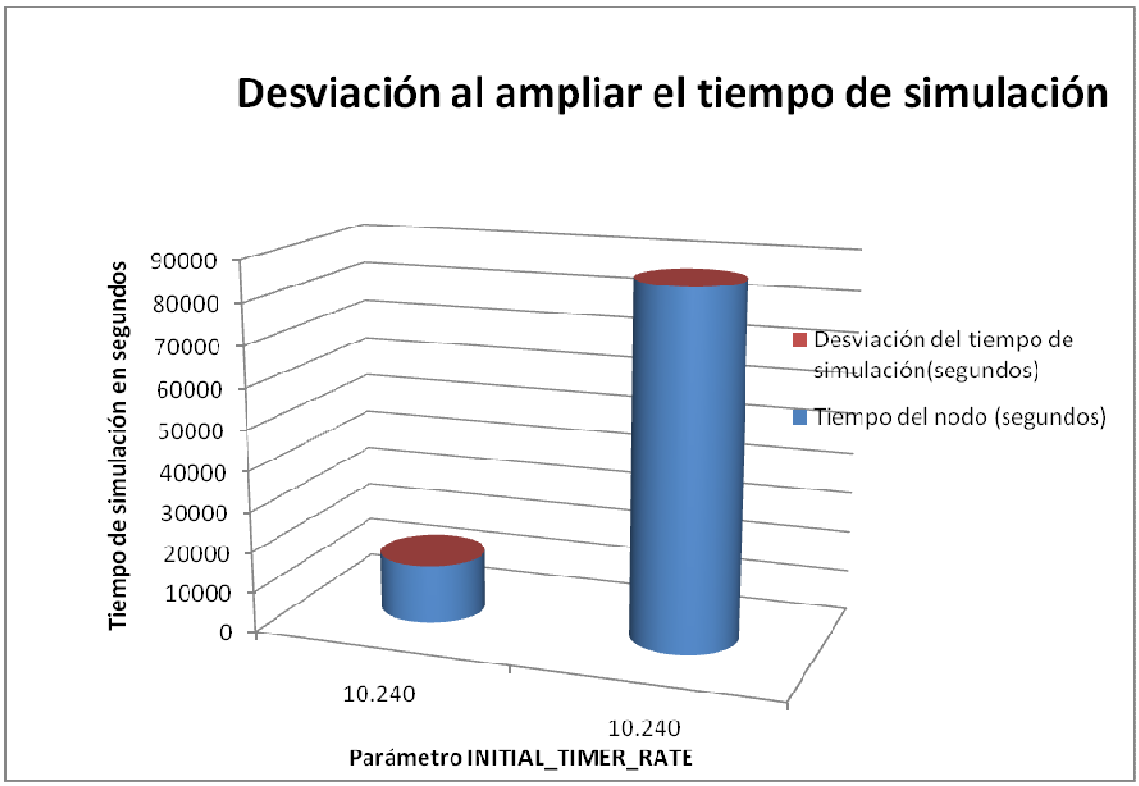

Figura 48: Desviación en dos simulaciones para el mismo INITIAL_TIMER_RATE

Esto nos indica que tras la optimización del parámetro INITIAL_TIMER_RATE la red permanecerá sincronizada y con una desviación mínima y evitará que el usuario tenga que estar re-sincronizándola en cortos periodos de tiempo y con ello ahorramos energía en nodos.

7.3 Análisis de consumos

En este apartado se va a estudiar el consumo de energía que conlleva la sincronización de una red de sensores usando el protocolo STP.

Para llevar a cabo este análisis se va a necesitar el simulador Tinyviz. Esta herramienta ya se ha usado en apartados anteriores pero para analizar el consumo se va a necesitar activar un plug-in nuevo, el plug-in *Power profiling.*

Cuando se realiza la simulación con TOSSIM se tiene que exportar el parámetro *power*, de esta manera cuando se ejecute en paralelo con Tinyviz se va a poder visualizar en esta herramienta los consumos de los nodos. Para ello se debe lanzar el comando:

```
export DBG=usr1,power
```

Una vez se han exportado estos parámetros ya se puede lanzar la simulación con la opción *–gui* para poder analizarla con Tinyviz y con la opción *–p* para que lo consumos también sean visibles:

```
./build/pc/main.exe –gui –p 5
```

Una mote MicaZ parte de un voltaje inicial de unos 3 voltios, como resultado de usar dos pilas tipo AA de 1,5V cada una.

En la Figura 49 se aprecia como Tinyviz muestra los consumos de los nodos en una simulación. Se puede observar el trabajo realizado (miliJulios) de los componentes de la radio, la CPU, los LED's y la memoria EEPROM.

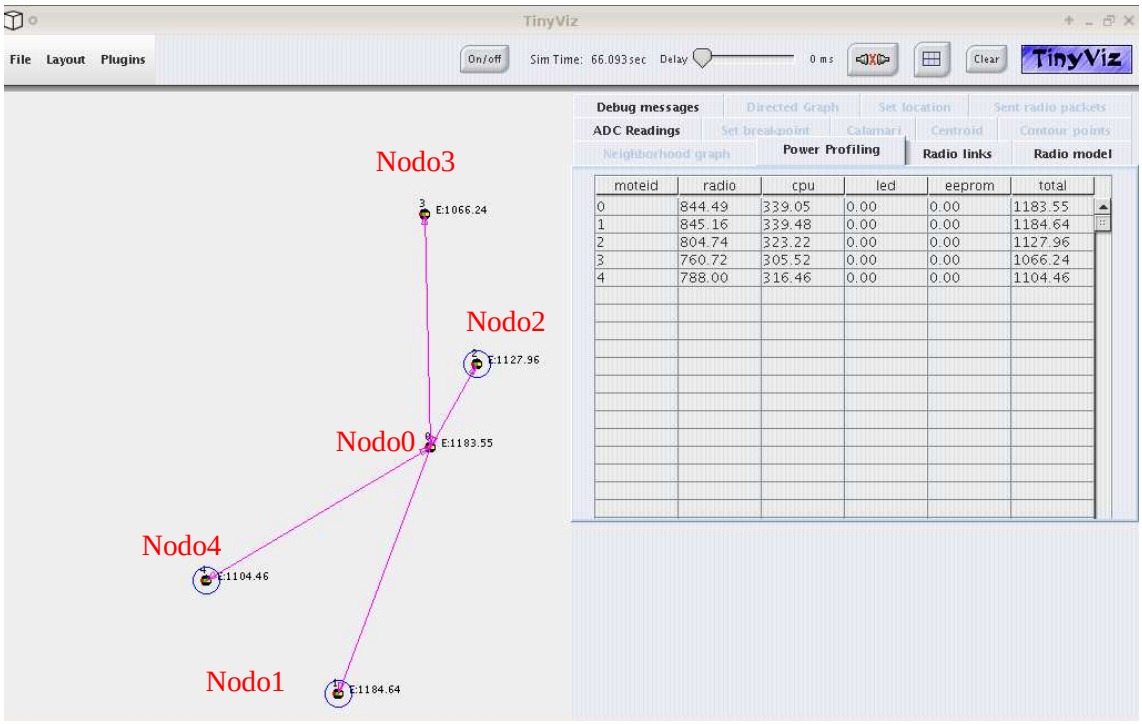

Figura 49: Simulación con el plug-in Power Profiling

Para empezar el análisis del consumo se va a valorar como varía el consumo del gateway maestro según va aumentando el tiempo de simulación.

Este nodo obtiene el tiempo de simulación de la estación base y debe enviar esa información a los gateways esclavos además de atender a las solicitudes del tiempo de sincronización de otros nodos.

En la Tabla 10 se ve como al aumentar el tiempo de simulación de la columna 1 va aumentando el consumo total de energía del nodo.

Tabla 10: Consumo del gateway maestro en distintas simulaciones

Duración de simulación (segundos)	Nodo / Rol	Consumo total (miliJulios)
64	Nodo 0 / Gateway maestro	1020,68
120	Nodo 0 / Gateway maestro	1991,57
338	Nodo 0 / Gateway maestro	5999,63
541	Nodo 0 / Gateway maestro	9682,62

En la Figura 50 se aprecia como el consumo se eleva considerablemente según avanza el tiempo de simulación ya que el nodo necesita usar con más frecuencia la radio.

