UACM

Universidad Autónoma
de la Ciudad de México
Nada humano me es ajeno

COLEGIO DE CIENCIA Y TECNOLOGÍA

LICENCIATURA EN INGENIERÍA EN SISTEMAS ELECTRÓNICOS

Y DE TELECOMUNICACIONES

"Implementación de un protocolo de ruteo eficiente en el uso de energía para redes de sensores"

TRABAJO RECEPCIONAL

PARA OBTENER EL TÍTULO DE LICENCIADO EN

INGENIERÍA EN SISTEMAS ELECTRÓNICOS Y DE TELECOMUNICACIONES

PRESENTAN:

Luciano Romero Ronquillo y Melissa Maricela Ibarra Puebla

Directora del trabajo recepcional

M. en C. Magali Cortez Vázquez

México, D.F. Agosto 2014

UACM
Universidad Autónoma de la Ciudad de México
Nada humano me es ajeno

CA Coordinación Académica

SISTEMA BIBLIOTECARIO DE INFORMACIÓN Y DOCUMENTACIÓN

UNIVERSIDAD AUTÓNOMA DE LA CIUDAD DE MÉXICO
COORDINACIÓN ACADÉMICA

RESTRICCIONES DE USO PARA LAS TESIS DIGITALES

DERECHOS RESERVADOS©

RESUMEN de la tesis de **Luciano Romero Ronquillo** y **Melissa Maricela Ibarra Puebla**, presentada como requisito parcial para la obtención del grado de **LICENCIADO EN INGENIERÍA EN SISTEMAS ELECTRÓNICOS Y DE TELECOMUNICACIONES**. México, D.F. Agosto 2014.

IMPLEMENTACIÓN DE UN PROTOCOLO DE RUTEO EFICIENTE EN EL USO DE ENERGÍA PARA REDES DE SENSORES

Resumen aprobado por:

M. en C. Magali Cortez Vázquez

Directora de Tesis

En el presente trabajo se programó y simuló un protocolo eficiente en el consumo de energía para redes inalámbricas de sensores (RIS) en el sistema operativo TinyOS. Se utilizó el lenguaje de programación NesC y la simulación se realizó con el software TOSSIM. El protocolo programado se basa principalmente en el protocolo EARWSN (*Energy Aware Routing for Wireless Sensor Networks*) y se complementa con el protocolo AODV (*Ad-Hoc On-demand Distance Vector*), ambos orientados al bajo consumo de energía. El primer protocolo se utilizó para descubrir las rutas disponibles entre un nodo concentrador y un nodo destino determinado mientras que la construcción de la ruta de regreso del nodo destino hacia el nodo concentrador está basado en el funcionamiento del protocolo AODV. De esta manera, se crean rutas bidireccionales para obtener información de cualquier nodo de la red desde el nodo concentrador.

La red considerada para la simulación es una red estática con topología tipo ad hoc. Físicamente, los nodos de la red pueden ser motes de la familia MICAz, Telosb, o cualquier familia que utilice el transceptor CC2420. Sin embargo, dado que para la realización de este trabajo no se contó con dichos motes, únicamente se generó el archivo ejecutable del programa.

Cabe resaltar que la mayoría de las tecnologías inalámbricas disponibles en el mercado están basadas en la especificación técnica ZigBee, la cual trabaja bajo un protocolo de enrutamiento de tipo inundación simple que no está orientado a la conservación de energía. De esta manera el protocolo propuesto representa una alternativa para las aplicaciones que requieran de esta característica.

Palabras clave: AODV, Energy Aware Routing for Wireless Sensor Networks, TOSSIM, TinyOS, MICAz.

Dedicatoria

Dedicada a todas aquellas personas que con su ayuda han colaborado en la realización del presente trabajo.

Agradecimientos

Antes que nada queremos agradecer a la Universidad Autónoma de la Ciudad de México UACM, por la oportunidad que nos otorgó para realizar nuestros estudios a nivel superior.

Asimismo queremos agradecer el tiempo y la dedicación que nos brindaron cada uno de nuestros lectores Yazbek Buendía Gómez, José Ignacio Castillo Velázquez, Marco Antonio Noguéz Córdoba y José Luis Quiroz Fabián, pero sobre todo a nuestra directora del trabajo recepcional la profesora Magali Cortez Vázquez quien estuvo al pendiente del desarrollo de este proyecto.

De la misma forma queremos agradecer a los maestros que nos apoyaron en nuestra formación académica. A los profesores que integran la academia de ingeniería de sistemas electrónicos y de telecomunicaciones ISET, gracias por su apoyo Eduardo Ramos y Daniel Tapia. También queremos agradecer a todos nuestros compañeros que estuvieron durante este proceso.

Finalmente agradecemos a la UACM por su apoyo al otorgarnos la Beca de obtención de grado, la Beca de impresión de trabajo recepcional con el acuerdo UACM/CU-3/EX-12/118/13 y la Beca de licenciatura del Convenio específico de colaboración UACM/OAG/ADI/014/2011.

Melissa Ibarra:

Con todo mi cariño y mi amor para las personas que hicieron todo en la vida para que yo pudiera lograr mi sueño, por motivarme, por sacrificar su tiempo para que yo pudiera cumplir con el mío, por su bondad, por su paciencia, por su comprensión y por su sabiduría que influyeron en mi para lograr este objetivo, es para ustedes este trabajo recepcional en agradecimiento por todo su apoyo.

A Dios, por acompañarme todos los días.

Mi madre Gloria Puebla Gómez, por ser mi mejor amiga, mi aliada, mi ejemplo. Gracias por todo el apoyo y el esfuerzo en este proyecto y en mi vida.

El mejor hermano José Eduardo Ibarra Puebla, gracias por tu infinita paciencia, por tu compañía y tu inagotable apoyo, gracias por compartir mi vida y mis logros.

Mi padre José Ibarra Vega, gracias por tu apoyo y tus enseñanzas, en esta travesía.

Mi tío Bernardo Ibarra Vega, gracias por tus oraciones, por haberme enseñado el valor de la familia, gracias por ser mi amigo, mi fortaleza, mi cómplice, gracias por estar conmigo siempre, por tu confianza, tú respeto, pero sobre todo tu enseñanza a que si se trabaja duro es posible cumplir todo aquello que nos propongamos en la vida.

A la profesora Magali Cortez Vázquez que por su compromiso, esfuerzo y dedicación se logró la conclusión del trabajo recepcional. Pero sobre todo por su sabiduría, sus consejos y el apoyo que siempre me brindó durante el desarrollo del mismo.

Mi compañero de este trabajo recepcional Luciano Romero, por su apoyo y motivación.

Mis amigas Lilibeth Rivera, Sharlene Mbombo, Dara López y familiares quienes me brindaron su tiempo, su motivación y apoyo incondicional, gracias por estar conmigo, por su confianza y cariño.

Luciano Romero:

Ante todo a mis padres Irma y Herminio por el apoyo incondicional que me han bridado toda la vida, a mis hermanas Olga y Luisa por sus consejos acertados, a mis sobrinos Luis, Sabrina y Diego que me inyectan las ganas de seguir adelante, a mi abuelita Juvencia por consentirme desde que era un bebe y hasta la actualidad.

A Magali por haberme tenido la confianza y haberme brindado las herramientas necesarias para hacer un trabajo de este tipo, gracias por todas tus enseñanzas y sobre todo gracias por ser mi amiga.

A Melissa por ser la compañera ideal para este trabajo, gracias por aguantarme te lo agradezco infinitamente.

A mis compañeros de la carrera que se preocupaban y apoyaron a mi persona dentro y fuera del aula son tantos que no quisiera mencionar sólo algunos ya que todos fueron importantes para poder concluir mi carrera.

A mis Warriors, mi grupo de amigos que siempre estuvo ahí cuando necesitaba olvidarme del mundo académico y laboral, ustedes han sido una parte fundamental en mi vida.

A Danahe por ser mi compañero en esta y varias batallas en las que hemos salido victoriosos, ahora te toca a ti concluir esta etapa.

Le agradezco a Dios por haberme dado la fuerza necesaria y ser mi guía todo este tiempo.

Contenido

Capítulo IV

Capítulo V

Capítulo VI

Apéndice A

Lista de figuras

Lista de tablas

Capítulo I

Introducción

1.1 Antecedentes

El uso de sensores en sistemas de comunicación electrónica se ha generalizado. Por ejemplo, mientras se llevaba a cabo la segunda guerra mundial, en 1948 Estados Unidos instaló el SONAR (del inglés SONAR, acrónimo de *Sound Navigation and Ranging,* 'Navegación y Alcance por Sonido'), que es un sistema de sensores basado en la propagación del sonido bajo el agua, usado principalmente para navegar, comunicarse o detectar otros buques. Por otro lado, durante la guerra fría, la necesidad de detección de amenazas marítimas se hizo prioritaria, por lo que la marina de los Estados Unidos instaló el SOSUS (*Sound Surveillance System, '*Sistema de vigilancia de Sonido'), en las profundidades marítimas de los océanos Atlántico y Pacífico. Compuesto de una matriz de sensores con capacidad de detección de largo alcance, el sistema resultó un éxito en el descubrimiento de submarinos enemigos, [1]. Sin embargo, ninguno de estos sistemas tenía la capacidad de formar una red ni de comunicarse a través de ella. Es hasta el año de 1978 cuando la DARPA (del inglés DARPA, acrónimo de Defense Advanced Research Projects Agency, 'Agencia de Proyectos de Investigación Avanzada de la Defensa de los Estados Unidos'), encargada

del estudio y desarrollo de nuevas tecnologías para el uso militar, organizó el taller de redes de sensores distribuidos, cuyo objetivo era estudiar y resolver los principales retos en la investigación de redes de sensores, así como en las diferentes técnicas de procesamiento y de algoritmos distribuidos.

En 1980, dicha agencia comienza el programa de DSN (del inglés DSN, acrónimo de *Distribuited Sensor Networks, '*redes distribuidas de sensores'), donde Robert Kahn, director de la oficina de Información de técnicas de procesamiento, planteó la posibilidad de extender los beneficios de la entonces ARPANET (del inglés ARPANET, acrónimo de *Advanced Research Projects Agency Network, '*Agencia de Redes de Proyectos de Investigación Avanzada'), a las redes de sensores, pensando en una red compuesta por sensores distribuidos que podían detectarse y comunicarse entre ellos, operando de forma autónoma.

Posteriormente en 1996, la Universidad de California y el Centro Rockwell de ciencia proponen el concepto de WINS (del inglés WINS, acrónimo de *Wireless Integrated Network Sensors*, 'redes inalámbricas integradas por sensores') [1].

En 1998, tras el surgimiento de las redes totalmente inalámbricas llamadas Ad-Hoc, DARPA inicia una nueva investigación con dos objetivos principales, el primero, desarrollar nuevas técnicas de red para entornos dinámicos Ad-Hoc ya que, en el campo de batalla, los nodos sensores requerían un despliegue rápido; y segundo, el procesamiento de información de red, es decir, cómo extraer la información útil, oportuna y confiablemente a través de la red de sensores. Este proyecto fue nombrado SensiTI, Tecnología de Información de Sensor.

En el año 2001, el Dr. Kristofer Pister concibió la idea de *Smart Dust* (del inglés *Smart Dust*, acrónimo de 'polvo inteligente'), la cual estaba enfocada a integrar todo un sistema de comunicación inteligente con sensores autónomos y su propio suministro y almacenamiento de energía en una pequeña plataforma de un milímetro cúbico. Este sistema estaba orientado al ahorro de energía para prolongar la vida de las baterías de los motes, alimentadas por celdas solares. La idea consistía en que un mote (del inglés mote: *re**mote** sensing*, 'teledetección') almacenara energía a través de las celdas y posteriormente se ponían a trabajar por periodos de tiempo, durante el día se cargaban las baterías y durante la noche los motes se enfocaban a capturar información, esto dependía de la aplicación a la que estuviera destinada la red de sensores.

En el año 2003, el IEEE (del inglés IEEE, acrónimo de *Institute of Electrical and Electronics Engineers*, 'Instituto de Ingenieros en Electricidad y Electrónica'), aprueba el estándar 802.15.4-2003, LR-WPAN (del inglés LR-WPAN, acrónimo de *Low Rate-Wireless Personal Area Network*, 'redes inalámbricas de área personal de baja velocidad'), de la que se desprende la revisión IEEE 802.15.4-2006 [1]. Este estándar fue creado con el objetivo de transmitir información a distancias relativamente cortas, con poca o ninguna infraestructura, además brinda soluciones a la eficiencia energética para dispositivos de tamaño reducido para implementarse en una amplia gama de aplicaciones en redes inalámbricas de área personal. De este modo, las redes de sensores se consideran WPAN (del inglés WPAN, acrónimo de *Wireless Personal Area Networks*, 'redes inalámbricas de área personal'), dado que su cobertura es pequeña, comúnmente cubren distancias del orden de 10 metros como máximo. Normalmente las WPAN son usadas para conectar dispositivos portátiles personales sin la necesidad de usar cables y, debido a que trabajan a bajas tasas de transmisión de

datos, tienen como resultado un bajo consumo de energía, haciendo a la tecnología WPAN adecuada para el uso de dispositivos móviles pequeños que funcionan con baterías.

Existen varios grupos de trabajo para la tecnología WPAN, cada uno de ellos con características e intereses específicos de comunicación. Tal es el caso del estándar IEEE 802.15.4-2006, enfocado a bajas tasas de transmisión y un mínimo consumo de energía con mejoras con respecto a la versión 2003. Por otra parte, en el 2004 Zigbee Alliance introduce ZigBee, la cual es una tecnología inalámbrica que complementa a nivel de red y aplicación el estándar 802.15.4, [2].

Por otro lado, con el desarrollo y aparición de nuevas tecnologías, comenzaron a fabricarse sensores más pequeños y el costo de los mismos disminuyó, lo que permitió comenzar a desplegar nuevas aplicaciones para sectores diferentes al militar, entre las que se encuentran la monitorización del entorno, la monitorización de seguridad, el tracking y las redes hibridas, que es la fusión de alguna de ellas.

En la actualidad, existen diferentes desarrolladores para redes inalámbricas de sensores, que fabrican sensores con sistemas MEMS (del inglés MEMS, acrónimo de *Microelectromechanical Systems*, micro- electromecánicos), que brindan una mayor detección y monitoreo del entorno de estudio. Empresas como Crossbow, Sensoria, WorldSens, DusNetworks y Ember Corporation, ofrecen dispositivos para diversos escenarios de aplicación, herramientas de gestión y herramientas para la visualización de información adquirida por el sensor.

Por otro lado, debido a que los nodos sensores interactúan e intercambian información con otros nodos, el consumo de energía es una cuestión crítica ya que generalmente es un recurso limitado. Por ello, es necesario implementar protocolos para la capa de red, de capa física y de capa de enlace de datos que ayuden a suministrar un consumo eficiente de este recurso. El consumo de energía de una red depende, entre otros aspectos, de su topología. Las topologías dinámicas consumen más energía que las redes de topología estática ya que, por su constante auto-organización los nodos de la red deben buscar nuevas rutas para enviar su información a un nodo concentrador o de destino, lo que requiere consumo de energía. Por ello, es importante contemplar el gasto energético en cada uno de los nodos para descubrir la mejor ruta hacia el nodo destino, eligiendo algún tipo de enrutamiento que minimice dicho consumo.