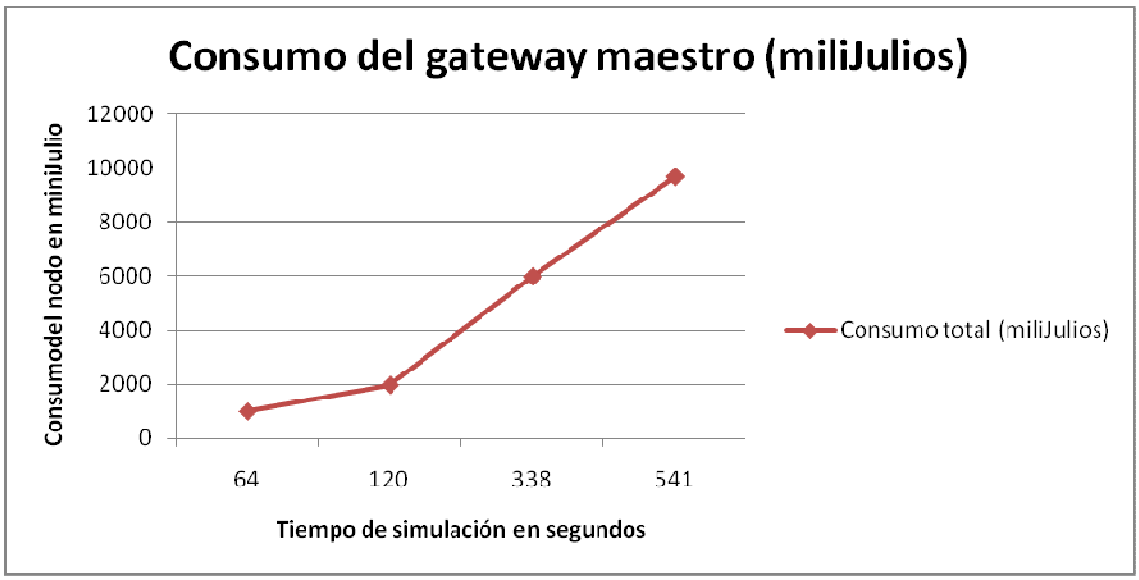

Figura 50: Consumo del gateway maestro

En la Tabla 11 se realiza la comparativa entre el consumo de la radio y el consumo total del gateway maestro para poder apreciar la proporción de energía que requiere el uso de la radio frente al consumo total.

Tabla 11: Consumo de la radio frente al consumo total para el gateway maestro

Duración de simulación (segundos)	Nodo / Rol	Consumo radio (miliJulios)	Consumo total (miliJulios)
64	Nodo 0 / Gateway maestro	728,32	1020,68
120	Nodo 0 / Gateway maestro	1420,98	1991,57
338	Nodo 0 / Gateway maestro	4280,57	5999,63
541	Nodo 0 / Gateway maestro	6908,34	9682,62

En la Figura 51 se aprecia que prácticamente la mitad del consumo que tiene el nodo es por el uso de la radio.

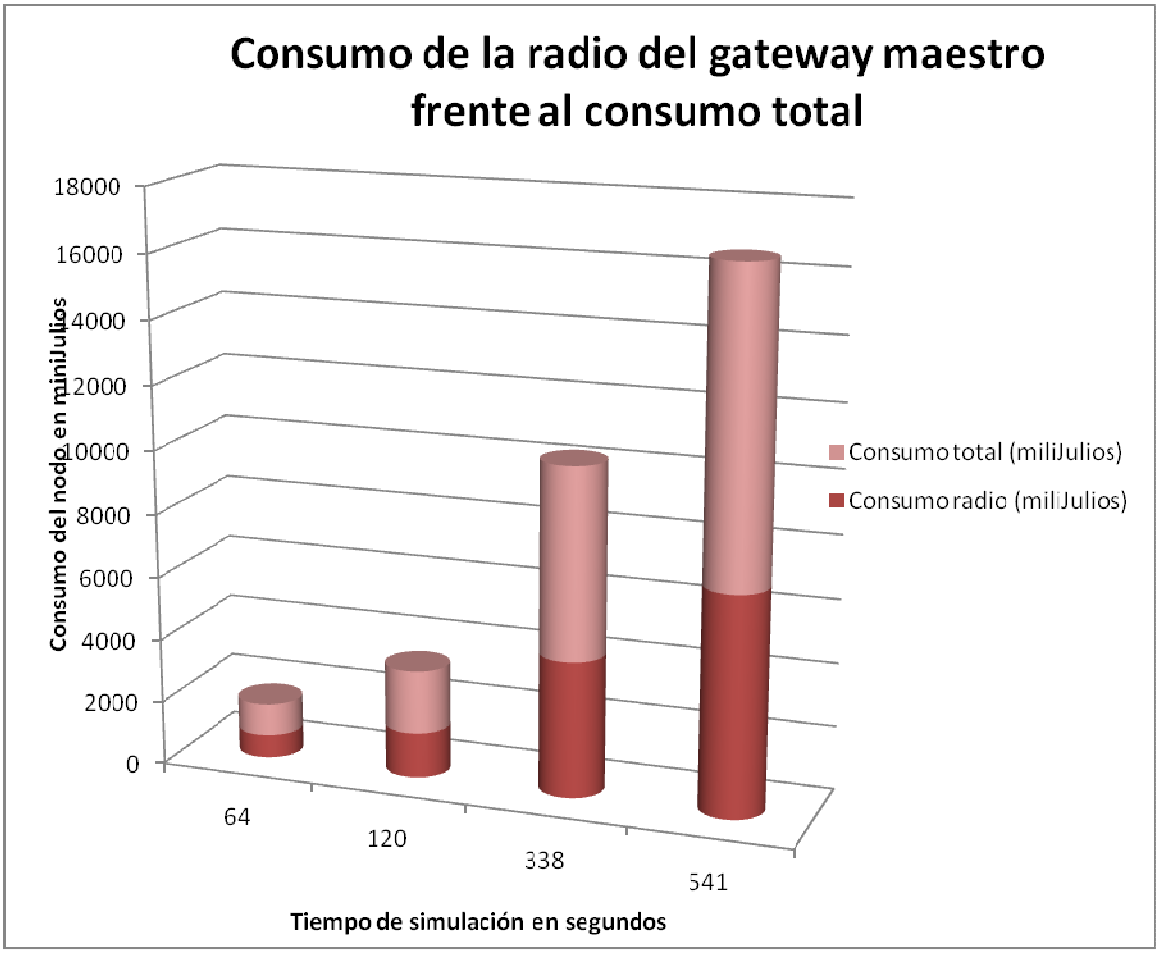

Figura 51: Comparación de los consumos de forma gráfica

Además se va a analizar si varía mucho el consumo del gateway maestro según el número de nodos que hay en la red.

En la Tabla 12 se puede observar que si el gateway maestro es el único de la red y por lo tanto el consumo de la radio es mínimo el consumo de energía es muy pequeño. En cambio si hay nodos en la red el consumo se eleva pero se mantiene estable, por lo que se puede afirmar que el tiempo de vida del nodo no se ve comprometido por incluir algunos nodos más en la red.

Tabla 12: Consumo del GM según los nodos de la red

Duración de simulación (segundos)	Nodos en la red	Nodo / Rol	Consumo total (miliJulios)
64	1	Nodo 0 / Gateway maestro	2,19
64	2	Nodo 0 / Gateway maestro	1020,68
64	4	Nodo 0 / Gateway maestro	1012,4

En la Figura 52 se puede observar esto de forma gráfica. Manteniéndose el consumo por debajo de 1000 miliJulios con redes de 2 y 4 nodos.

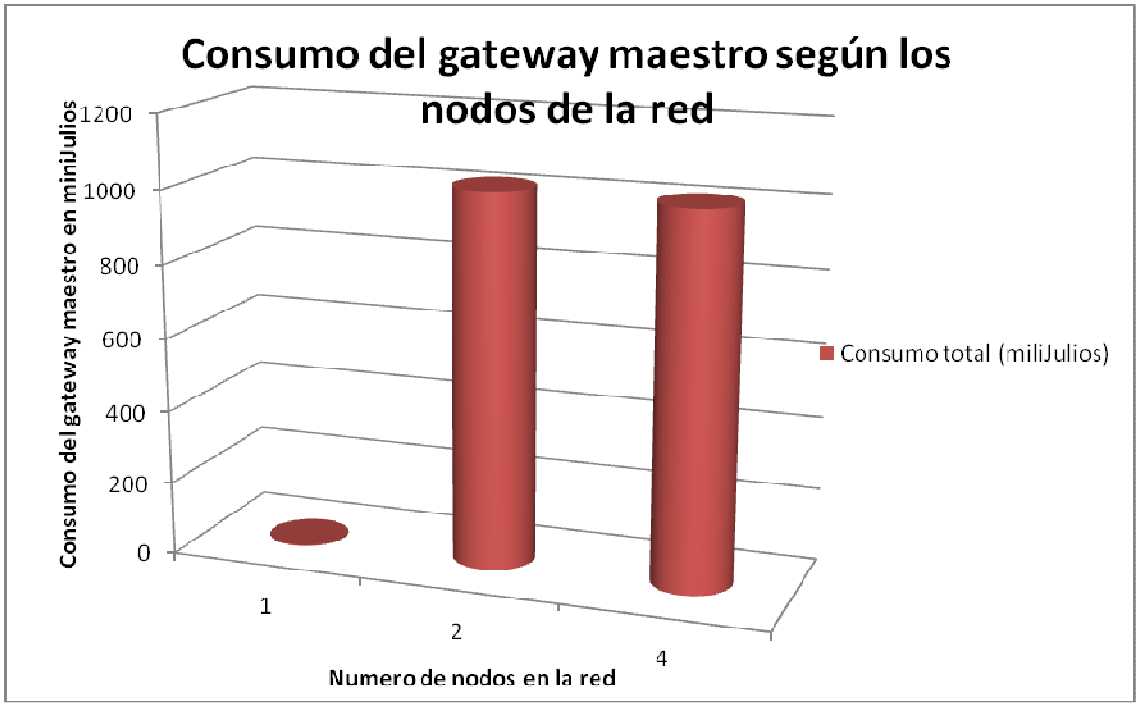

Figura 52: Consumo de GM según los nodos que hay en la red

A continuación se va analizar el consumo de distintos nodos en la red para una misma simulación.

En la Figura 53 se puede apreciar un pantallazo de Tinyviz donde se ve la distribución de los nodos en la red y los consumos realizados por cada uno de ellos en la simulación de 66 segundos.

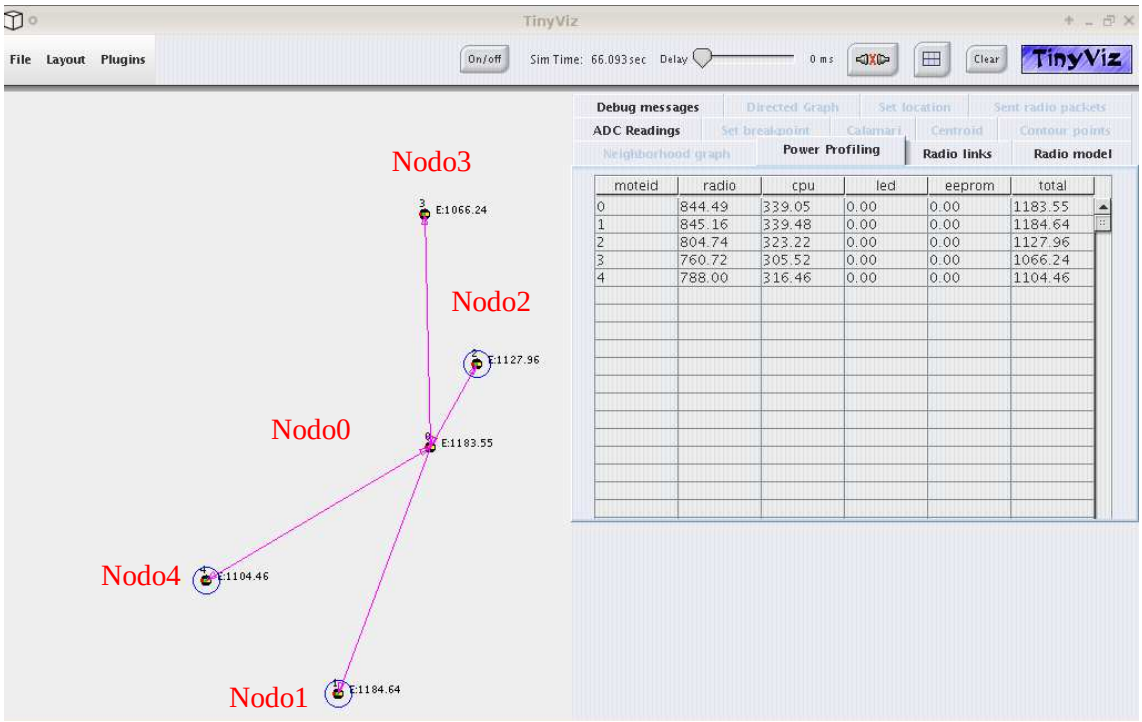

Figura 53: Simulación de sincronización para una red de 5 nodos

En la Tabla 13 se observa en la segunda columna que rol desempeña cada uno de los nodos de la red y los consumos de cada uno de ellos

Tabla 13: Consumos según el rol del nodo

Duración de simulación (segundos)	Nodo / Rol	Consumo radio (miliJulios)	Consumo total (miliJulios)
66	Nodo 0 / Gateway maestro	844,49	1183,55
66	Nodo 1/ Gateway esclavo	845,16	1184,64
66	Nodo 2/ Gateway esclavo	804,74	1127,96
66	Nodo 3/ Gateway esclavo	760,72	1066,24
66	Nodo 4 / Nodo sensor	788	1104,46

En la Figura 54 se observa de forma gráfica que todos los nodos tienen un consumo simular durante la simulación y además ningún nodo tiene un consumo de energía de su radio mucho mayor que otro, lo que implica que la sincronización de la red es una labor desempeñada por todos los nodos de la red y que los nodos no tienen un mayor consumo por tener un rol distinto en la red o estar más o menos cerca del gateway maestro.

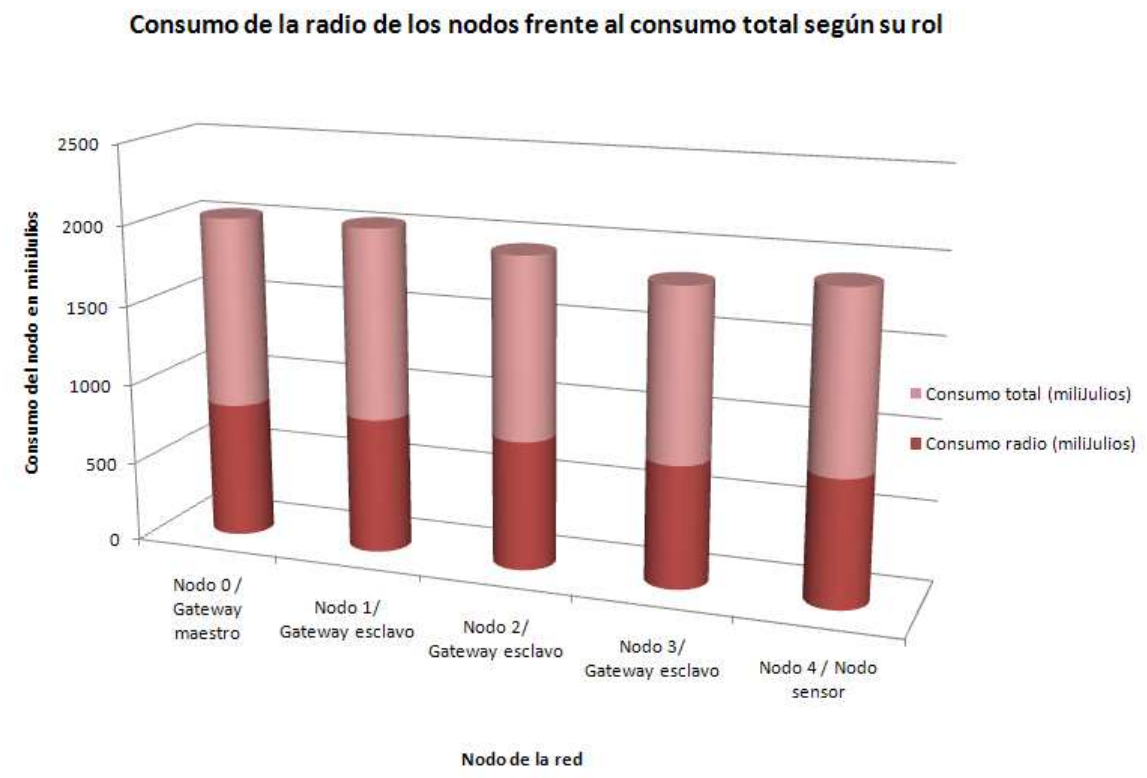

Figura 54: Consumo de la radio frente al consumo total según su rol

Por último se va a analizar el consumo con periodos de simulación de 1, 5 y 10 minutos de simulación para tres nodos distintos.