Las redes de sensores utilizan protocolos de enrutamiento debido a que los nodos no tienen conocimiento de la topología de red, por lo tanto deben descubrirla y lo hacen a través de estos protocolos. La idea básica es que cuando un nuevo nodo se agregue a una red, éste anuncie su presencia y sea capaz de escuchar los mensajes de sus vecinos. De esta manera, los nodos se informan de los nuevos nodos a su alcance y a qué otros nodos pueden acceder a través de éstos. Existen diversos tipos de protocolos de enrutamiento eficientes en el uso de energía, como el enrutamiento plano, jerárquico y por localización. En el enrutamiento plano todos los nodos de la red desempeñan el mismo papel, es decir, procesan la misma información al mismo tiempo y colaboran entre sí para llevar a cabo el proceso de medición de magnitudes físicas necesarias. En contraste, el enrutamiento jerárquico selecciona a los nodos de mayor energía para procesar la información y de esa forma se construye una ruta escalonada. Por su parte, los protocolos basados en la localización, son protocolos que tienen por

objeto ahorrar energía manteniendo dormidos a los nodos sino existe actividad alguna en la red.

1.2 Motivación del proyecto

A la par del desarrollo de las redes inalámbricas de sensores, las herramientas para diseñar y simular dichas redes también fueron generándose, tal es el caso del sistema operativo TinyOs diseñado por el Dr. Culler junto con un equipo de investigadores de la Universidad de Berkley, en la etapa de desarrollo del proyecto *Smart Dust*, [4].En TinyOs es posible implementar protocolos para redes de sensores o utilizar protocolos ya implementados como son *Dissemination y Collection*. *Dissemination* es un protocolo que asegura la entrega fiable de datos a todos los nodos de una red, permitiendo al usuario administrar, configurar y restablecer redes. El *Collection*, es el complemento del Dissemination, es decir, recolecta los datos generados por los nodos de la red para enviarlos a un nodo concentrador. Sin embargo, los protocolos antes mencionados están basados en protocolos de enrutamiento de tipo inundación simple sin restricciones en el consumo de energía.

Por otro lado, cabe resaltar que la mayoría de las tecnologías inalámbricas disponibles en el mercado, basadas en la especificación técnica ZigBee también usan un protocolo de enrutamiento de tipo inundación simple. De esta manera la implementación en TinyOs del protocolo que se propone en este trabajo, representa una alternativa para desarrollar aplicaciones que requieran de esta característica y que sean compatibles con dicho sistema operativo.

1.3 Objetivos

1.3.1 Objetivo general

El objetivo general de este trabajo es implementar en TinyOs un protocolo de enrutamiento eficiente en el uso de energía basado en el funcionamiento de los protocolos EARWSN y AODV para generar un código que pueda ejecutarse en motes basados en el estándar IEEE 802.15.4. La evaluación del desempeño de dicho protocolo en una red con dichos motes no forma parte de este trabajo.

1.3.2 Objetivos particulares

- Crear los algoritmos necesarios para programar el protocolo. Para ello se diseño el algoritmo para el descubrimiento de ruta (RREQ), los algoritmos para la confirmación de ruta (ACKImp y ACKExp), todos ellos basados en el protocolo EARWSN y el algoritmo para la respuesta de ruta (RREP), este último basado en AODV.
- Programar y generar un código ejecutable a partir de dichos algoritmos para motes de la familia MICAz mediante el lenguaje de programación NesC del sistema operativo TinyOs.
- Simular una RIS en el simulador TOSSIM de TinyOs mediante el lenguaje de programación Python, incluyendo la configuración de un entorno de simulación adecuado para comprobar el funcionamiento del protocolo de enrutamiento programado.

1.4 Organización de la tesis.

El capítulo dos está dedicado a explicar detalladamente las redes inalámbricas de sensores, abarcando desde su definición, características principales, tipos de arquitectura, hasta los distintos ámbitos de aplicación. Asimismo se explican brevemente los protocolos de red y los estándares de mayor uso en redes inalámbricas de sensores enfocados a características como son la baja tasa de transmisión y el bajo consumo de energía, características esenciales tomadas en cuenta para la implementación de la RIS en este trabajo. Además se detallan los protocolos en los que se basan los mismos, EARWSN y AODV.

En el capítulo tres se explica ampliamente el estándar 802.15.4, que establece la capa física y la capa de enlace de datos. Este es el estándar más usado para desarrollar aplicaciones en motes para RIS, debido a su bajo consumo energético y su baja tasa de transmisión de datos.

En el capítulo cuatro se presentan los motes MICAz en los cuales es posible instalar el protocolo desarrollado y se describen las herramientas empleadas para la programación y simulación del código de este trabajo.

En el capítulo cinco se presenta el proceso de programación mediante la explicación de los algoritmos propuestos para este trabajo. Por otro lado, se explica el proceso de simulación en TOSSIM.

Finalmente en el capítulo seis se encuentran las conclusiones de este trabajo y el trabajo futuro.

Capítulo II

Redes inalámbricas de sensores

2.1 ¿Qué son las redes inalámbricas de sensores?

Una red inalámbrica de sensores es un conjunto de sensores autónomos llamados también motes o pods, distribuidos en un área determinada que se comunican entre sí de manera inalámbrica. Los sensores son dispositivos que capturan información a través de la detección y monitoreo constante del entorno de interés. Esta tecnología puede usar microprocesadores muy potentes y MEMS para proporcionar baja demanda de energía.

La recolección de datos es posible debido a que los sensores son capaces de adquirir, almacenar y transmitir información de un nodo a otro, a partir de dispositivos inalámbricos. La información que se emita o transmita puede ser interpretada por software de bajo costo y centralizada en un banco de información para un futuro análisis.

De manera general, los elementos que integran a una RIS son: los nodos inalámbricos, los nodos de enlace y la estación base, como se muestra en la figura 2.1. Los nodos inalámbricos que integran una red de sensores suelen ser de tamaño reducido y se alimentan con diferentes tipos de baterías, por ejemplo AA, de botón e incluso algunos pueden utilizar celdas solares. Por su lado, los nodos de enlace también

llamados *gateway*, son los encargados de interconectar diferentes tipos de redes para obtener información de ellas. Finalmente, la estación base se encarga de recolectar los datos de una red, con el objetivo final de que los usuarios puedan tener acceso a éstos y procesarlos. Esta infraestructura define a una red de adquisición y distribución de datos monitorizada.

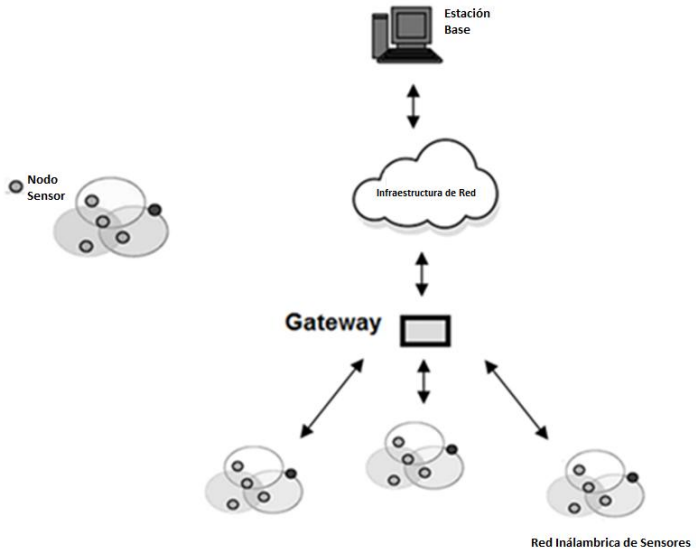


Figura 2.1 Red Inalámbrica de Sensores (RIS), [1].

2.2 Descripción de un nodo sensor.

Un nodo sensor es un dispositivo electrónico integrado por un procesador, una fuente de energía, un radio transceptor y un sensor, capaz de formar una red Ad-hoc [5], arquitectura de red que se explicará en el siguiente apartado. En la figura 2.2 se muestra un diagrama de bloques que representa el funcionamiento básico de un nodo sensor. El primer bloque se refiere a la adquisición de información por medio de un sensor, el segundo bloque representa la conversión de información adquirida dado que el sensor genera una señal analógica que debe convertirse a una señal digital para su procesamiento; el tercer bloque representa un microprocesador, el cual puede ser un

micro-controlador o un FPGA, en este bloque se procesa la información obtenida por el sensor. Por último, el cuarto bloque representa al transmisor del nodo que envía la información hacia los nodos cercanos o vecinos para que pueda llegar al nodo encargado de recolectarla.

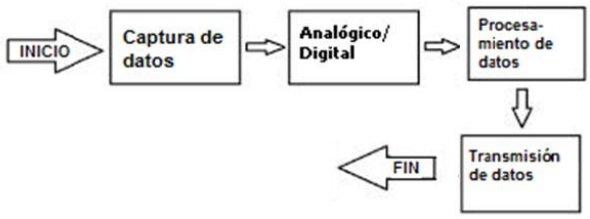


Figura 2.2 Diagrama de bloques del nodo sensor.

De manera general, la alimentación por baterías es la fuente principal de energía de los nodos. Su consumo depende de la actividad de los sensores, la comunicación y el procesamiento. Por ejemplo, el consumo energético de las RIS con motes MICAz en una transmisión de 100 horas continuas puede ser aproximadamente de 20mA, [9].

Por ser una comunicación inalámbrica, los nodos utilizan la banda de frecuencia no licenciada llamada ISM (del inglés ISM, acrónimo de *Industrial, Scientific and Medical*, 'Industrial, Científica y Médica'), con rangos de frecuencias de 902 MHz a 928 MHz, 2.4 GHz a 2.4835 GHz y 5.725 GHz a 5.85 GHz respectivamente, permitiendo enviar y recibir datos en el radio de cobertura de cada nodo.

La memoria en un nodo sensor, está integrada al microcontrolador y cuenta con poca capacidad de almacenamiento, aproximadamente de 128 KB, [10]. Las memorias óptimas para las RIS son de tipo flash preferentemente y EEPROM. Éstas son programadas mediante impulsos eléctricos, tienen capacidad de lectura y escritura y son

regrabables. Así como las EEPROM, las memorias flash tienen altas velocidades de acceso y escritura y son no volátiles, es decir, cuando se desconectan de la energía los datos almacenados en ellas no serán eliminados, de tal manera que la memoria almacena los datos adquiridos por los sensores y además el programa con la aplicación del usuario.

2.3 Aplicaciones

Las aplicaciones de las RIS en donde el consumo de energía es importante, se mencionan a continuación.

- Monitorización del entorno: se caracteriza por un gran número de nodos sincronizados que miden y transmiten periódicamente información en entornos accesibles o inaccesibles. La topología es estable y requiere de datos en tiempo real para análisis inmediatos, ejemplo de ello es la monitorización ambiental y de ésta se desprende la detección de incendios de bosques y el mapeado de la biodiversidad.

- Monitorización de seguridad: Detectan anomalías o ataques en entornos monitorizados por sensores. Dado que no están continuamente enviando datos, consumen poca energía. Aquí lo importante es el estado del nodo, la latencia de la comunicación y el procesamiento de la información en tiempo real. Son usados para el control de edificios inteligentes, a través de la red de vigilancia mediante video, control de tránsito, monitorización de estructuras y en aplicaciones militares.

- Tracking: Controla objetos que están etiquetados con nodos sensores en una región determinada. La topología es muy dinámica debido al movimiento e incorporación de nuevos objetos ya que descubren nuevas rutas de nodos y topologías, por dar un ejemplo: el control de edificios y almacenes.

- Redes Híbridas: Reúnen aspectos de las tres categorías anteriores. Usan micro-sensores de forma inalámbrica para áreas comerciales o del sector salud.

2.4 Características de las RIS.

Las principales características de una RIS se describen en los siguientes párrafos.

1. Topología

En una red de sensores la topología de red puede ser cambiante debido a la aplicación para cual está designada. Sin embargo, los nodos tienen que poder comunicar los nuevos datos adquiridos ya sea para redes móviles o sin movimiento. Se habla entonces de una topología dinámica o estática.

2. Difusión multidireccional (broadcast) o unidireccional (unicast)

Las RIS trabajan mediante protocolos que permiten la difusión de la información de la red broadcast o unicast. En la difusión *broadcast* los paquetes de datos son enviados a todos los nodos que se encuentren en su cobertura de red de forma simultánea y en la *unicast*, los datos son enviados a un nodo en específico.

3. Variabilidad de canal

El canal de radio es un canal muy variable, por esta razón la red debe de ser capaz de minimizar una serie de efectos debido a éste entre los que se encuentran fenómenos como la atenuación, desvanecimientos rápidos, desvanecimientos lentos e interferencias que pueden producir errores en los datos.

4. Tiempo real

Generalmente los datos llegan con cierto retraso cuando se transmiten a un nodo, sin embargo, algunos de éstos deben entregarse dentro de un intervalo de tiempo predeterminado ya que pasado éste dejan de ser válidos para una red que necesite reacción inmediata del sistema. En caso de que una aplicación tenga estas restricciones se deben tomar las medidas que garanticen la llegada a tiempo de los datos.

5. Consumo energético

Es uno de los factores más sensibles debido a que tienen que conjugar autonomía con capacidad de procesamiento ya que cuentan con una unidad de energía limitada. Un nodo sensor tiene que contar con un procesador de consumo ultra bajo, así como de un transceptor con la misma característica [2]. A esto hay que agregar protocolos de acceso al medio y de enrutamiento que también compartan esta característica haciendo el consumo aún más restrictivo.

6. Seguridad

Las comunicaciones inalámbricas viajan por un medio vulnerable y accesible a personas y dispositivos ajenos a la red de sensores. Esto implica un riesgo potencial para los datos recolectados y para el funcionamiento de la misma red. Por lo mismo se deben establecer mecanismos que protejan la información de intrusos y cubrir propiamente la vulnerabilidad de los datos.

7. Costes de producción

Dado que una red de sensores generalmente trabaja con un número elevado de motes para poder obtener datos con fiabilidad en un espacio determinado es deseable que éstos sean económicos.

2.5 Topologías

La topología en una RIS se clasifica desde la perspectiva externa e interna. La externa incluye topologías de red de un conjunto de redes inalámbricas y la interna incluye topologías de red de un conjunto de nodos, como se explicará a continuación.

2.5.1Topología externa

Debido a la infraestructura de red, las RIS se dividen en dos tipos de topologías: la centralizada y la distribuida.

a) Topología centralizada

Los nodos de las RIS que monitorean un fenómeno físico, como se muestra en la figura 2.3, envían sus datos directamente al *gateway* más cercano, uno por cada RIS, éste a su vez dirige el tráfico de las redes participantes a través de la infraestructura de red a la estación base y posteriormente a un servidor central.

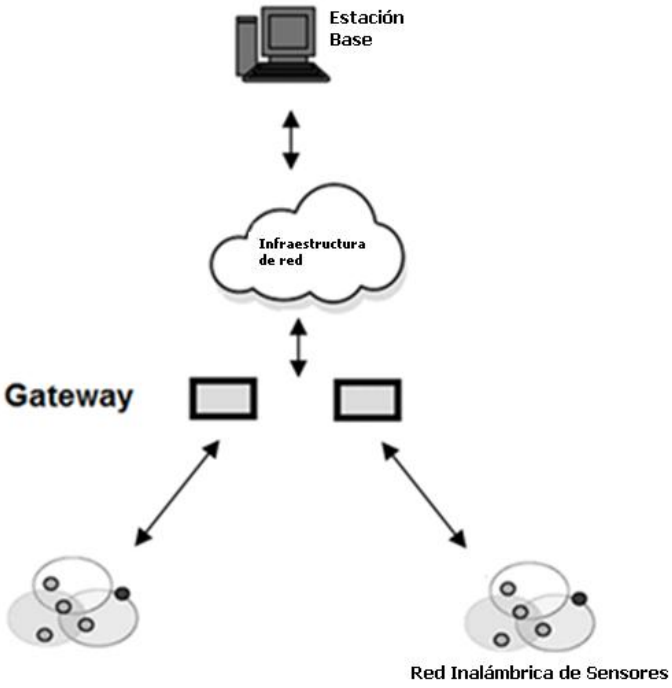


Figura 2.3 Arquitectura centralizada en RIS.

b) Topología distribuida

Utilizan una arquitectura de red de tipo jerárquica y un proceso conocido como *clustering*, en éste los nodos de la red se organizan en grupos llamados *clusters* uno por cada RIS, que es la forma en que los nodos en una MANET (por sus siglas MANET, acrónimo de *Mobile ad hoc networks,* 'red móvil ad-hoc') se organizan. Además como se muestra en la figura 2.4, los nodos se comunican entre sus vecinos y cooperan entre ellos, ejecutan algoritmos para obtener como resultado la respuesta global del sistema, [5]. Después, para enviar los datos recolectados a la estación base utilizan un *gateway* que envía los datos a través de la infraestructura de red a la estación base y posteriormente a un servidor central.

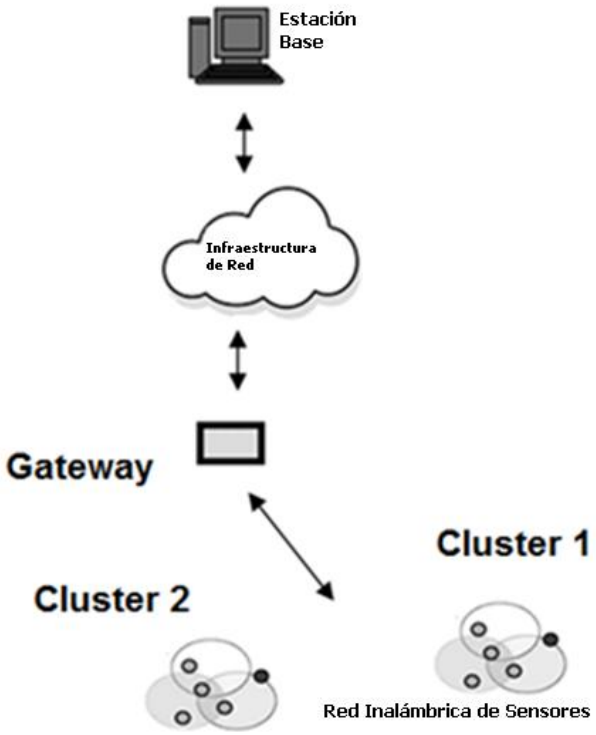
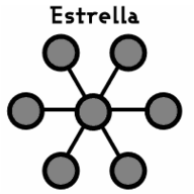
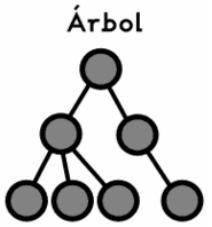
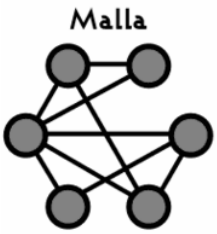


Figura 2.4 Arquitectura distribuida en RIS

La diferencia fundamental entre estas topologías radica en que la centralizada usa por cada RIS un *gateway* mientras que la distribuida sólo usa uno para él envió de datos a la estación base.

2.5.2 Topología Interna

Dentro de la topología interna están incluidas las topologías de red para la interconexión de los nodos de las RIS. Las topologías se explican en la tabla 2.1.

Topología	Diagrama
Estrella: topología más básica. Los nodos se conectan directamente al nodo fuente o al *gateway* a una distancia aproximada de 30 a 100 metros.	
Árbol: topología donde la información se lleva de los nodos sensores más alejados al nodo fuente. Su complejidad es mayor al de tipo estrella por las jerarquías que maneja.	
Malla: arquitectura utilizada comúnmente en las RIS debido a que todos los nodos funcionan como un router, además pueden intercambiar información con el *gateway* o con otros nodos. Tiene tolerancia a fallos, es decir, si un nodo de la red falla, éste tiene la capacidad de buscar otra ruta para comunicarse con nodos vecinos hasta el destino. Reconfigura la red de inmediato y establece tiempos de envíos de información en los nodos.	
Malla con conexión total: existe un enlace directo entre todos los pares de nodos de la red. Una malla completa con *n* nodos requiere de *n(n-1)/2* enlaces directos. Es una topología costosa pero confiable.	

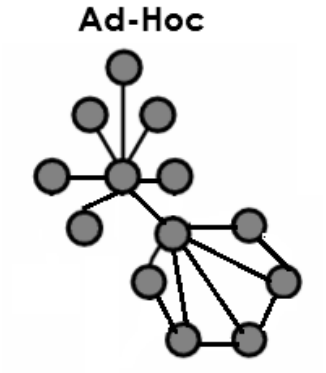
Ad-Hoc: también conocidas como MANET, es una topología de red capaz de interconectarse de forma inalámbrica con otras semejantes, auto-organizables, escalables, seguras y temporales. Redes dinámicas o estáticas según la aplicación. Cada nodo funciona como terminal final de la red.	

Tabla 2.1 Arquitecturas de red en las RIS.

2.6 Estándares

Para que pueda existir comunicación e interoperabilidad entre dispositivos inalámbricos portátiles y que además tengan la capacidad de enviar, transmitir, recibir y procesar información para diferentes aplicaciones de RIS, es necesario hacer uso de estándares de redes tipo HAN (del inglés HAN, acrónimo de *Home Área Network*, 'red de área doméstica') y WPAN, que estén enfocados a redes con enlaces de cortas distancias, un gasto mínimo de energía y bajas tasas de transferencia como alguno de los estándares del grupo de trabajo 802.15, [11]. Este grupo de trabajo se divide a su vez en 5 subgrupos que trabajan en diferentes estándares, que se muestran en la tabla 2.2.

Estándar	Especificación
802.15.1	Estándar basado en la especificación de Bluetooth. Incluye nivel físico y la subcapa MAC (del inglés MAC, acrónimo de Medium Access Control, 'control de acceso al medio') de enlace de datos. Indica limitantes en potencia [10].

802.15.2	Estándar basado en la coexistencia de WPAN con otros dispositivos inalámbricos, que usen bandas de frecuencia no reguladas, tal como las WLAN. Indica limitantes en potencia [8].
802.15.3	Estándar que define altas velocidades de transmisión en capa física y subcapa MAC para WPAN. Indica especificaciones para UltraWideBand [9].
802.15.4	Estándar basado en RIS para tasas de transmisión muy bajas y consumo mínimo de energía, para redes WPAN.
802.15.5	Estándar basado en especificaciones para redes en malla y en redes de tipo WPAN.

Tabla 2.2 Subgrupos del estándar IEEE 802.15.

El estándar 802.15.4 define el nivel físico y la subcapa MAC de la capa enlace de datos. La eficiencia energética reside fundamentalmente en la sincronización de nodos de la red para que puedan permanecer en modo de ahorro de energía el mayor tiempo posible, extendiendo la vida útil de los nodos a través de protocolos de capa 2, [12].Este estándar, es la base de otras tecnologías tales como *ZigBee* o *WirelessHART*, que se encargan de la capa de red y aplicación de las RIS, permitiendo así la personalización, adaptación e integración de aplicaciones.

Otra posible opción para la capa física y la subcapa MAC de enlace de datos de las RIS es el estándar 802.15.1 basado en la especificación técnica de Bluetooth v1.1, este estándar define comunicaciones inalámbricas con mayor ancho de banda que el estándar 802.15.4 lo que posibilita la transmisión de voz y datos en tiempo real. Utiliza transceptores de bajo coste, sin embargo, no es óptimo para las RIS porque está diseñado para trabajar largos periodos de tiempo sin restricciones de energía, [13].

Otro estándar enfocado a la transmisión de información en WLAN, que así como los anteriores define la capa física y la de enlace de datos, es el estándar 802.11. Este estándar define el funcionamiento e interoperabilidad de redes inalámbricas en la banda de los 2.4 GHz. Conocidos también como WiFi, los estándares IEEE 802.11b, IEEE 802.11g y el IEEE 802.11n transmiten a velocidades de 11 Mbps, 54 Mbps y 300 Mbps, respectivamente, además proporcionan tasas de transferencia en voz y envío de datos en tiempo real [9]. Cabe señalar que el gasto de energía no es limitante, ya que los dispositivos bajo este estándar, normalmente son alimentados por la red eléctrica, o requieren de carga de sus baterías cuando agotan su energía y esto puede realizarse en pleno funcionamiento, ya sea en etapas de transmisión o recepción de información de dichos dispositivos. Este tipo de redes inalámbricas son utilizadas básicamente entre computadoras o incluso periféricos que requieren un alto ancho de banda como podría ser una cámara IP.

Es importante señalar que las RIS no requieren una alta tasa de transmisión de datos debido a que es poca la información que envía a los nodos, además si envía una gran cantidad de información su consumo de energía es mayor y por el contrario, si envía pocos datos su consumo de energía disminuye y de esta manera hace eficiente este recurso.

2.7 Protocolos de enrutamiento

Debido a que los nodos no tienen un conocimiento de la topología de red, deben descubrirla y lo hacen a través de protocolos de enrutamiento, [5]. La idea básica es que cuando un nuevo nodo se une a una red anuncie su presencia y sea capaz de escuchar los anuncios *broadcast* y *unicast* de sus vecinos. El nodo se debe informar acerca de los

nuevos nodos a su alcance y de cómo llegar a ellos, al mismo tiempo, avisar al resto de los nodos cuáles nodos pueden ser accedidos a través de él. Después de cierto tiempo, cada nodo sabrá qué nodos se encuentran en su cobertura y las rutas disponibles para alcanzarlos.

Como se muestra en la figura 2.5, existen dos tipos de protocolos de enrutamiento para las RIS: los basados en estructuras de red y los basados en el criterio de enrutamiento, ambos enfocados a prolongar la vida útil de la red al administrar el gasto energético de cada nodo.

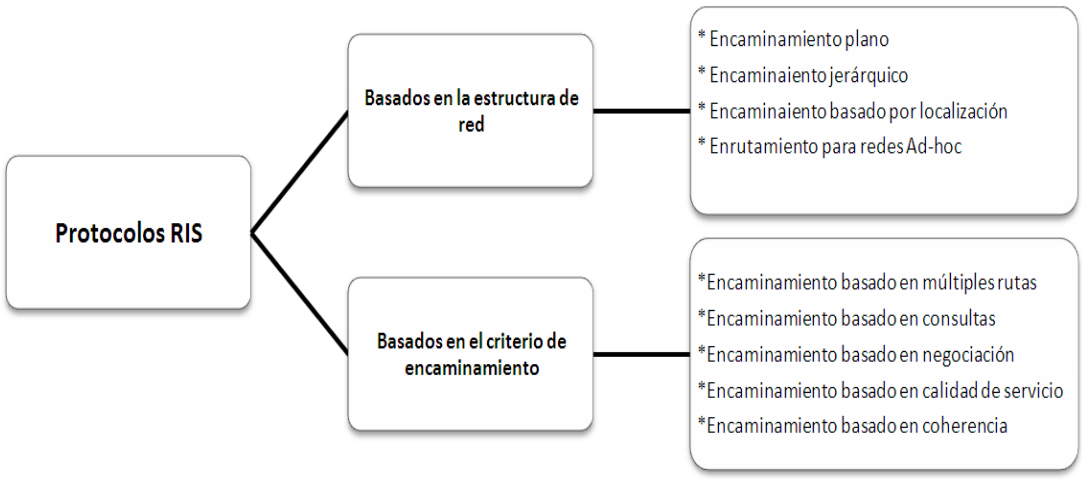


Figura 2.5 Protocolos de enrutamiento.

Tomando en cuenta el criterio de enrutamiento, se consideran protocolos adaptativos si ciertos parámetros pueden ser controlados con el fin de amoldarse a las condiciones actuales de la red y a los niveles de energía disponible, [5].

2.8.1 Protocolos basados en la estructura de red

2.8.1.1 Enrutamiento Plano

El enrutamiento plano es aquel en el que todos los nodos desempeñan el mismo papel y colaboran entre sí para establecer las rutas necesarias de comunicación. Los protocolos que integran esta categoría son SPIN, Directed Difusion, COUGAR, AQUARE, enrutamiento basado en energía y por pasos aleatorios. A continuación se describe cada uno de ellos.

SPIN (*Sensor Protocol for Information via Negotiation*)

Envía datos a los nodos sensores solo si ellos están interesados en recibir dicha información. Envía tres tipos de mensajes, ADV, REQ y DATA. ADV advierte sobre nuevos datos, REQ realiza la petición de datos y DATA contiene la información que procesan los nodos de la red. El protocolo comienza cuando un nodo obtiene nuevos datos vía *broadcast*, después, reenvía un mensaje REQ y DATA. Este proceso se repetirá continuamente hasta que los nodos reciban una copia de la información hasta llegar al nodo destino. Dentro de la familia de protocolos de SPIN, se encuentra SPIN1 y SPIN2, cuyo objetivo principal es hacer eficiente la difusión para actualizar datos en la red mediante el estado de los nodos vecinos. Estos protocolos son usados principalmente en ambientes móviles.

Directed Difusion

Obtiene información de los nodos para reenviarla al nodo fuente durante una inundación. Cada nodo tiene presente su nodo central y éste a su vez por difusión llama y asigna direcciones a cada uno de ellos. La idea principal es combinar los datos que

provienen de diferentes fuentes, eliminando redundancias, minimizando el número de transmisiones y guardando información de la energía de la red.

Existen dos aplicaciones de la difusión directa, la primera es AC *routing* y la segunda DC *routing*. En la primera todos los nodos de la red saben cómo llegar al nodo fuente, con un número de saltos determinado, figura 2.6 a). En este ejemplo, por las tres rutas elegidas inicialmente es por donde la información será direccionada y así todos los nodos entregarán dichos datos. Por otro lado, la DC *Routing*, primero descubre las tres rutas y después elige una de ellas para mandar por esos nodos la información hasta el nodo fuente, figura 2.5 b) y c).