En la Tabla 14 se muestra el consumo total del nodo 0, 2 y 4 en la simulación de la duración indicada en la segunda columna.

Tabla 14: Tabla de consumos por rol y tiempo

Tiempo de referencia (minutos)	Duración de simulación (segundos)	Nodo / Rol	Consumo total (miliJulios)
1	64	Nodo 0 / Gateway maestro	1020,68
5	338	Nodo 2 / Gateway esclavo	6074,6
10	541	Nodo 4 / Nodo sensor	9630,51

En esta tabla se observa que para distintos roles el consumo va elevándose según avanza el tiempo de simulación.
Como hemos visto antes el consumo de los nodos es bastante estable con independencia del rol desempeñado por lo que los tiempos de la columna 4 van a ser muy simulares para el resto de nodos de la red.

En la Figura 55 se muestra una gráfica donde se puede apreciar como el consumo crece de forma casi lineal durante la ejecución de STP:

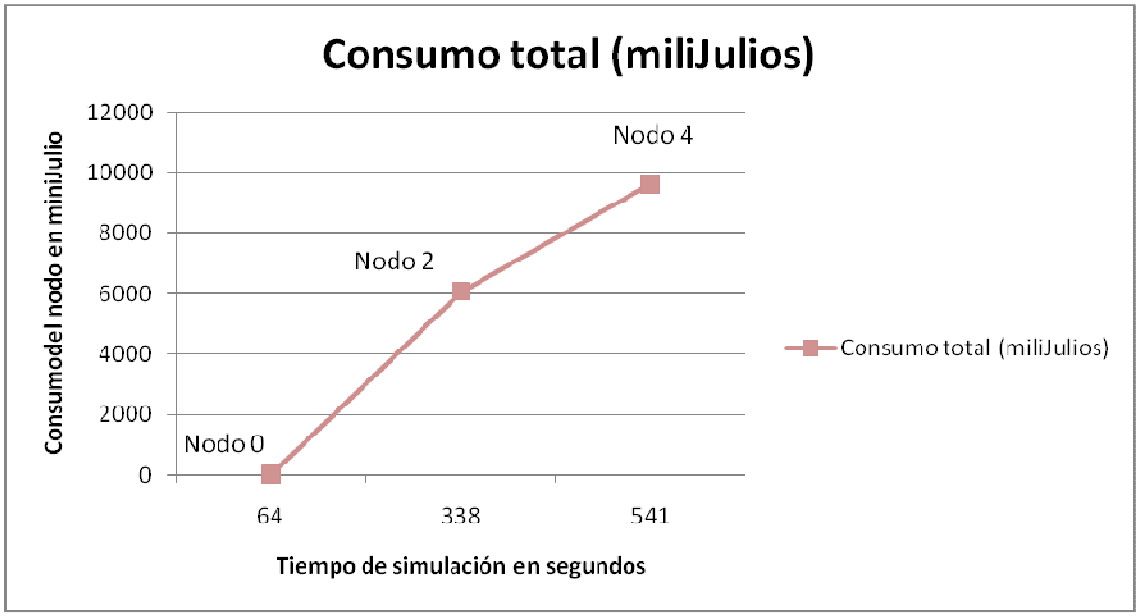

Figura 55: Consumo total por nodo y tiempo

En la Tabla 15 se hace una comparativa entre el consumo de la radio y el consumo total para distintos nodos, de la cual se puede concluir que unas dos quintas partes de la energía que consume el nodo se utiliza en el envío de mensajes a través de su radio.

Tabla 15: Comparativa de consumos

Tiempo de referencia (minutos)	Duración de simulación (segundos)	Nodo	Consumo radio (miliJulios)	Consumo total (miliJulios)
1	64	Nodo 0	1,61	2,19
5	338	Nodo 2	4333,63	6074,6
10	541	Nodo 4	6870,37	9630,51

En la Figura 56 se puede observar esta comparativa de forma gráfica.

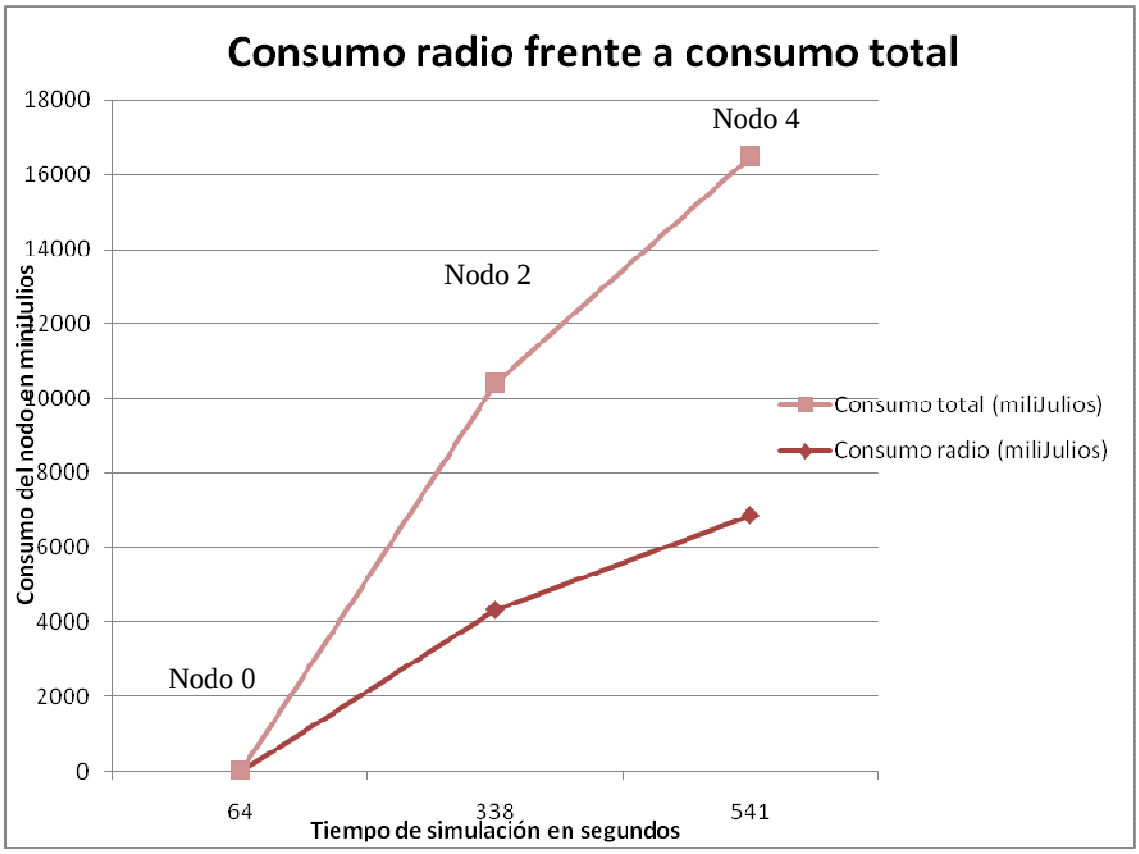

Figura 56: Consumo de la radio frente al consumo total

Se considera que el consumo es adecuado para un protocolo de sincronización que requiere el envío y recepción de mensajes por radio, y la actualización interna de la hora cada 10 segundos.

8. Conclusiones y trabajos futuros

En este capítulo se van a resumir brevemente las principales conclusiones que se han obtenido, las limitaciones y las posibles mejoras.

8.1 Revisión de los objetivos

Tal y como se explicaba en el capítulo 1 el principal objetivo de este proyecto es el diseño e implementación de un protocolo de tiempo que permita que una red de sensores inalámbrica pueda estar sincronizada; para ello era necesario estudiar y comprender qué es una WSN y cómo funciona. Para conseguirlo hubo que estudiar las WSN, los actores que existen en estas redes, las funciones de cada uno de ellos y los protocolos de comunicación que les permiten interactuar.

Por otro lado se tenía que comprender qué es un protocolo de tiempo, en qué consiste y como permite que dos elementos pudieran compartir una hora común sin tener un reloj común.

Estos análisis permitieron esbozar STP (Synchronized Time Protocol). Un protocolo que permite la sincronización de una red de sensores inalámbricos a través de un medio no guiado, para el que no es necesario que un nodo solicite bajo petición la hora de sincronización, sino que el propio protocolo va a intentar sincronizar cada uno de los nodos de la red.

STP permite eliminar un gran número de mensajes de sincronización dentro de la red porque no requiere establecer comunicación con la estación base cada vez que un nodo necesite sincronizarse. Esto tiene gran importancia porque cuantos menos mensajes requiera la sincronización de la red, menos necesidad tendrán los nodos de usar las radios y como consecuencia directa, menos energía consumirán.

Se consideró que en un protocolo de tiempo era más importante que un nodo se pudiera sincronizar al entrar en la red que el ahorro energético, por lo que se dio más preferencia a la tolerancia a fallos.

Por otro lado, fue posible conseguir que STP y SerialSTP fueran dos software fáciles de instalar y usar, permitiendo que cualquier usuario sin conocimientos demasiado avanzados en tecnología WSN y protocolos de tiempo pueda sincronizar una red que ya está en funcionamiento a través de un PC.

8.2 Líneas futuras de trabajo

A medida que se ido desarrollando este proyecto, han surgido bastantes posibilidades de mejora. De hecho, se puede considerar que la implementación realizada de este protocolo de tiempo es un prototipo básico aunque completamente funcional del protocolo STP

A nivel hardware se han encontrado varias mejoras que se podrían tener en cuenta en futuros trabajos, como por ejemplo: se podría usar nodos de dimensiones más reducidas, con un menor consumo de energía (existen placas que requieren menor consumo del microprocesador y de transmisión de datos vía radio). Además, se podrían usar nodos con retroalimentación según la situación medioambiental en la que se encuentren, por ejemplo se podrían desarrollar nodos que se alimentaran de la luz solar o del viento.

A nivel software es tal vez donde se presentan más posibilidades de mejora. El desarrollo de la aplicación se ha realizado sobre la versión 1.x del sistema operativo TinyOS. En la actualidad, existe una nueva versión (versión 2) que proporciona un número mayor de módulos e interfaces para facilitar el desarrollo de las aplicaciones.

La aplicación utiliza el modelo multi-salto para la comunicación lo cual va a provocar que necesariamente la radio de los nodos tenga que estar activa, tal vez sería posible realizar la sincronización de los nodos sin necesidad de usar el algoritmo multi-salto permitiendo que la radio estuviera apagada mientras no estuviera recibiendo o enviando un mensaje.

El protocolo STP permite sincronizar correctamente la red de sensores. Sin embargo, existen escenarios muy específicos que no hemos llegado a controlar. Por ejemplo, si el gateway maestro se estropease y no pudiera enviar la hora de la estación base a la red, pese a que los nodos van manteniendo la hora de forma interna, poco a poco se irían desincronizando, ya que el procesador de los nodos va a tardar algunos milisegundos en calcular la hora actual, así que si la hora exacta son las 12:00;000 y se tiene que actualizar la hora del nodo pero el microprocesador tarda 1 milisegundo en calcular la hora, el nodo considerará que son las 12:00;000 cuando realmente será las 12:00;001; y así se irá produciendo un pequeño desfase.

Dado que nuestro protocolo STP incluye varios niveles de sincronización que evitan que un nodo tenga que dirigirse siempre a la estación base, conseguimos reducir el consumo de energía. Sin embargo, se podría haber contribuido a hacerlo aún más. Por ejemplo, mediante el encendido-apagado dinámico de la radio una vez transmitido el mensaje. Sin embargo, en la versión de TinyOS utilizada, esta característica no está implementada.

Además los nodos no pueden por una cuestión energética calcular la hora real a cada segundo, ya que acortaría considerablemente la vida del nodo por el uso continuo de la radio, así que se decidió que este cálculo se realiza cada 10 segundos. Esto va a provocar que los nodos puedan tener un desfase como máximo de 9 segundos entre actualizaciones. En cualquier caso, si se necesita que la red sea más precisa se puede modificar el tiempo de ejecución de la actualización interna de la hora como se explicó en capítulos anterior.

Por otro lado para que el protocolo STP pueda funcionar es necesario que haya algún nodo en la red que tome el papel de gateway esclavo, si por algún motivo no hubiera ningún nodo con función de gateway esclavo sería necesario reconfigurar la red a través del programa serialSTP por parte de un usuario. Teniendo como limitación adicional que no se puede configurar más de 255 gateways esclavos.

8.3 Conclusiones personales

En este apartado se va realizar un análisis personal de lo que ha supuesto la realización de STP. Desde el punto de vista académico ha sido interesantísimo poder estudiar una tecnología que personalmente no conocía, poder comprender que es una red de sensores inalámbricos, observar las múltiples utilidades que ya tiene y atisbar los desarrollos que puede permitir esta tecnología que prácticamente acaba de nacer.

Además ha sido necesario ampliar los conocimientos que ya tenía sobre los protocolos de tiempo. Estos dos campos de conocimientos (WSN y NTP) son muy extensos y permiten que un desarrollador pueda realizar un software de múltiples maneras, así que el propio análisis y estudio de WSN, STP y del entorno de trabajo: TinyOS, nesC ha conllevado bastante dificultad y esfuerzo.

El objetivo del proyecto ha estado claro desde el primer momento pero la definición del mismo y cómo llevarlo a cabo ha tenido bastantes dificultades que he podido salvar gracias a la ayuda de mi tutora. Y poco a poco se ha podido dar forma al protocolo STP, observando los fallos y mejorando lo ya implementado.

En el aspecto personal, sólo el planteamiento de un proyecto de estas características supuso un gran reto. El hecho de realizar un trabajo de que requiere tanta investigación y estudio de elementos que no conocía ha requerido mucha dedicación. Pero todo ello se compensa al ver este protocolo de tiempo que puede ser utilizado por la comunidad científica, y en el fondo me permite sentir que formo parte de ella. Con la finalización de este proyecto estoy absolutamente satisfecha tanto por el resultado como por el tiempo y esfuerzo empleado.

8.4 Presupuesto del proyecto

En este apartado se va a elaborar el presupuesto del proyecto basado los costes de los recursos utilizados ya sean recursos humanos y recursos materiales.

En el caso de los costes de recursos humanos, se tienen en cuenta cada una de las fases que conforman el proyecto. Aplicando el Modelo en Cascada como herramienta de análisis del ciclo de vida de una aplicación software, se tienen las siguientes fases:

- Análisis del entorno del proyecto y adquisición de conocimientos sobre el estado de la cuestión.
- Análisis del problema y estudio de requisitos.
- Diseño.
- Implementación.
- Evaluación y pruebas.
- Documentación.

Para la evaluación de los costes materiales, se tendrán en cuenta todos los elementos hardware necesarios para la implementación del proyecto, desde el PC donde se desarrolla y se simula, hasta los motes físicos con los que se evalúa.

8.4.1 Recursos humanos

El coste humano (esfuerzo) se establece teniendo en cuenta las siguientes consideraciones:
El proyecto ha sido desarrollado por un único recurso (persona).

Dicha persona trabaja a tiempo completo de lunes a viernes, teniendo un tiempo escaso de dedicación al proyecto durante estos días, por lo que la mayor parte del tiempo ha sido obtenido de los fines de semana, con una media de 18 horas semanales.

Para el desarrollo de este proyecto se ha seguido un modelo de ciclo de vida en cascada. La Figura 57 se muestra el ciclo de vida en cascada con las distintas fases que tiene este proyecto.