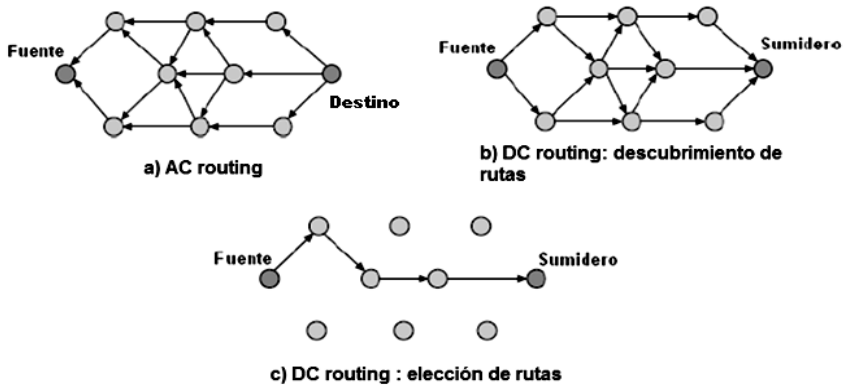


Figura 2.6 Difusión directa.

Existen variantes de la difusión directa, tales como: enrutamiento por rumor, algoritmo de reenvío de coste mínimo, enrutamiento basado en gradiente, enrutamiento de sensores dirigidos por información y enrutamiento de difusión por dirección restringida.

COUGAR

Es un protocolo centrado en datos. Considera la red como un sistema enorme de base de datos distribuida. La idea principal de este protocolo es agregar datos en la red para conseguir más ahorro de energía mediante una capa adicional de consulta que se encuentra entre las capas de red y aplicación. COUGAR incorpora una arquitectura para el sistema de base de datos del sensor donde los nodos sensores seleccionan un líder para realizar la agregación y la transmisión de datos a la estación base. Este último es el encargado de generar un plan de consulta que sirve para especificar la información necesaria sobre un flujo de datos y actualizar la información.

AQUARE (*Active Qwery Forwarding in Sensor Networks*)

Este protocolo ve a la red como una base de datos distribuida, donde las consultas complejas se dividen en varias sub-consultas. El nodo estación base envía una consulta que es retransmitida por cada nodo que la recibe. Después de esto, cada nodo intenta responder a la consulta usando la información obtenida con anterioridad. Los nodos reúnen información de sus vecinos que se encuentran a un cierto número de saltos. Cuando la consulta se resuelve por completo, ésta se envía de vuelta a través del camino más corto a la estación base.

Enrutamiento basado en ahorro de energía

Este tipo de protocolo comienza desde el nodo destino y tiene por objetivo alargar el tiempo de vida de la red. Mantiene un conjunto de rutas en vez de forzar una ruta óptima. Estos caminos son elegidos a través de ciertas probabilidades cuyo cálculo depende de cómo se logra el consumo mínimo de energía en cada ruta. Debido a que los

caminos se eligen de manera aleatoria, la energía de una sola ruta no se agotará rápidamente y en consecuencia se obtiene un mayor tiempo de vida para la red. El protocolo inicia una inundación para descubrir todas las rutas y los costes entre el nodo fuente y el destino. De esta forma se crean las tablas de rutas.

Enrutamiento con pasos aleatorios

Este protocolo trata de obtener un equilibrio de carga usando el enrutamiento multi-trayectoria. Esto se realiza a través de una técnica a gran escala donde los nodos tienen movilidad muy limitada. Aquí los nodos se pueden activar y desactivar de forma aleatoria para encontrar una ruta del origen al destino. La información de su ubicación se calcula a través de la distancia entre los nodos. Un nodo intermedio se selecciona como el siguiente salto al nodo vecino si se encuentra cerca del nodo destino.

2.8.1.2 Enrutamiento Jerárquico

Están enfocados a la escalabilidad y a la comunicación eficiente. Es una arquitectura jerárquica porque los nodos de mayor energía son usados para procesar información mientras que los nodos de baja energía se usan para detectar la proximidad del nodo destino. Los nodos con más energía son asignados como jefes de grupos y contribuyen a la escalabilidad del sistema global, al tiempo de vida y a la eficiencia energética. Este tipo de enrutamiento reduce el consumo energético de forma eficaz con el fin de disminuir el número de mensajes transmitidos a la estación base, es decir, rechaza mensajes duplicados en la red.

Algunos protocolos de este tipo de enrutamiento son: LEACH, PEGASIS, TEEN, APTEEN y MECN. A continuación se describe cada uno de ellos.

LEACH (*Low Energy Adaptive Clustering Hierarchy*)

Es un protocolo enfocado en la arquitectura de conjunto, su objetivo es minimizar la energía distribuyendo el consumo entre los nodos de las RIS. El protocolo se encarga de seleccionar de manera aleatoria a dos nodos de la red de sensores y uno de ellos será el nodo maestro. Este nodo se encargará de distribuir el consumo de la energía en su red de forma equitativa. Este proceso llamado ronda, se generará cada vez que exista una reelección de nodos, lo anterior está condicionado al nivel de energía que posee cada nodo. Si su energía es mayor será el nodo maestro, también llamados jefes de clúster, y por el contrario, serán nodos esclavos aquellos que tengan una energía menor a la del maestro. Este protocolo es usado para la monitorización constante en una RIS.

PEGASIS (*Power Efficient Gathering in Sensor Information System*)

La idea central de este protocolo es que para alargar el tiempo de vida de la red, los nodos se organizan en cadenas en las que cada nodo solamente pueda transmitir su información al siguiente nodo, el que se encuentra detrás o delante de él, es decir en cadena, hasta que el último de ellos lo comunique con la estación base. Así como el LEACH, PEGASIS también funciona a base de rondas, de modo que se seleccionan a los nodos de acuerdo a su nivel de energía para establecer comunicación directa con la estación base. Cada vez que la información es entregada a ésta, comienza una nueva ronda.

TEEN (*Threshold-Sensitive Energy Efficient Sensor Network Protocol*) y APTEEN (*Adaptative Periodic TEEN*)

Estos protocolos se usan para aplicaciones de tiempo crítico. En TEEN todos los sensores están detectando continuamente el medio, sin embargo, la transmisión de datos se realiza con frecuencias mínimas. El protocolo eficiente en energía sensible a umbrales, detecta en cada nodo el umbral de energía alto y el bajo y lo envía a la estación base. De acuerdo a estos valores se condiciona si es posible transmitir datos bajo esos rangos de umbral. La principal desventaja de este protocolo es que si no se detectan los rangos máximos y mínimos nunca existirá comunicación entre los nodos de la red. Otro inconveniente es el gasto de energía doble, el primero es cuando se transmiten los rangos de energía y el segundo momento es cuando se realiza la recolección de datos.

APTEEN es un protocolo híbrido que modifica la periodicidad o los valores usados por TEEN. Los nodos que detectan un nivel de energía por encima del umbral alto transmiten sus datos, en caso contrario no lo hacen. Si un nodo no envía datos durante un tiempo igual a un valor establecido, es obligado a detectar y transmitir los datos. APTEEN utiliza TDMA (*Time Division Multiple Access*, 'acceso múltiple por división de tiempo) para sus transmisiones en tiempos alternados.

MECN (*Minimum Energy Communication Network*)

Protocolo que calcula la eficiencia energética de una subred. La red de comunicación se establece a través de un GPS de baja potencia. MECN identifica una región de retransmisión para cada nodo, ésta consta de nodos en un área donde la

transmisión a través de éstos es más eficiente energéticamente que la transmisión directa. El radio de cobertura de un nodo es creado mediante la unión de todas las regiones de retransmisión que dicho nodo es capaz de alcanzar. El objetivo principal de MECN es encontrar una subred que tenga menos nodos ya que requerirá menos energía para la transmisión entre dos nodos de la misma.

2.8.1.3 Protocolos basados en localización

Son protocolos que tienen por objeto ahorrar energía manteniendo dormidos a los nodos de la red si no existe alguna actividad. Inicialmente los nodos estiman sus niveles de RSSI (del inglés RSSI, acrónimo de *Received Signal Strength Indicator*, 'indicador de intensidad de la señal recibida), después identifican las coordenadas donde se encuentran localizados. Finalmente, los nodos intercambian estas dos informaciones a sus nodos vecinos. Esta comunicación es a través de receptores GPS de baja potencia y mediante un satélite. A continuación se describen dos de este tipo de protocolos.

GEAR (*Geographic Energy Aware Routing*)

Protocolo que examina mediante atributos geográficos y de ahorro de energía la transmisión de información entre nodos vecinos hasta encaminar los mensajes a la dirección del nodo destino. La idea clave es limitar a los nodos de la red para que solamente algunos por localización geográfica puedan interactuar entre ellos y hacer un gasto de energía mínimo. Los nodos seleccionados determinan qué región usarán y los nodos no seleccionados enviarán su información a aquellos para transmitirla hacia el nodo destino. Este protocolo hace una inundación restringida estimando la distancia

entre ellos, el gasto de energía por saltos a los nodos vecinos y seleccionando la mejor ruta para conservar la energía hasta llegar al destino.

SPEED

Protocolo de enrutamiento basado en calidad de servicio, proporciona garantía de entrega desde el nodo fuente hasta el nodo destino en tiempo real. Exige que cada uno de los nodos mantenga actualizada la información acerca de sus vecinos y utiliza el reenvió geográfico para encontrar las mejores rutas. Asegura también una velocidad de transmisión de paquetes, de allí su nombre, con el fin de que los mismos nodos tengan en cuenta los tiempos de envío o si existen retardos en los paquetes, [6].

2.8.1.4 Protocolos de enrutamiento para redes Ad-Hoc

Estos protocolos se dividen en tres tipos: protocolos proactivos, protocolos reactivos y protocolos híbridos.

a) Características de los protocolos proactivos

- Todas las rutas son calculadas antes de que se necesiten.
- Periódicamente envían información de enrutamiento.
- La sobrecarga de información es alta.
- Ejemplos: WRP (*Wireless Routing Protocol*) y DSDV (*Destination Sequence Distance Vector*).

b) Características de los protocolos reactivos

- Las rutas se descubren y se establecen bajo peticiones.
- Se toma en cuenta el gasto de energía que se consumirá en la red.

- Para cada comunicación entre el nodo fuente y el nodo destino se descubre la mejor ruta.
- Se evita la retransmisión de paquetes innecesarios.
- Se sufren pérdidas de tiempo al descubrir una ruta.
- Ejemplos: DSR, EARWSN y AODV.

c) Características de los protocolos híbridos

- Combinación de los otros dos.
- Ejemplos: ZRP (*Zone Routing Protocol*).

Para este proyecto es esencial que el gasto de energía sea mínimo en el envío de datos, que el descubrimiento de rutas se realice mediante peticiones y que la comunicación comience desde un nodo fuente y llegue hasta un nodo destino y se emita una respuesta. Por ello, los protocolos en los que se basó este trabajo pertenecen al tipo de protocolos reactivos. En los siguientes apartados se explican los protocolos EARWSN y AODV que pertenecen a dicha categoría. El protocolo EARWSN se eligió debido a la relativa sencillez para su implementación. Por otro lado, del protocolo AODV sólo se consideró la parte de "Generación de la respuesta de ruta desde el nodo destino", ya que esta parte no viene explicada a fondo en el protocolo EARWSN, por lo que se tomó a AODV como referencia debido a que éste usa un tipo de paquete similar para la respuesta de ruta.

2.9 Protocolo EARWSN (Energy Aware Routing for Wireless Sensor Networks)

Protocolo para redes inalámbricas de sensores destinado a proporcionar un entorno de transmisión fiable con un bajo consumo de energía. Utiliza eficientemente la

RSSI y el nivel de energía de los nodos para identificar la mejor ruta disponible entre el nodo fuente y el nodo destino.

Por otro lado, EARWSN trabaja bajo un mecanismo *event-driven*, es decir, los nodos únicamente gastan energía cuando realizan eventos tales como el descubrimiento de vecinos, las respuestas de ruta y la transmisión de paquetes.

2.9.1 Componentes de red

La red está integrada por un conjunto de nodos, uno de ellos llamado nodo fuente o nodo *Sink*, otro de ellos es el nodo destino y los nodos restantes son llamados nodos intermedios, figura 2.7. El nodo *Sink*, es responsable de la agregación de los nodos en la red y es el nodo que comienza la inundación, el nodo destino y los nodos intermedios permiten la interacción con los demás nodos de la red.

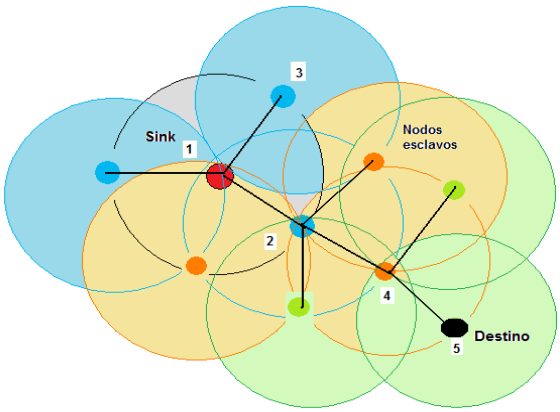


Figura 2.7 Componentes de red en el protocolo EARWSN.

2.9.2 Paquetes

Para que los nodos puedan conocer la topología de la red este protocolo utiliza cuatro paquetes que son: RREQ (Route Request), es enviado cuando el nodo fuente desea conocer una ruta para llegar a un determinado nodo destino, sólo puede ser retransmitido por un nodo a la vez que es elegido por la cantidad de energía que tenga disponible; RREP (Route Replay), es el mensaje de respuesta de ruta que construye el nodo destino cuando es localizado y es retransmitido en unicast por nodos previamente elegidos hasta llegar al nodo fuente; ACKImp (ACK Implícito) y ACKExp (ACK Explícito), estos últimos son utilizados para evitar retransmisiones innecesarias de solicitudes de ruta en la red y así mantener el consumo de energía de la red al mínimo, [7].

Aunque el protocolo EARWSN menciona cuatro tipos de paquetes para su funcionamiento, sólo especifica la estructura del paquete de solicitud de ruta RREQ, llamado RREQMsg en esta implementación. Por ello, en esta sección también se presenta la estructura propuesta de los paquetes AckImpMsg, AckExpMsg y RREPMsg.

2.9.2.1 Paquete de solicitud de ruta (RREQMsg)

Es un paquete de solicitud de ruta, que es enviado para solicitar una ruta hacia un determinado nodo destino. Utiliza un valor numérico como identificador de nodo (Id), figura 2.8. Contiene los campos siguientes:

- IdFuenteRREQ: Id del nodo que ha originado la inundación.
- NoSaltosRREQ: Número de saltos desde el nodo origen hasta el nodo destino.

- NoSeqRREQ: Número de secuencia de la inundación actual.
- ThEnergy: Nivel de energía requerido para retransmitir los paquetes.
- ThPower: Indicador de la intensidad de la señal recibida (RSSI).
- IdDestinoRREQ: Id del nodo con que se quiere establecer comunicación.
- IdNodoInter: Id del nodo intermedio que reenvía un paquete RREQ.

IdFuenteRREQ	NoSaltosRREQ	NoSeqRREQ	ThEnergy	ThPower	IdDEstinoRREQ	IdNodoInterRREQ

Figura 2.8 Paquete RREQMsg.

2.9.2.2 Paquete de confirmación implícita (ACKImpMsg)

Es un paquete de confirmación de reenvío, confirma el reenvío de una solicitud de ruta de forma directa para evitar que otros nodos reenvíen dicha solicitud, figura 2.9.

Contiene los siguientes campos:

- IdFuenteACKImp: Id del nodo que origina el RREQMsg.
- NoSeqACKImp: Número de secuencia actual de la inundación.
- IdNodoOriginadorImp: Id de nodo que origina el ACKImpMsg.

IdFuenteACKImp	NoSeqACKImp	IdNodoOriginadorImp

Figura 2.9 Paquete ACKImpMsg.

2.9.2.3 Paquete de confirmación explicita (ACKExpMsg).

Es un paquete de confirmación de reenvío, confirma el reenvío de una solicitud de ruta de forma indirecta para evitar que otros nodos retransmitan dicha solicitud, figura 2.10.

Contiene los siguientes campos:

- IdFuenteACKExp: Id del nodo que origina la inundación.
- NoSeqACKExp: Número de secuencia actual del nodo fuente.
- IdNodoOriginadorExp: Id del nodo que origina el ACKImpMsg.
- CampoAux: Campo que permite diferenciar entre los paquetes ACKImpMsg y ACKExpMsg.

IdFuenteACKExpl	NoSeqACKExpl	IdNodoOriginadorExpl	CampoAux

Figura 2.10 Paquete ACKExpMsg.

2.9.3 Tabla de rutas

Los paquetes mencionados anteriormente contienen la información necesaria para generar las tablas de rutas de los nodos. Las tablas de rutas guardan información de manera que cada nodo de la red sabe a cuántos saltos se encuentra del nodo fuente, por cuál de los nodos intermedios puede acceder a él y también identifica cuáles nodos son sus vecinos. Así mismo, para evitar las retransmisiones innecesarias de paquetes y mantener las rutas actualizadas, los paquetes cuentan con un parámetro llamado número

de secuencia que permite a las tablas de rutas de los nodos distinguir la inundación más actual mediante un valor entero.

Al término de cada inundación se conocen las rutas disponibles para llegar del nodo fuente al nodo destino. Si existen más de dos rutas posibles se elegirá en primer lugar la ruta cuyo número de nodos cuente con mayor energía para transmitir los paquetes y en segundo lugar, que el camino seleccionado tenga menos saltos al nodo destino. Por otro lado, los nodos que no actúen en el proceso de inundación permanecerán en un estado de reposo, es decir, se encontrarán alertas para actuar en cuanto se les solicite y mientras servirán para conservar energía en la red, dado que los nodos que están participando activamente en la comunicación actual, en algún momento dado tendrán menor energía que estos últimos que se mantuvieron en reposo.

La tabla de rutas, figura 2.11 contienen los siguientes campos:

- IdNodoTabla: Id del nodo con que se quiere establecer comunicación.
- SigSaltoTabla: Id del nodo intermedio para llegar al destino.
- NoSeqTabla: Número de secuencia actual de la inundación.
- NoSaltosTabla: Número de saltos para llegar al nodo destino.
- ThPower: Indicador de la intensidad de la señal recibida (RSSI).

IdNodoTabla	**SigSaltoTabla**	**NoSeqTabla**	**NoSaltosTabla**	**ThPower**

Figura 2.11 Tabla de rutas.

2.10 AODV

Dado que el protocolo EARWSN no especifica la estructura del paquete de confirmación de ruta RREP, la estructura propuesta está basada en el protocolo AODV.

AODV es un protocolo que permite crear de modo dinámico o estático rutas en una red. El protocolo AODV trabaja en arquitecturas de red tipo Ad-Hoc, por lo que garantiza establecer diversas rutas aún cuando alguno de los nodos de la red falle o desaparezca, AODV pertenece a la familia de los protocolos reactivos, debido a que su proceso de descubrimiento de ruta inicia cuando un nodo no sabe cómo llegar a otro y necesita comunicarse con éste.

AODV adopta técnicas de dos protocolos. Del protocolo DSDV, toma el concepto de número de secuencia con el objetivo de implementar un contador en incremento cada vez que en la red se realiza una inundación. Con este valor numérico, es posible identificar la inundación actual con el fin de evitar el procesamiento repetido de paquetes previamente recibidos. Del protocolo DSR adopta el mecanismo de descubrimiento de rutas de modo que el DSR se encarga de calcular la ruta completa desde el nodo fuente hasta el nodo destino. AODV va trazando rutas a través de peticiones a los nodos en modo *broadcast*. El camino se forma con base a la información mantenida en las tablas de rutas de los nodos intermedios.

2.10.1 Información de enrutamiento

Esta información se almacena en forma de tablas de rutas. Cada uno de los nodos que integran la red posee una tabla cuyas entradas equivalen a la cantidad de

nodos con los que se ha comunicado. Cada uno de los nodos se identifica por un valor numérico llamado Id.

2.10.2 Descubrimiento de rutas

El proceso de comunicación en una red Ah-doc comienza cuando un nodo desea comunicarse con otro. El primer paso es buscar en su tabla de rutas si existe una ruta previamente almacenada, en caso de encontrarla no inicia ningún proceso de descubrimiento de ruta, pero por el contrario, si no encuentra alguna ruta comienza con el proceso de descubrimiento. Éste comienza cuando un nodo hace una solicitud de ruta mediante un paquete RREQ y termina cuando dicho nodo recibe una respuesta de ruta mediante un paquete RREP. Cabe destacar que la difusión de paquetes RREQ es en modo broadcast y la de RREP es de modo unicast.

2.10.3 Formación del camino de ida

El paquete de solicitud de ruta RREQ se difunde por inundación simple, es decir, el nodo fuente envía la solicitud de ruta a los nodos vecinos y ellos los retransmiten hasta llegar al nodo destino. Cada vez que un nodo recibe un paquete RREQ comprueba si es el nodo buscado o si al menos sabe cómo acceder a él. Cada uno de los nodos almacena la información recolectada y de esta forma evita el procesamiento repetido del paquete. Es posible que se encuentre más de una ruta, sin embargo, la selección de alguna de ellas depende de la cantidad del número de saltos sea él menor. El número de saltos es la cantidad de nodos que debe de pasar para llegar desde el nodo fuente al nodo destino.

Como se observa, la formación del camino de ida no hace consideraciones para el ahorro de energía.

2.10.4 Formación del camino de vuelta

Cuando un nodo intermedio sabe cómo comunicarse con el nodo destino, este último envía una respuesta de ruta, es decir, transmite un paquete RREP de vuelta al nodo fuente de forma *unicast*. Un paquete RREP sigue el camino trazado por el paquete RREQ, ya no realiza ningún proceso de descubrimiento de ruta. Por otro lado, cada vez que un nodo recibe un RREP, actualiza su tabla de rutas, es decir, refresca el número de secuencia cuando es menor que el actual. De esta manera la recepción de paquetes también es utilizada para actualizar las tablas de enrutamiento, [8].

Para este trabajo solo se tomará la información de la formación del camino de vuelta, ya que formación del camino de ida se resuelve con el protocolo EARWSN.

2.10.5 Paquete de respuesta de ruta RREP

Este paquete se envía como contestación de ruta a un paquete de solicitud de ruta RREQ, confirmando un establecimiento de ruta. La estructura del paquete RREP de AODV se muestra en la figura 2.12. Sin embrago, para los propósitos de este trabajo se adaptaron los campos como se muestran en la figura 2.13.

Prefijo de tamaño	Número de saltos	Dirección IP de destino	Número de secuencia	Dirección IP de originador	Tiempo de vida

Figura 2.12 Paquete RREP del protocolo AODV.

IdDestinoRREP	NoSeqRREP	IdFuenteRREP	NoSalRREP	IdNodoAuxRREP

Figura 2.13 Paquete RREPMsg propuesto para el protocolo EARWSN.

- IdDestinoRREP: Id del nodo destino (nodo con el cual se quiere establecer comunicación).
- NoSeqRREP: Número de secuencia actual de la inundación.
- IdFuenteRREP: Id del nodo que ha originado la respuesta de ruta.
- NoSaltosRREP: Número de saltos desde el nodo destino hasta el nodo origen.
- IdNodoAux: Id del nodo intermedio que reenvía el paquete RREQ.

Finalmente, la tabla 2.3, muestra un breve resumen de los protocolos expuestos.

Tipos de protocolos	Protocolos
I. Protocolo basado en la estructura de la red: a) Enrutamiento plano	• SPIN • Difusión directa • COUGAR • AQUARE
b) Enrutamiento Jerárquico	• LEACH • PEGASIS • TEEN • APTEEN • MEC
II. Protocolo basado en localización	• GEAR • SPEED
III. Protocolo de enrutamiento para redes Ad-Hoc	• Proactivos: WRP, DSDV • Reactivos: DSR, **EARWSN y AODV** • Híbridos: ZPR

Tabla 2.3 Protocolos de enrutamiento en RIS.

Capítulo III.

Estándar IEEE 802.15.4

Para la implementación física de este protocolo se recomienda utilizar motes de la familia MICAz que permiten reducir desde las capas 1 y 2 del modelo OSI (en inglés, *Open System Interconnection* 'sistemas de interconexión abiertos') el consumo de energía debido que están basados en el estándar IEEE 802.15.4. El estándar 802.15.4 define un protocolo de comunicaciones con baja tasa de información en la transmisión y bajo consumo de energía para WPAN. El estándar IEEE 802.15.4 define la capa física y la capa de enlace de datos en donde se encuentra la capa MAC. Fue presentado en el 2003 por el IEEE y la actualizaron en el 2006. Se caracteriza por su transferencia de datos confiable, por su operación de corto alcance, el bajo costo en su implementación y una duración de batería razonable.

En este capítulo se describe brevemente este protocolo utilizado ampliamente en la implementación RIS.

3.1 Componentes de una WPAN en 802.15.4

Una WPAN se forma de la conexión de dos o más dispositivos. Existen dos tipos de dispositivos: FFD (en inglés, *Full Function Device* 'dispositivos de función completa') y RFD (en inglés, *Reduced Function Device* 'dispositivos de Función Reducida'). Los FFD funcionan como coordinadores dentro de una red y tienen

comunicación con otros FFD y RDF, mientras que los RDF sólo se pueden comunicar con un FFD.

Dentro de la red existe un FFD que opera como coordinador de la PAN (en inglés, *Personal Area Network* 'red de área personal') el cual se encarga de controlar y administrar la red, [14].

3.2 Topologías

Existen tres tipos de topologías: topología de estrella, topología punto a punto y topología de árbol. Las características de estas topologías se mencionaron en el capítulo II, sin embargo, en este capítulo se menciona cómo se forma cada una en el contexto de 802.15.4.

3.2.1 Formación de una red estrella

Después de que un FFD es activado, éste puede establecer su propia red siendo él mismo el coordinador PAN. Cada dispositivo dentro de la red únicamente tiene comunicación con el coordinador PAN. La comunicación sólo es de dos tipos: de subida que va de los dispositivos hacia el coordinador PAN y de bajada que se dirige del coordinador PAN hacia los dispositivos de la red. En la figura 3.1, se puede observar la estructura de esta topología.

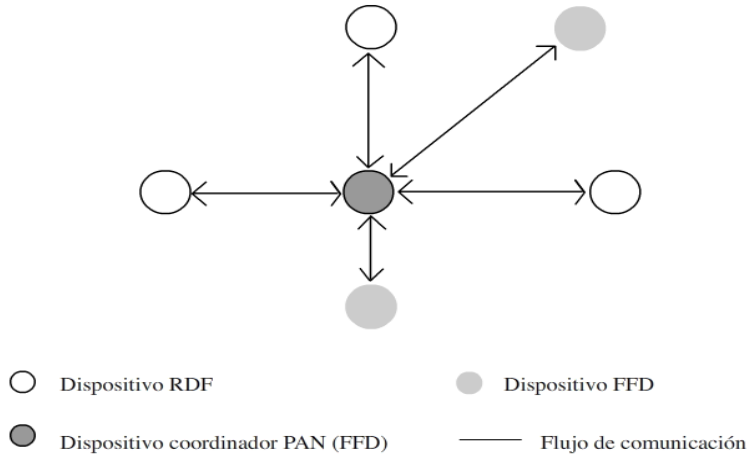


Figura 3.1 Topología de estrella.

3.2.2 Formación de una red punto a punto

Cada dispositivo puede comunicarse con otros dispositivos de la red siempre y cuando estén dentro de su radio de cobertura. Debe de existir un coordinador PAN que puede ser cualquiera de los dispositivos FFD que estén en la red. Pueden existir dentro de la red dispositivos RDF, aunque éstos no pueden retransmitir la información son capaces de capturarla y procesarla. En la figura 3.2 se observa un ejemplo de las conexiones de esta topología.

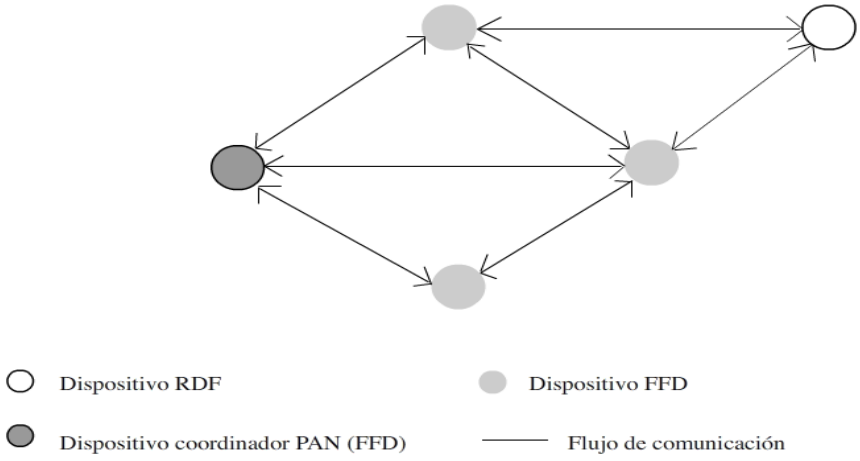


Figura 3.2 Topología punto a punto.

3.2.3 Formación de una red tipo árbol

Esta topología es más compleja y se basa en la topología punto a punto. Está formada sólo por dispositivos FFD. Esta topología se forma por grupos de nodos, designa un grupo de cabecera el cual aloja al coordinador que tiene como nombre DD (dispositivo designado). Los demás grupos dependen del DD, como se muestra la figura 3.3, los otros grupos dependen del grupo de cabecera y cada uno cuenta con un nodo como CH (cabecera de grupo). Cada grupo se identifica con un CID (identificador de grupo) para diferenciarse de los demás grupos, así como también cada nodo cabecera de grupo se identifica como líder de su grupo. Si un grupo desea ser parte de la agrupación, lo hace mediante una petición al nodo DD, si es aceptada se conecta al grupo más cercano pidiendo autorización a su CH. Esta topología se nombra de árbol porque cada grupo conforma una rama haciendo una extensa unión de nodos mediante grupos.