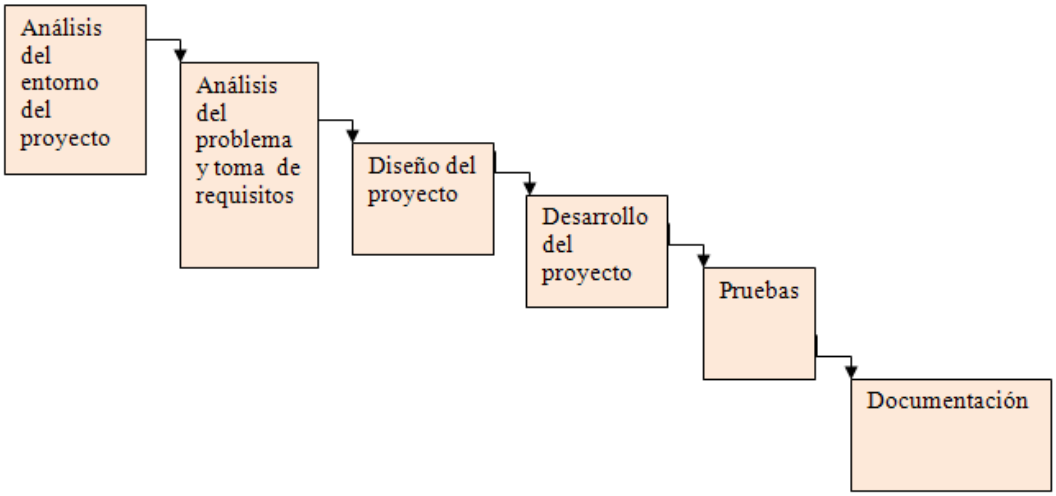

Figura 57: Ciclo de vida del STP

Las fechas de comienzo y finalización del proyecto son del 1 de Noviembre del 2011 al 1 de Septiembre del 2012 respectivamente, lo que hace un total de 792 horas distribuidas en 44 semanas.

Para determinar el coste en términos monetarios de los recursos humanos, se considera una retribución de 40€/hora.

La Tabla 16 muestra la distribución de las horas empleadas en la consecución de cada una de las fases en las que se divide el proyecto:

Tabla 16: Tabla de costes de Recursos Humanos

Fase	**Precio/hora (€)**	**Horas**	**Coste de la fase (€)**
Análisis del entorno	40	90	3.600
Análisis del problema	40	105	4.200
Diseño	40	110	4.400
Implementación	40	162	6.480
Evaluación y pruebas	40	140	5.600
Documentación	40	185	7.400
Total	**40**	**792**	**31.680**

En la Tabla 17 se muestra el desglose del coste para cada persona implicada en el proyecto.

Tabla 17: Desglose coste personal

Apellidos y nombre	N.I.F.	Categoría	Dedicación (hombres mes) [a)]	Coste hombre mes	Coste (Euro)
María Illera Bermejo		Ingeniero Técnico	1	2.694,39	2.694,399
Hombres mes 1				**Total**	2.694,39

8.4.2 Recursos materiales

Todo proyecto software tiene asociados unos costes de materiales y servicios, además de los descritos en el punto anterior. La Tabla 18 muestra el coste unitario y total de cada uno de los elementos necesarios:

Tabla 18: Tabla de recursos materiales utilizados.

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d]
Ordenador portátil	990,00	100	10	60	924,00
Conexión a internet (coste mensual)	48,00	100	10	60	44,80
Impresión del proyecto (en tapa dura)	55,00	100	10	60	51,33
Impresión del proyecto (en espiral)	35,00	100	10	60	32,67
Mote Micaz (MPR2400CA)	100,00	100	10	60	93,33
Placa programadora MIB520	74,55	100	10	60	69,58
Placa sensora MTS310CB	190,156	100	10	60	177,48
	1492,706			**Total**	1.393,19

8.4.3 Costes totales

En la Tabla 19 se muestra un resumen con el coste total del proyecto:

Tabla 19: Costes totales del proyecto

Presupuesto Costes Totales	**Presupuesto Costes Totales**
Personal	2.694
Amortización	1.393
Subcontratación de tareas	0
Costes de funcionamiento	0
Costes Indirectos	818
Total	4.905

9. Bibliografía

En este capítulo se recapitulan toda la bibliografía que ha sido necesaria a la hora de realizar este trabajo. Todas las referencias que se han encontrado a lo largo de este documento se corresponde con el índice de se muestra a continuación.

[1] Wireless Sensor Networks (WSN). Available online: http://www.vs.inf.ethz.ch/publ/papers/wsn-designspace.pdf

[2] NTP. Network Time Protocol. http://www.pool.ntp.org/en/

[3] Intel. http://www.intel.com/#/es_ES_01

[4] Wisewine project: Wireless Sensor Networks for Habitat Monitoring. http://upcommons.upc.edu/pfc/bitstream/2099.1/8776/1/PFC_CD.pdf

[5] CodeBlue. Available online: http://www.eecs.harvard.edu/~mdw/proj/codeblue/

[6] Universidad de Harvard. Available online: http://www.harvard.edu/

[7] Great Duck Island: Pervasive Computing and Proactive Agriculture. http://www.computer.org/portal/web/csdl/doi/10.1109/MPRV.2004.1269130

[8] ARGO - Global Ocean Sensor Network: www.argo.ucsd.edu

[9] MEMSIC. http://www.memsic.com/

[10] XSILOGY Solutions. http://www.xsilogy.com/home/main/index.html.

[11] ENSCO. http://www.in-q-tel.com/tech/dd.html

[12] EMBER. http://www.ember.com

[13] H900. http://www.sensicast.com

[14] SOFTLINK. http://www.soflinx.com

[15] XYZ. http://www.cbe.berkeley.edu/research/briefswirelessxyz.htm

[16] Japan's Omron Corp. http://www.omron.com

[17] MicaZ. Available online: http://www.xbow.com/Products/SelectCountry.aspx?sid=164

[18] MIB600 gateway: http://www.willow.co.uk/html/mib600-_ethernet_gateway.html

[19] TinyOS. Available online: www.tinyos.net

[20] Universidad de Berkeley. Available online: http://www.berkeley.edu

[21] NesC. Available online: http://nescc.sourceforge.net/

[22] The Contiki Operating System - Home. Available online http://www.sics.se/contiki/.

[23] MANTIS: MultimodAl NeTworks of In-situ Sensors. Available online . http://mantis.cs.colorado.edu/index.php/tiki-index.php.

[24] LiteOS Home. Available online. http://www.liteos.net/.

[25] Bertha. Available online: http:/www.media.mit.edu/resenv/pubs/papers/2002-09-PushpinPervasiveWF.pdf

[26] CORMOS: A Communication-Oriented Runtime System for Sensor Networks. Available online. http://www.ics.forth.gr/~bilas/publications/pdffiles/cormos-ewsn05-cr.pdf.

[27] The MagnetOS Operating System. Available online. http://www.cs.cornell.edu/people/egs/magnetos/

[28] C. http://cm.bell-labs.com/cm/cs/who/dmr/chist.html

[29] Unix. http://www.unix.org/

[30] TOSSIM. Available online: http://www.cs.berkeley.edu/~pal/research/tossim.html

[31] TinyViz. Available online: mail.millennium.berkeley.edu/pipermail/tinyos-help/2005-April/009194.html

[32] Avrora. Available online: compilers.cs.ucla.edu/avrora/

[33] Universidad de UCLA. Available online: www.ucla.edu/

[34] Atmel AT45DB011 Serial Dataflash. Available online: http://www.datasheetctalog.com/datasheets_pdf/A/T/4/5/AT45DB.shtml

[35] ZigBee. Available online: www.zigbee.org/

[36] IEE 802.15.4. http://www.ieee802.org/15/pub/TG4.html

[37] Universidad de Delaware. http://www.udel.edu/

[38] Sheng-Po Kuo, Chun-yu Lin , Yueh-Feng Lee Hua-Wei Fang, Y.-W. Peter Hong, Hwa-Chun Lin, Yu-Chee Tseng, Chung-Ta King, Chin-Liang Wang. The NTP experimental platform for heterogeneous wireless sensor networks. http://portal.acm.org/ft_gateway.cfm?id=1390619&type=pdf&coll=GUIDE&dl=GUIDE&CFID=86553805&CFTOKEN=55480160

[39] Jeremy Elson, Deborah Estrin: Time Synchronization for Wireless Sensor Networks. http://www.isi.edu/scadds/papers/timesync.pdf

[40] FTSP. SenSys_1568933812_Maroti_v11.doc

[41] Universidad de Vanderbilt. http://www.vanderbilt.edu/

[42] Serial Forwarder. http://www.tinyos.net/tinyos-1.x/doc/serialcomm/index.html

Anexos

Anexo 1: Instalación del entorno de trabajo

En este anexo se va a describir el entorno de trabajo sobre el que hemos diseñado e implementado el protocolo STP.

Se va a describir tanto la instalación de las aplicaciones que han formado el entorno como el uso de ellas.

Máquina Virtual

Para poder implementar STP se necesita tener un sistema operativo basado en Unix y además perfectamente compatible con TinyOS.

Se disponen de varias posibilidades a la hora de definir el entorno.
Como primera alternativa se puede instalar en un equipo un sistema operativo Linux como Debian o Ubuntu, o instalar una máquina virtual.

En este proyecto se ha elegido instalar la máquina virtual porque permite mantener el sistema operativo original del ordenador y heredar los drivers y las conexiones, simplemente se tiene que "puentear". De esta manera se mantienen configurados los puertos UBS y la conexión a internet sin tener que configurarlo en Linux.

El siguiente punto que se tiene que definir era el sistema operativo a usar en la máquina virtual. Este sistema operativo debe ser un sistema basado en Unix, ya que es uno de los requisitos del proyecto. Tras estudiar varias posibilidad se opta por Xubuntos, que es una unión de Xubuntus y TinyOS, tal y como se muestra en la Figura 58.

Figura 58: Logo xubuntos

Estos son los paquetes que contiene XubunTOS:

- Xubuntu 6.10
- TinyOS 2.0.1 paquetes Debian
- + TinyOS 1.x CVS repositorio

Referencia: http://toilers.mines.edu/Public/XubunTOS

Una vez definidos estos puntos se puede empezar a preparar el entorno:

Para el proyecto se elige el programa VMWare Server para manejar la máquina virtual siguiendo las instrucciones de la web: *http://klueska.doesntexist.com/installing_xubuntos_vm.html*

Y se descarga la una imagen ISO de un disco XubunTOS 2.0 desde el repositorio: http://5secondfuse.com/tinyos/XubunTOS-2.0.iso

Esta imagen de CD contiene un ejecutable que instala todas las librerías necesarias para que TinyOS funcione (java 1.5, las librerías de nesC, msp430, avr_lib, avr_gcc….). Una vez finalizada la instalación se tiene que instalar TinyOS 1.x para ello es necesario descargarlo desde el repositorio cvs a través de los comandos:

```
cvs -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/tinyos login
```

```
cvs -z3 -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/tinyos co tinyos-1.x
```

Ambos comando se ejecutan desde un terminal (o ventana de comandos) dentro de la máquina virtual y descargan e instalan tinyOS 1.15.

Una vez instalado todo se puede compilar cualquier código en NesC tanto para tinyOS 1.x como para tinyOS 2.x. Simplemente para cambiar de entorno se tiene que ejecutar el script: *tos1* para carga tinyOS 1.x o *tos2* para cargar tinyOS 2.x, sabiendo que por defecto el entorno cargado es tinyOS 2.x

TinyOS 1.x

TinyOS en un sistema operativo para trabajar con redes de sensores, desarrollado en la Universidad de Berkeley. TinyOS puede ser visto como un conjunto de programas avanzados, el cual cuenta con un amplio uso por parte de comunidades de desarrollo, dada sus características de ser un proyecto de código abierto.

Este "conjunto de programas" contiene numerosos algoritmos, que permitirán desde generar enrutamientos, como también aplicaciones pre-construidas para sensores. Además soporta diferentes plataformas de nodos de sensores, arquitecturas bases para el desarrollo de aplicaciones.

En la Tabla 20 se proporcionan las referencias básicas sobre TinyOS

Tabla 20: Referencias TinyOS

Dirección	Contenido
http://www.tinyos.net	Página Principal del Proyecto.
http://www.tinyos.net/download.html	Sección donde se pueden bajar las últimas versiones del sistema operativo
http://www.tinyos.net/support.html	Página con el soporte para TinyOS
http://www.tinyos.net/tinyos-1.x/doc/tutorial/index.html	Tutorial on-line.
http://sourceforge.net/projects/tinyos/	Control de Versiones (CVS) de TinyOS, en el se pueden encontrar tanto aplicaciones que se encuentran en desarrollo para implementarse con TinyOS, como el desarrollo del código fuente de éste.

El lenguaje en el que se encuentra programado TinyOS es un meta-lenguaje que deriva de C, cuyo nombre es NesC. Además existen variadas herramientas que ayudan el estudio y desarrollo de aplicaciones para las redes de sensores, que van desde aplicaciones para la obtención y manejo de datos, hasta sistemas completos de simulación.

En la Figura 59 se muestra la estructura de directorios de TinyOS:

Directorio	Descripción
/tos/interfaces	Contiene todas las interfaces que son proporcionadas por los componentes primitivos y por las aplicaciones de ejemplo
/tos/lib	Contiene librerías para resolver determinados problemas
/tos/system	Contiene todos los componentes primitivas que proporciona TinyOs
/tos/types	Contiene los tipos que se utilizan en las primitivas de TinyOs
/tos/platform	Contiene los ficheros necesarios para la ejecución en las diversas plataformas
/tos/senseorboard	Contiene los ficheros que son específicos de cada placa.

Figura 59: Estructura de directorios

Además en la ruta: `/apps` TinyOS tiene un repositorio con varias aplicaciones a modo de ejemplo.

Anexo 2: Archivo Makefile

En este anexo se muestra el fichero Makefile que se definió para la compilación del protocolo STP.

```
PLATFORMS=mica mica2 mica2dot micaz pc
COMPONENT=Cliente_ServidorC
#SENSORBOARD=micasb

PFLAGS= -I%T/lib/Route -I%T/lib/Queue

include ../Makerules
```

Anexo 3: TinyViz

TinyViz es una herramienta escrita en lenguaje Java, que permite visualizar, controlar, y analizar una simulación de TOSSIM. Provee una interacción en línea, al ser capaz de modificar parámetros como ADC, el modo de la comunicación inalámbrica, etc., propios tanto de cada nodo como de las redes de sensores.

Esta herramientas se encuentra en la ruta `opt/tinyos-1.x/tools/java/net/tinyos/sim/` y para que se pueda simular una aplicación implementada para TinyOS 1.x en el sistema operativo Xubuntus es necesario que previa a la ejecución de TinyViz se cargue el entornos de TinyOS 1.x y para ello se lanza el comando `tos1` en la Shell.

Una vez se haya realizado este paso previo se ha de posicionar en la ruta:
`opt/tinyos-1.x/tools/java/` donde se ejecutará la orden

```
./net/tinyos/sim/tinyviz
```

En la Figura 60 se muestra la pantalla inicial de TinyViz.