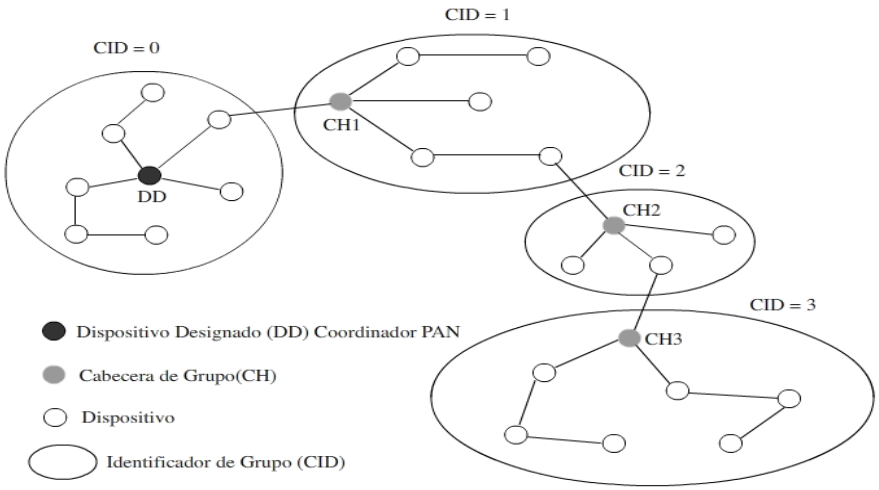


Figura 3.3 Topología de árbol.

3.3 Arquitectura

El estándar IEEE 802.15.4, se apega al modelo OSI, el cual es un conjunto de normas entre sistemas de comunicación que tiene como objetivo contar con los lineamientos para el intercambio de información ya sea en terminales o redes, su arquitectura ésta basada en siete capas donde cada una de ellas consiste en protocolos específicos para la comunicación, [15]. Cada capa es responsable de ciertas funciones al momento de hacer la comunicación con otros dispositivos. La capas tienen como objetivo pasar información de una a otra, esta información son datos o comandos que cumplen con el objetivo de informar a la siguiente capa lo que se requiere que efectúen. Los dispositivos que ocupan este estándar se dividen en las capas que se observan en la figura 3.4. En los siguientes párrafos se explica la capa física y la capa de enlace de datos además de la sub capa MAC (en inglés *Medium Acces Control* 'control de acceso al medio'), [13]. En la capa de red es donde se lleva a cabo la aplicación del protocolo en este trabajo el cual se explicará en el capítulo V.

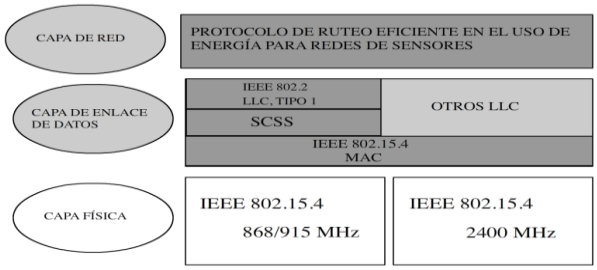


Figura 3.4 Capas del estándar IEEE 802.15.4 del Modelo OSI.

3.3.1 Capa física

Este estándar puede operar en las tres bandas de frecuencia de la banda ISM. Los medios físicos en donde puede trabajar son varios y se definen en la tabla 3.1.

Capa física (MHz)	Banda de frecuencia (MHz)	Extensión de parámetros		Datos de parámetros		
		Tasa de Chips (Kchip/s)	Modulación	Tasa de Bit (Kb/s)	Tasa de símbolo	Método de Dispersión
780	779-787	1000	O-QPSK	250	62.5	16 series Ortogonal
780	779-787	1000	MPSK	250	62.5	16 series Ortogonal
868/915	868-868.6	300	BPSK	20	20	Binario
	902-928	6000	BPSK	40	40	Binario
868/915 (Opcional)	868-868.6	400	ASK	250	12.5	20-bit PSSS
	902-928	1600	ASK	250	50	5-bit PSSS
868/915 (Opcional)	868-868.6	400	O-QPSK	100	25	16 serie Ortogonal
	902-928	1000	O-QPSK	250	62.5	16 serie Ortogonal
950	950-956		GFSK	100	100	Binario
950	950-956	300	BPSK	20	20	Binario
2450 DSSS	2400-2483.5	2000	O-QPSK	250	62.5	16 serie Ortogonal

Tabla 3.1 Características de las capas físicas del estándar IEEE 802.15.4, [13].

A continuación se exponen los medios físicos más usados en cada una de estas bandas, basados en métodos de DSSS (en inglés, *Direc Secuence Spread Spectrum* 'espectro expandido por secuencia directa'):

- 868 MHz. Emplea la modulación BPSK con una tasa de 20 Kbps, utilizando un sólo canal.

- 915 MHz. Emplea la modulación BPSK con una tasa de 40 Kbps. Utiliza 10 canales.

- 2.4 GHz. Emplea una modulación O-QPSK con una tasa de transmisión de 250 Kbps. Utiliza 26 canales.

- Los canales se distribuyen como se muestra en la tabla 3.2 y en la figura 3.5 se observa la división de los canales en el espectro de frecuencia.

Número de canales	**Canal de Frecuencia (MHz)**
n=0	868.3
n=1,2,3,……,10	906+2(n-1)
n=11,12,……..,26	2405+5(n-11)

Tabla 3.2 Canales del estándar IEEE 802.15.4.

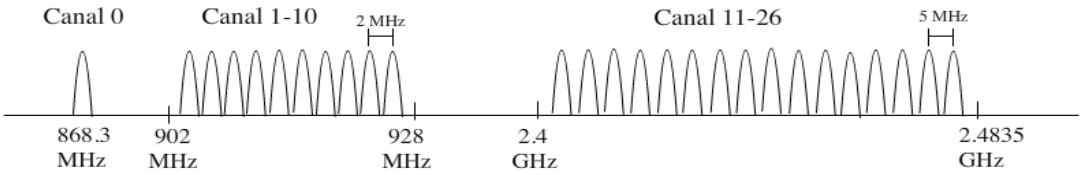


Figura 3.5 Estructura de los canales del estándar IEEE 802.15.4, [16].

Debido a las múltiples redes inalámbricas que trabajan en las mismas bandas de frecuencia y debido al ruido que pueda existir en el medio así como a la interferencia, es de suma importancia el reordenamiento en los canales, por lo que el estándar está diseñado para implementar una selección dinámica de éstos. La capa física contiene

funciones que se encargan de detectar los niveles de energía recibidos, contiene indicadores de calidad en el enlace e indicadores para la conmutación de canales, estas funciones permiten la asignación adecuada de canales y una eficaz agilidad al seleccionar la frecuencia que le corresponda a cada canal. Por su lado la capa de enlace de datos tiene funciones que se encargan de buscar a través de los canales una señal guía. Estas funciones son utilizadas por la capa de red para elegir un canal en donde inicie sus operaciones.

En la figura 3.6 se muestra la trama que se utiliza en la capa física. Se puede observar que el preámbulo utiliza 32 bits para sincronizar los símbolos, el delimitador de 8 bits sincroniza la recepción de la trama, la cabecera de 8 bits especifica el tamaño de los datos y hasta 127 bytes para datos o carga útil.

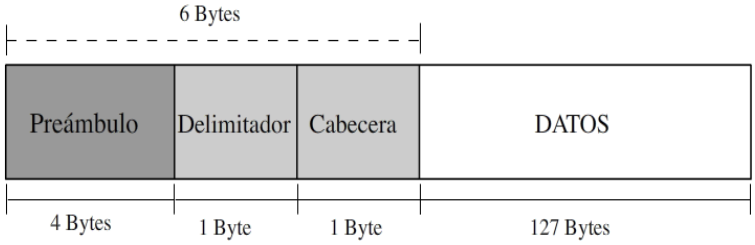


Figura 3.6 Estructura del paquete utilizado en la capa física del estándar IEEE 802.15.4, [16].

La duración máxima en tiempo de cada paquete se especifica en la tabla 3.3.

Banda (MHz)	Duración (ms)
2400	4.25
915	26.6
868	53.2

Tabla 3.3 Duración de los paquetes de la capa física del estándar IEEE 802.15.4.

3.3.2 Capa de enlace de datos

Representa la capa dos del modelo OSI, se divide de acuerdo a la figura 3.7. Conocida como capa de enlace de datos, se divide en dos sub capas. La subcapa LLC (en inglés, *Logical Link Control* ‘control de enlaces lógicos’) y la sub capa MAC. La sub capa LLC es compatible para todos los estándares tipo IEEE 802, es la encargada de llevar el control de las tramas y definir cómo son entregadas a las capas superiores.

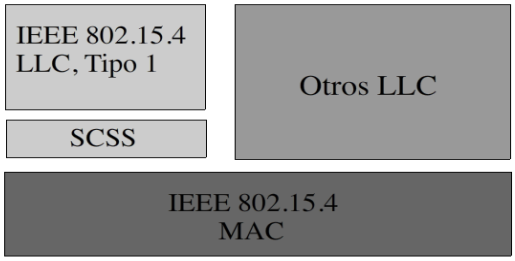


Figura 3.7 División de la capa de enlace de datos en el estándar IEEE 802.15.4.

La capa MAC IEEE 802.15.4 se encarga de acceder al canal de comunicación ya que es un medio compartido con otros dispositivos, también se encarga del manejo de las ranuras de tiempo, de la validación de tramas y del manejo de guías. Para la comunicación entre la subcapa de convergencia de servicios específicos (SSCS), otro LLC y la capa física se requiere de la subcapa MAC que proporciona dos servicios a capas superiores. El primero es el servicio de datos MAC que se accede por medio de la

parte común de la sub capa (MCPS-SAP), el segundo es el manejo de servicios MAC que se accede por medio de la capa de manejo de identidades (MLME-SAP).

La trama MAC o unidad de datos del protocolo MAC (MPDU), figura 3.8, se compone de la siguiente manera: encabezado MAC (MHR), unidad de servicios de datos MAC (MSDU) y pie de MAC (MFR).

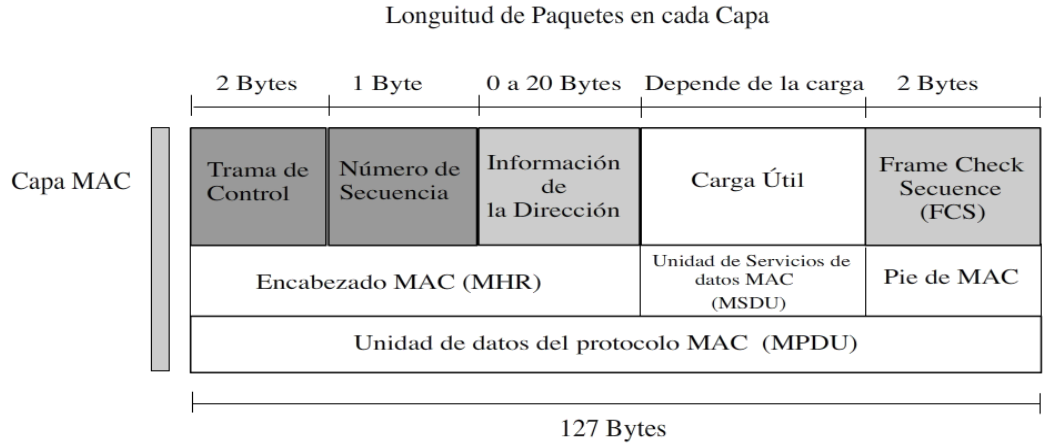


Figura 3.8 Estructura del paquete utilizado en la capa MAC del estándar IEEE 802.15.4, [13].

Dentro del encabezado existe el campo de control, el cual especifica el tipo de la trama. El tipo de trama puede ser: la trama de datos o la trama guía que contienen información proveniente de las capas superiores, la trama de confirmación o la trama de comandos MAC que solo son utilizadas por la capa MAC. La longitud de dichas tramas depende de la carga útil y no excede los 127 bytes de información. Existen otros campos dentro de las tramas como el número de secuencia y las tramas de chequeo (FCS), el primero verifica que el paquete entregado haya sido correspondiente a la secuencia del que se envío y el segundo se encarga de corroborar la integridad de las tramas.

Para lograr la transmisión sobre un mismo canal de frecuencia, los dispositivos utilizan CSMA/CA (en inglés, *Carrier Sense Multiple Acces Whit Collision Avoidance* 'Acceso múltiple por detección de portadora y evitación de colisiones'). Para poder usar el canal, un dispositivo primero debe evaluar el canal para saber si está despejado y que no está siendo ocupado por otro dispositivo, después comienza a transmitir estas acciones las logra mediante la medición de la energía espectral en el canal deseado, esta acción se conoce como ED (en inglés *Energy Detection* 'detección de energía').

3.4 Funcionamiento

Existen dos modos de transmisión: modo ranurado y modo no ranurado.

3.4.1 Modo ranurado

El coordinador es el encargado de transmitir una trama *beacon* cada cierto periodo de tiempo. Esta trama sirve para la señalización y contiene las especificaciones de la supertrama y su tiempo de duración. De esta forma organiza a los demás dispositivos que se encuentran en la red además de administrar el acceso al canal y la transmisión de datos que vienen especificados en un *superframe* o supertrama.

La supertrama se representa en la figura 3.9, consta de dos periodos activo e inactivo. El periodo activo está constituido por 16 ranuras de tiempo, estos tiempos pueden ser configurables, el primer bloque es para la trama *beacon*, diez más para el CAP (en inglés, *Contention Acces Period* 'periodo de acceso por contención') y cinco para el acceso exclusivo por GTS (en inglés *Guaranteed Time Slot* 'ranuras de tiempo garantizadas'). Durante el ciclo completo de la supertrama el coordinador está activo mientras que los dispositivos sólo están activos el tiempo que dura el GTS, durante el

CAP el transceptor puede estar en reposo, siempre y cuando no tenga nada que recibir o transmitir. El periodo inactivo es muy importante para el ahorro de energía. En este periodo todos los nodos de la red RDF y FFD pueden estar apagados o en reposo, estos nodos despertarán antes de que llegue la trama de señalización o *beacon*.

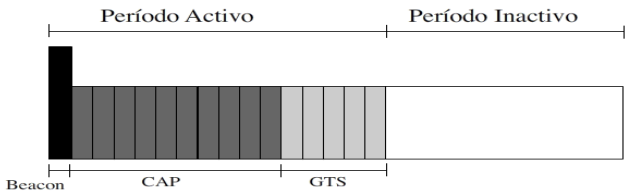


Figura 3.9 Supertrama del estándar IEEE 802.15.4, [3].

Si los dispositivos envían solicitudes durante el CAP el coordinador les asignará ranuras de tiempo. Dentro de la solicitud existe una señal que indica lo que se requiera para saber si el GTS es para recibir o transmitir hacia el coordinador, en otro campo se indican las ranuras de tiempo que se necesitan y por lo tanto deben ser reservadas. Cuando el coordinador responde, primero confirma que recibió la solicitud. Cuando recibe la confirmación del coordinador, el dispositivo espera un determinado tiempo para recibir una trama de *beacon*. Una vez que el coordinador cuenta con los suficientes recursos para otorgar las ranuras solicitadas inserta en el GTS una serie de descripciones al *beacon* siguiente. Las descripciones contienen la dirección del nodo solicitante, la cantidad y posición de las ranuras necesarias dentro del ciclo GTS. El dispositivo puede hacer uso de estas ranuras cada que lo solicite y que el coordinador lo permita. Estas ranuras son liberadas hasta que el dispositivo lo pida, si el coordinador detecta que no han sido utilizadas después de un tiempo, las cancela por medio de un GTS marcándolo como inválido, liberando las ranuras para otro dispositivo que

requiera utilizarlas, si el dispositivo requiere ocupar nuevamente esas ranuras de tiempo requiere volver a iniciar el proceso de transmisión.

La transmisión de datos de un nodo RDF a un nodo FFD puede ser de dos formas diferentes: mediante ranuras GTS o mediante el periodo CAP, esto sólo si se está utilizando CSMA/CA ranurado. Cuando una transmisión FFD a RDF ocurre se requiere de ranuras GTS. En el caso de que no se hayan reservado estas ranuras se sigue una serie de pasos; primero el FFD da la alerta que tiene datos para un RDF mediante la trama *beacon* donde se incluye la dirección del dispositivo que va establecer la comunicación, si al RDF que fue solicitado le llega el mensaje, hace una petición de datos mediante el ciclo CAP, por medio de un mensaje de confirmación el FFD responde y hace el envío de datos, una vez que los datos llegan al RDF envía un mensaje de confirmación donde avisa que recibió los datos. En caso de que el FFD no haya recibido el acuse, hace una serie de repeticiones y espera un tiempo hasta el siguiente *beacon*, en este proceso el dispositivo puede apagar su transceptor hasta que llegue otro *beacon*.

3.4.2 Modo no ranurado

La importancia de este modo es que depende de la configuración de la aplicación, no hay periodo GTS, el FFD no transmite trama de *beacon*, ahora la transmisión de los RDF se hace utilizando CSMA/CA no ranurado. Ejecuta la operación CCA (en inglés, *Clear Channel Assessment* ‘valoración del canal libre’) antes de transmitir esta que permite evaluar si el canal está libre para hacer la transmisión. Los RDF se pueden encontrar dormidos y sólo despertarán cuando necesiten recibir o enviar datos al coordinador FFD. El proceso de transmisión del RDF al FFD es el siguiente: el

RDF envía una solicitud de datos utilizando CSMA-CA no ranurado, el FFD responde con un mensaje de confirmación de recibido. Ahora cuando se necesita transmitir del nodo coordinador FFD al nodo dispositivo RDF se transmite mediante CSMA-CA por lo tanto también responde con un mensaje de confirmación, [16].

Capítulo IV

Entorno de desarrollo

En este capítulo se presentan las características de los motes MICAz en los cuales es posible ejecutar el protocolo desarrollado en este trabajo, se explica también el entorno de desarrollo que se usó para la programación y simulación de este.

4.1 MICAz

Para la implementación física de este proyecto se recomienda utilizar el mote MICAz ya que permite obtener datos de diferentes variables medioambientales, como temperatura, presión, iluminación y gas a través de una RIS de bajo consumo energético, ya que trabaja bajo el estándar 802.15.4. Por otro lado, también se recomienda el mote MICAz porque es posible simularlo en TOSSIM, simulador que se describe más adelante. Por ello, esta tecnología es óptima para el análisis de reducción y optimización en el consumo de energía de una RIS.

A continuación se describen las principales características del mote MICAz.

La motes MICAz son fabricados por la compañía Crossbow, trabajan en la banda de frecuencias de 2400 MHz a 2483.5 MHz de la banda ISM. En la figura 4.1 se muestra un mote MICAz con dimensiones: 5.7 cm de largo, 3.1 cm de ancho y 0.6 cm de alto.

Figura 4.1 Mote MICAZ.

El mote MICAz usa el transceptor de RF del fabricante Chipcon modelo CC2420 y un microcontrolador Atmega 128L. El transceptor de RF cuenta con protección contra interferencias y seguridad de datos. Su tasa de transmisión es de 250 Kbps, suficiente para redes con bajo intercambio de datos. Pueden ejecutarse programas desde la memoria flash interna de 128 Kbytes y almacenarse mediciones en la memoria flas interna de 512 Kbytes [17], además dispone de pines de entrada, de salida y un conector de RF MMCX (del inglés MMCX, acrónimo de *Micro-Miniature CoaXial*, 'conector coaxial micro-miniatura') para antenas WiFi o GPS, como se muestra en el diagrama de bloques de MICAz de la figura 4.2.

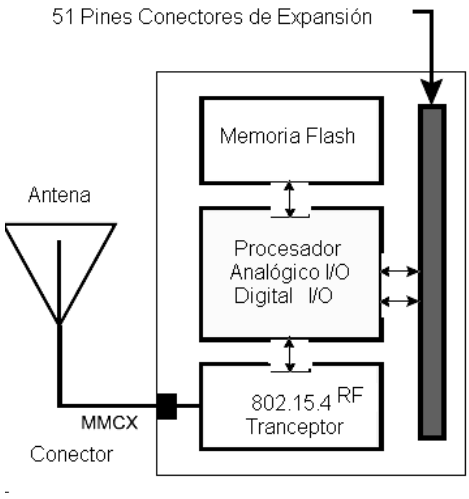


Figura 4.2 Diagrama de bloques.

Para la programación de los motes se emplea la placa de desarrollo MIB510. Esta placa actúa como interfaz entre el PC y los motes a través del puerto serie, permitiendo la programación de los dispositivos, figura 4.3.

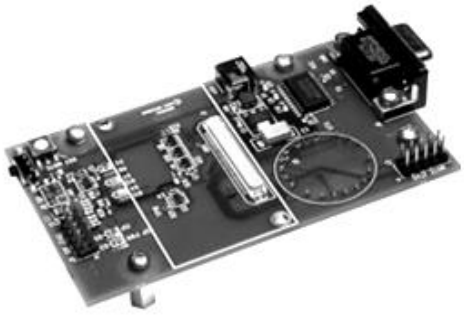


Figura 4.3 Placa de desarrollo MIB5105

El código se descarga al ISP (del inglés ISP, acrónimo de *In System Programmer*, 'procesador integrado en la placa'), a través del puerto serie RS-232, y es el ISP el que programa el código en el mote. Esta placa tiene conectores para motes MICAz, MICA2 y MICA2DOT.

El dispositivo MICAz ha sido diseñado para trabajar bajo la alimentación de baterías de tipo AA, sin embargo, pueden usarse otras fuentes cuyo rango de alimentación se encuentre entre 2.7 y 3.6 VDC. Las baterías suministradas son alcalinas con una carga de 2000 mA (2 W/V).

Como el objeto de estudio se refiere al consumo de energía, se muestran características de consumo importantes del MICAz en las tablas 4.1

Consumo del procesador en mA	
Modo activo	8
Modo rendimiento completo	12
Modo durmiente	0.01
Modo recepción	19.70
Modo transmisión -10dBm	11
Modo transmisión 5dBm	14
Modo de transmisión 0dBm	17.40
Modo inactivo	0.02
Escribir en memoria flash	15
Sensor de Temperatura	0.40
Sensor de Humedad	1.50
Sensor de Luz	0.03
Sensor de Gas	174.63

Tabla 4.1 Consumo del procesador del mote MICAz

La batería es de 2000 mAh., lo que se refiere a que soporta un consumo de 20 mA durante 100 horas, [18]. Sin embargo, realizando una estimación sobre el consumo para ejecutar procesos básicos del mote con base en la tabla 4.1 se obtiene un consumo total de 64mAh, es decir que se puede activar el nodo aproximadamente 31 veces

durante una hora durante 4 días para ejecuciones de aplicaciones sobre una red. La figura 4.4, muestra el consumo de la tecnología MICAz.

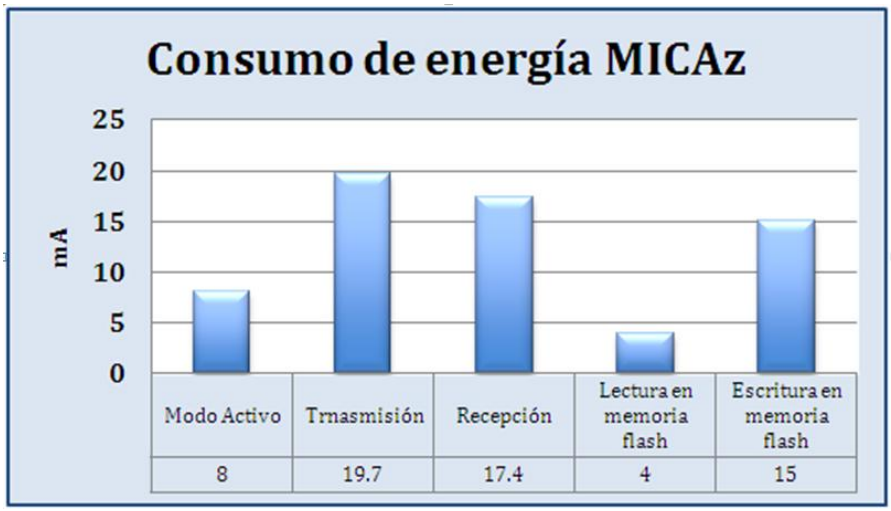


Figura 4.4 Consumo de energía en la tecnología MICAz, [4].

4.2 TinyOs

Debido a que la implementación física del protocolo es deseable se utilizó el sistema operativo TinyOs ya que permite generar programas para las arquitecturas de microcontroladores CISC y RISC, por lo que es posible compilar diferentes plataformas de nodo sensor basados en las familias de microcontroladores MSP430 y Atmega128, como son: MICAz, Micadot, Neomote, Shimmer, Telosb, entre otras.

4.2.1 Características principales del Sistema Operativo TinyOs.

- Está adaptado a protocolos orientados a trabajar con recursos limitados como energía, procesamiento y almacenamiento.
- Su arquitectura está basada en componentes y aplicaciones.

- Tiene capas de abstracción bien establecidas, limitadas claramente a nivel de interfaces, a la vez que se pueden representar los componentes automáticamente a través de diagramas.

- Permite la concurrencia de tareas ejecutadas en primer y segundo plano.

- Dado que está dirigido por eventos (event-driven), reacciona ante la generación de datos de los sensores, mismos que se procesan y se envía en paquetes.

- Enlaza interfaces internas y externas, proceso conocido como wiring.

- Los comandos y eventos son implementados por el lenguaje de programación y también se pueden generar nuevos.

- Los eventos y tareas pueden ser creados por el usuario.

- Pequeño núcleo de *footprint* (huella del ejecutable del SO) de 400 bytes entre código y datos.

- Proporciona un conjunto de interfaces para abstraer la comunicación entre motes.

- Usa operaciones divididas en fases llamada *Split-Phase*, útiles para ejecutar varias operaciones en paralelo separadas en intervalos de tiempo, además de consumir menos memoria.

- Usa aplicaciones que consisten en una configuración de alto nivel y todos los módulos asociados.

- Un paquete en TinyOs tiene una longitud máxima de 255 bytes, esta cantidad en bytes incluye los cuatro tipos de campos que integra el paquete: *Header, Data, Metadata, y Footer* [19].

4.3 NesC

El sistema TinyOs, sus bibliotecas y aplicaciones están escritos en un lenguaje de programación llamado NesC. Es una versión de C que fue diseñada para programar

sistemas embebidos, dicho de otra forma, es un lenguaje de programación de alto nivel para programar la funcionalidad del dispositivo, de tal forma que NesC combina además de componentes una programación orientada a objetos y a eventos.

En NesC, los programas están compuestos por componentes y aplicaciones que se enlazan para formar un programa completo. La diferencia entre un componente y una aplicación es la mínima diferencia de que esta última contiene un componente especial llamado “Main” en el código del programa con extensión .nC. Es decir, es un código ya disponible para ser ejecutado en un sensor y ya no un componente en desarrollo.

NesC maneja dos tipos de componentes. Por una parte, los que ofrece de forma intrínseca el sistema operativo se denominan componentes primitivos y por otro, los componentes complejos que son los componentes proporcionados por terceros, tal es el caso de bibliotecas y/o aplicaciones.

Los componentes y aplicaciones se enlazan a través de sus interfaces. Estas interfaces son bidireccionales y especifican un conjunto de funciones que están implementadas por los proveedores de la misma interfaz. Un componente contiene tres secciones en su código: configuración, implementación y módulo. A continuación se explica cada una de estas secciones, [19].

4.3.1 Configuración

La configuración está integrada por componentes, interfaces y proveedores (provee la información para la ejecución de un código que se compila). Esta sección de código debe incluirse obligatoriamente cuando se crea un componente mediante la

composición de otros ya creados, o incluirlo en la sección módulo. El bloque configuración tiene la siguiente estructura.

Configuration NombredelaAplicaciónAPPC{}
Implementaction{
Components NombredelComponente.
NombredelaAplicacion.Interfaz-> ProveedordelaInterfaz;
}

Un ejemplo de configuración e implementación de una aplicación es la aplicación BlinkC mostrado en la figura 4.5, cuya función básica es encender y apagar periódicamente un led de un mote en tiempos establecidos utilizando un contador.

```
configuration BlinkAppC {
}
implementation {
   components MainC, BlinkC, LedsC;
   components new TimerMilliC() as Timer0;
   BlinkC -> MainC.Boot;
   BlinkC.Timer0 -> Timer0;
  BlinkC.Leds -> LedsC;
}
```

Figura 4.5 Aplicación BlinkC.

Se muestra en la figura 4.5 el bloque de configuración, donde en el apartado de implementación se declaran las componentes MainC, BlinkC, TimerMilliC () y LedsC, además incluye las interfaces Timer () y Leds () y el proveedor de ellas.

4.3.2 Implementación

La implementación define componentes e interfaces, es decir, en esta parte de código se definen las conexiones que hay entre los diferentes componentes que utiliza

la aplicación, generalmente componentes primitivos, con la siguiente estructura de código.

implementation{
components NombredelComponente,
interface NombredelaInterfaz,
NombredelaAplicacion.Interfaz->ProveedordelaInterfaz;
}

En la figura 4.6, se muestran de forma gráfica las conexiones entre componentes, interfaces y proveedores de interfaces que se usaron en el ejemplo anterior, donde se representa que a través de las interfaces Leds, Boot y Timer, se enlazan los componentes LedsC, MainC y Timer0 a la aplicación BlinkC.