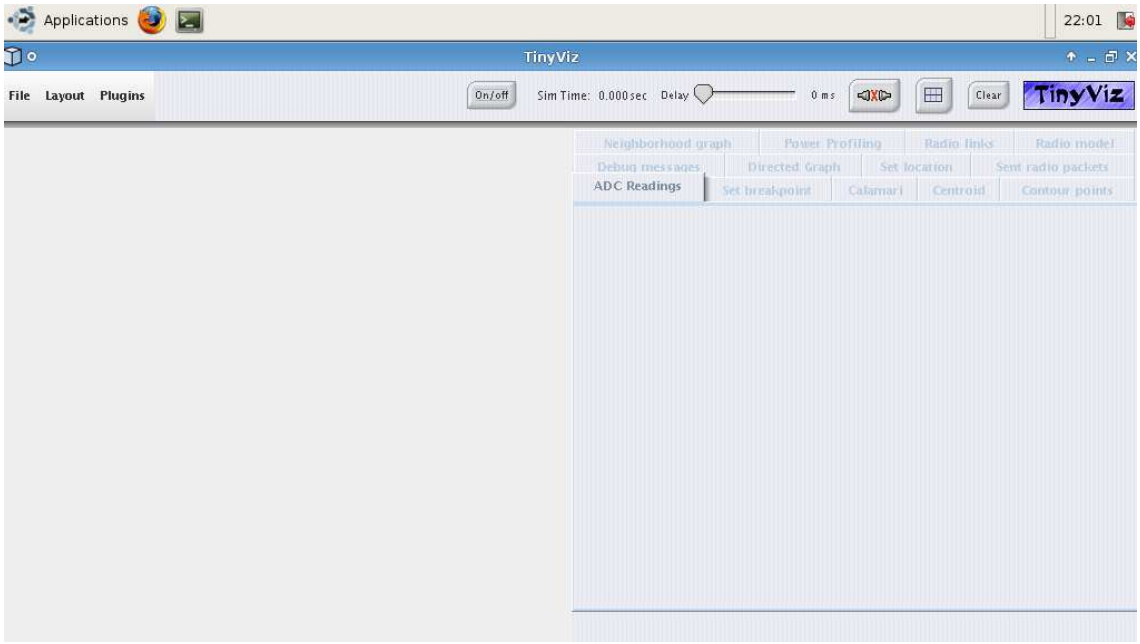

Figura 60: Pantalla inicial TinyViz

TinyViz posee un sistema de plugins que se encuentran es `tools/java/net/tinyos/sim/plugins` y permiten una mejor simulación al considerar variables físicas en la simulación. De esta forma, el usuario interactúa con la simulación habilitando (cargando) los plugins que considere necesarios para la simulación del software.

Los plugins se pueden clasificar en dos tipos, dependiendo de su direccionalidad:

- Envían mensajes a TOSSIM
- Reciben salidas generadas por TOSSIM

.
La Figura 61 muestra el aspecto de la interfaz gráfica de TinyViz.

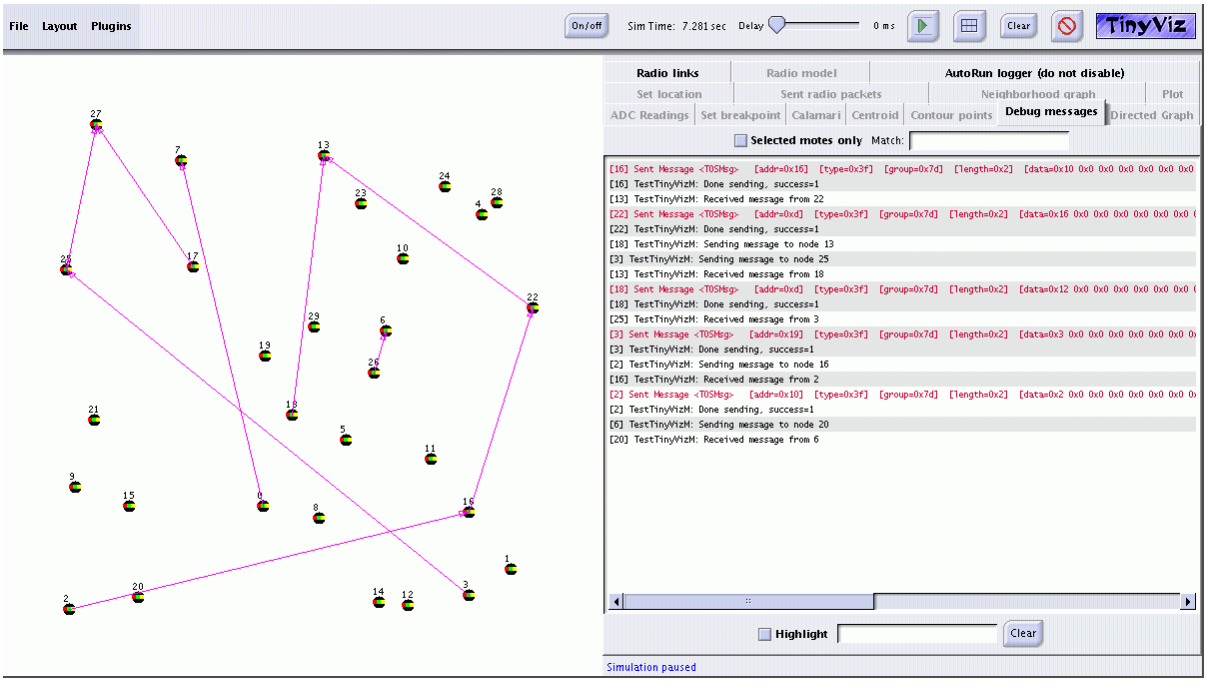

Figura 61: GUI de la aplicación TinyViz

Los plugins disponibles son:

- Debug messages: Muestra una ventana con todos los mensajes de debug generados por la simulación. Se pueden seleccionar nodos y también aplicar filtros para seleccionar qué tipo de mensaje se desea observar.
  Es importante mencionar que los mensajes mostrados dependen del valor asignado a la variable DBG, que se le asignó al ejecutar TOSSIM.
  Por ejemplo:

  ```
  DBG=usr1,am tinyviz -run build/pc/main.exe 30
  ```

  Sólo mostrará los mensajes del tipo AM y el definido por usr1.

- Set breakpoint: Permite agregar breakpoints, lo que pausará la ejecución de la aplicación cuando la condición se cumpla.

- ADC readings: Muestra los valores más recientes de cada ADC.

- Sent radio packets: Muestra el tráfico de paquetes en la red.

- Radio links: Muestra gráficamente la actividad de los mensajes. Los mensajes broadcast se muestran con un círculo azul y los enviados a un mote en particular (punto a punto) con una línea roja entre los nodos.

- Set location: Realiza una localización virtual de los nodos, disponible gracias a la interfaz `apps/TestTinyViz/Location.nc`. Existe la posibilidad de cargar mapas de fondo en la simulación, o agregar una malla, la cual está graduada en una escala de distancia.

- Radio model: Configura la tasa de error de bits entre cada nodo, acorde a su localización y al modelo de conectividad que se está ocupando. Habilitar este plugin permite tener una simulación más realista. Existen básicamente dos modelos:

- Empirical: Basado en una conexión en lugar abierto, a través de la radio RFM1000 (integrada en la mote Mica2).

- Fixed radius: Todos los nodos dentro de una distancia delimitada tienen conectividad, mientras que el resto no la tiene. Al configurar el *scaling factor* en el panel de control, los parámetros de distancia del modelo a simular se ordenan siguiendo una escala.

Anexo 4: Cygwin

Cygwin es un entorno de emulación Unix/Linux para máquinas Windows. De manera que permite trabajar en un entorno Unix sin tener que tener instalado ese sistema operativo en el equipo.

Cygwin es un software libre que se puede descargar desde la página http://www.cygwin.com/ en la que además del instalador se puede encontrar documentación sobre su uso e instalación.
Se compone de dos partes:

- Un archivo DLL (cygwin1.dll), librería dinámica, que actúa como una capa de emulación del API Linux.
- Una colección de herramientas que proporcionan un aspecto y modo de trabajo como si de Linux se tratara.

Cygwin es un interfaz de usuario opcional para compilar y descargar aplicaciones para los motes. La Shell de cygwin se ejecuta haciendo doble click sobre el icono situado en el escritorio.

A continuación aparecerá una nueva ventana con el *prompt* de usuario esperando a leer comandos.

En la Tabla 21 se describen una serie de comandos básicos del sistema operativo Linux.

Tabla 21: Comandos básicos de Linux

Descripción	Comando
Moverse un directorio hacia arriba	. . /
Moverse dos directorios hacia arriba	. . / . . /
Ir a un sub-directorio llamado “midirectorio”	cd midirectorio
Listar todos los ficheros y directorios	Ls
¿Dónde está el ejecutable?	which <ejecutable>
Mostrar todas las variables de entorno	Set
Añadir una variable de entorno	export MYHOME=c:/mydev/apps
Mostrar una variable de entorno	echo $MYHOME
Borrar una variable de entorno	unset MYHOME
Compilar para plataforma micaz	make micaz
Compilar e instalar para plataforma micaz	make micaz install
Compilar e instalar para plataforma micaz el nodo con id=0	make micaz install, 0
Instala la aplicación precompilada	make micaz reinstall
Genera diagramas de componentes en formato HTML	make micaz docs