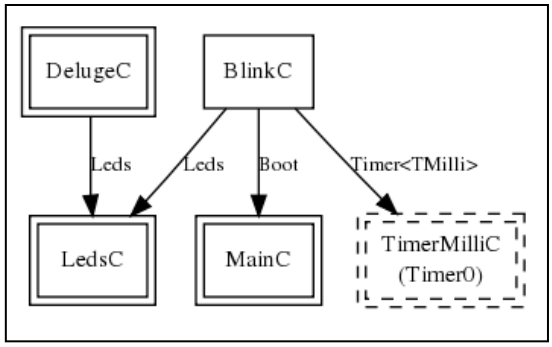


Figura 4.6 Conexión entre componentes e interfaces.

4.3.3 Módulo

El bloque de módulo, es donde se programa el cuerpo general del programa y está dividido en tres secciones de código: proveedores, usos e implementación, con la siguiente estructura.

```
Modulo NombredelaAplicacionM{
provides{…….}
uses{…….}
}
implementation{…….}
```

Provides proporciona al lenguaje de programación las interfaces que va a proporcionar el componente, uses informa de las interfaces que va a utilizar la aplicación o componente, y por último implementation contiene los métodos necesarios para configurar el comportamiento de la aplicación o componente. Se entiende por métodos a las funciones que ha de utilizar la aplicación.

Dentro de la sección *provides*, la forma de anunciar que se va a proporcionar una interfaz es la siguiente:

```
provides{
interface NombredelaInterfazqueproporciono;
}
```

Para indicar que se va a utilizar una interfaz se usan las siguientes líneas de código:

```
uses{
interface NombredelIinterfazqueutilizo;
}
```

Para la implementación se escribe lo siguiente:

```
implementation{
    event void interface.evento(){
    }
}
```

Como se hizo notar, existen dos tipos de implementaciones, la primera en la estructura de un componente y la segunda, la que contiene el módulo. Ésta última tiene por objetivo llamar a eventos, tareas y comandos dentro del cuerpo del programa, y la primera, sólo definir componentes e interfaces que usará la aplicación. Aquí, un ejemplo de la sección módulo de la aplicación BlinkC, figura 4.7.

```
 module Blink {
provides {
  interface StdControl;
}
uses {
  interface Timer;
  interface Leds;
  interface Boot;
}
 implementation {
  event void Boot.booted(){
   call Timer0.startPeriodic (250);
   }
  event void Timer.fired(){
   call Leds.ledToggle();
  }
}
```

Figura 4.6 Sección módulo de la aplicación BlinkC.

4.3.4 Funciones en NesC

Los eventos, tareas y comandos son funciones que usa el lenguaje de programación NesC. Los comandos son funciones vistas como ordenes que ejecuta un programa. La forma de declarar un comando es *call interfaz.comando*, donde se invoca al comando que provee dicha interfaz. Por ejemplo: call Leds.ledToggle(), llama al comando ledToggle() de la interfaz Leds, que hará que se encienda y se apague un led.

Por su parte, los eventos son funciones que se ejecutan cuando el programa detecta un cambio en alguna variable. Existen dos formas de declararlas:

```
evento void interfaz.evento(){
call interfaz.ciomando    ó    signal interfaz.evento
}
```

Un ejemplo de ello, se muestra en la figura 4.7 en la parte de *implementation.*

Por otro lado, las tareas son funciones que se ejecutan concurrentemente. Una de las particularidades es que una tarea es una función que no recibe ni entrega argumentos, simplifica el código de las aplicaciones minimizando el consumo de memoria RAM y puede aplazar la ejecución de un segmento de código mediante el comando *atomic*. Su función es colocar la operación en fila de espera y ejecutarla cuando se hayan completado las que están antes que ésta.

Su declaración es:

```
Task void tarea(){
Atomic{
}
                    }
```

Un ejemplo de tarea, se muestra en la figura 4.8, donde después de declarar la tarea llamada *uartSendTask* de tipo *void*, declara variables, después el *atomic* y finaliza con una estructura de control.

```
task void uartSendTask() {
        uint8_t len;
        am_id_t id;
        am_addr_t addr, src;
        message_t* msg;
        atomic
         if (uartIn == uartOut && !uartFull)
        {
           uartBusy = FALSE;
          return;
        }
```

Figura 4.7 Tarea de la aplicación BaseStationC.

Cabe señalar que el programa se ejecuta de forma asíncrona de acuerdo a los eventos. Por otro lado, si se desea que un componente ejecute varias acciones en paralelo se usan interfaces llamadas Splite-Phase, estas tienen grandes ventajas, ya que consumen menos memoria en su ejecución y reducen el tamaño de la pila del dispositivo.

Después de la compilación se crea un programa ejecutable que contiene todos los elementos mencionados antes, y con base a ellos se generan archivos: componente y módulo con extensión .nc: "*NombredelaApliacion*.nC" y para la aplicación *"NombredelaApliacionAppC.nc"*. Del mismo modo, se debe incluir un archivo de cabecera con extensión "header.h" que contendrá los datos creados por el usuario como son: estructuras de mensajes que maneja la red y/o definición de constantes usadas en el programa. Finalmente el archivo Makefile, es un archivo que aporta información para realizar la compilación de la aplicación.

En resumen, los archivos que son necesarios para la ejecución y compilación de una aplicación en NesC, bajo el sistema operativo TinyOs, son: *NombredelaAplicacion**App.nc,*** *NombredelaAplicacion**.nc,*** *NombredelaAplicacion**.h*** *y Makefile*,[19].

4.4 TOSSIM (TinyOS SIMulator)

Dado que en este trabajo no se contaron con los motes MICAz, se tomó como alternativa comprobar el funcionamiento del protocolo mediante TOSSIM, que es un emulador de hardware y simulador de eventos discretos del sistema operativo TinyOs. Es usado en la etapa de desarrollo para aplicaciones en redes inalámbricas de sensores. Puede simular protocolos de capa de red para motes Mica, MICAz y Telosb que utilizan microcontrolador Atmega 128L. Con este software es posible comprobar el desempeño de la aplicación generada en TinyOs a través de la emulación de componentes como son: topología de red, modelo del canal radio, improvisación del modelo de ruido en RF, asignación de la capa MAC, inserción dinámica de paquetes en los nodos; todos ellos parámetros establecidos por el usuario que pueden ser modificados para verificar el comportamiento de la aplicación. Sin embargo, es necesario que antes de ejecutar la simulación, se asignen manualmente valores a las tablas de enrutamiento de los nodos dado que el simulador no genera dichos datos y necesita de esos valores para ejecutar la simulación.

Por otro lado, TOSSIM emula un procesador de 7.3 MHZ AtmegaL128, una memoria de instrucciones de 128 Kbytes, 152 Kbytes de EEPROM y una memoria de datos de 4 Kbytes, para las simulaciones.

Las aplicaciones simuladas en TOSSIM son ejecutadas por eventos de forma asíncrona, el orden de ejecución se determina de acuerdo a la ocurrencia de los eventos, por lo que se almacenan las solicitudes posteriores. La ejecución en un mote termina cuando finaliza la lista de dichas solicitudes almacenadas. Del mismo modo, TOSSIM ejecuta simulaciones de funciones llamadas tareas, que son porciones de código que se ejecutan más tarde.

4.4.1 Modelo de ejecución

Visto desde el punto de programación, TOSSIM soporta dos tipos de lenguajes, Python y C++. Python permite interactuar con una simulación dinámica, se refiere a que, no es necesario declarar el tipo de dato que va a contener una determinada variable sino que su tipo se determina en tiempo de ejecución donde, éstas a su vez pueden tomar valores de distinto tipo en distintos momentos.

Para la simulación de las aplicaciones se usan dos códigos, uno de ellos programado en NesC y el segundo en Python. En el primero, solo se añaden las etiquetas de acuerdo a la estructura del código, para imprimir la información que están procesando los nodos en el *shell* de Linux, mediante la etiqueta *"dbg"*

dbg("NombredelCanaldeSalida","Letrero \n");

Dbg imprime además de leyendas, parámetros de la aplicación. Para esto es necesario indicar el tipo de variable a imprimir según la tabla 4.2.

Tipo de variable	Especificación
%hhu	Longitud de paquete de 8-bits
%u	Números enteros de 8-bits
%s	Cadena de tiempo

Tabla 4.2 Tipos de variables para la etiqueta dbg

Un ejemplo de ello es:

dbg(“NombredelCanaldeSalida”,”Imprimiendovalor:%u\n”,nombredelaVariable);

Por su parte Python, es un lenguaje de programación que trabaja junto con TinyOS para ejecutar estructuras de paquetes, creación de objetos, simulación de eventos, configuración del escenario de red y todos los factores que intervienen para la transmisión de paquetes entre los nodos de una red, por ejemplo: la topología, el canal de comunicación, el modelo de ruido y la ganancia del sistema, [20], características que se explican más adelante.

4.4.2 Compilación del programa

Una vez programada la aplicación bajo el lenguaje en NesC en TinyOs, se compila en el *shell* de Linux, por medio de la siguiente instrucción: *make micaz.* Enseguida para compilar la simulación, usamos él comando*: make micaz sim* y por último para ejecutar el programa en Python se escribe el comando *python nombredelaaplicacion.py*. Este último programa contiene las configuraciones programadas por el usuario para corroborar el adecuado funcionamiento de la

aplicación. Algunas de las instrucciones de mayor relevancia para la elaboración del código se muestran en la tabla 4.3.

Atributo	Función
From TOSSIM import*	Manda a llamar y/o activar el simulador TOSSIM.
t=TOSSIM([])	Crea objetos en TOSSIM. Un objeto se define como una colección de nodos agrupados en un espacio específico y donde cada uno de ellos es diferenciado por un número entero conocido como identificador de nodo (Id).
t.runNextEventos()	Ejecuta la simulación por eventos en TOSSIM con el código programado en Python.
m=t.getNode(9)	Crea un nodo llamado m con un Id=9.
m.isturnOn/m.isturnOff	Encender/Apagar el nodo m.
import sys	Invoca bibliotecas de Python.
r=t.radio()	Crea en t, un objeto llamado modelo de radio, que será el medio de transmisión de la aplicación.
m=t.getNode(0)	Se extraen datos del nodo 0 y se asignan en m.
m=t.currentNode()	Regresa el Id del nodo actual.
m=t.mac()	Objeto que representa el modelo radio.
m.bootAtTime(45654)	m correrá su primera simulación en el tiempo 45654 milisegundos.

Tabla 4.3 Atributos de Python.

4.4.3 Ventajas y limitantes del simulador TOSSIM

Para finalizar, se mostrarán algunas ventajas y limitantes del simulador TOSSIM que se encontraron durante la fase de pruebas de la simulación del protocolo:

a) Ventajas

- Se puede tomar el código de la aplicación de TinyOs para realizar la simulación, sin necesidad de reescribirlo.
- El usuario es capaz de manipular los parámetros de la topología de red, así como las de ganancia de transmisión para adecuarlo a su enfoque de estudio.
- Simula la pila de protocolos de red del transceptor que usa la familia MICAz, que es la que se pretende usar para hacer la implementación.
- Es confiable, porque captura el comportamiento de la aplicación, es decir, simula todo el funcionamiento del sistema incluidas las interrupciones.

b) Limitantes

- Todos los nodos de la simulación utilizarán los mismos parámetros de simulación, es decir, todos desempeñaran la misma función y solo se verá un único comportamiento.
- Un nodo podrá procesar y transmitir únicamente un sólo paquete a la vez, es decir, no se almacenarán paquetes en el nodo. TOSSIM únicamente recibe un paquete, lo procesa y termina su funcionamiento. Esto es una limitante porque en nuestro sistema de transmisión, los nodos almacenan paquetes según el tiempo de llegada y los procesa de esa misma forma. Debido a esta limitante es imposible conocer el procesamiento continuo y completo del sistema ya que se deberá ejecutar cada paquete por separado.
- El modelo de ruido y las señales de radio son configurables, así que por cada ejecución de la simulación es posible variar estos parámetros tomando en cuenta los valores ideales que propone el simulador para llevarlas a cabo.
- No simula el tiempo de ejecución de un mote en tiempo real, debido a que el simulador compila las funciones del código según se ejecuta el programa, es decir en tiempos muy cortos.

- No es posible conocer el nivel de energía de la batería del nodo, ni el consumo de la misma, así que no se puede monitorear, por lo que representa una limitante para este trabajo ya que no se percibe el nivel de energía de los nodos, así que todos serán aptos para transmitir o viceversa, todo depende del valor que se establezca para la simulación.

4.5 Configuración de la simulación

A continuación se dan a conocer cuáles son los parámetros que se configuran para la simulación de la RIS.

4.5.1 Configuración de la topología de red

Es usada en TOSSIM para establecer comunicación entre nodos de la red dentro del objeto *t* de Python. En dicha comunicación intervienen factores que habrá que configurar para indicar el comportamiento de los componentes de la red, mediante la topología de red que elija el usuario. Para ello, TOSSIM genera un modelo de radio con una señal basada en parámetros que se muestran en la tabla 4.4, para dicha configuración.

Atributo	Función
add(src,dest,gain)	Añade al canal de comunicación del nodo fuente al destino una ganancia. Cuando transmita el nodo fuente, el destino recibirá un paquete atenuado en ganancia.
conected(src,dest)	Indica si existe un enlace entre el nodo fuente y el nodo de destino.
gain(src,dest)	Regresa el valor de la ganancia del nodo fuente al nodo de destino.

Tabla 4.4 Atributos para la topología de red.

4.5.2 Configuración del modelo de ruido

Los valores predeterminados que usa TOSSIM para simular el transceptor inalámbrico, están basados en el transceptor modelo CC2420, que se usa en las familias

MICAz, Telosb e Imote2. Para ejecutar la simulación, el programa necesita de un archivo externo llamado meyer-heavy.txt, que contiene un modelo de ruido para simular el canal radio a simular cuyos valores oscilan entre -98dBm a -38dBm. Para ejecutarlo, se incluye en el código de Python la siguiente línea: *noise=open("meyer-heavy.txt","r")*, donde r indica el modelo ruido y de forma aleatoria selecciona un valor del archivo y lo usa en la simulación, tabla 4.5.

Modelo Ruido (dBm)
-39
-98
-98
-98
-99
-98
-94

Tabla 4.5 Valores del archivo meyer-heavy.txt.

TOSSIM simula el ruido de RF y la interferencia de un nodo de escucha, tanto desde otros nodos así como de fuentes externas a través de un modelo de ruido, este modelo contiene una huella de ruido como entrada y genera un modelo estadístico de la misma, además contiene una serie de lecturas de ruido tomados del archivo meyer-heavy.txt. Este modelo puede capturar ráfagas de interferencia y otros fenómenos correlacionados de tal manera que mejora en gran medida la calidad de la simulación de RF.

4.5.3 Configuración de las ganancias de transmisión

Se puede especificar mediante la línea de código: *1 2 -54.0*, la ganancia (-54.0) de la señal que se transmite del nodo fuente (1) al destino (2). Sin embargo, es útil crear

un archivo que contenga todos los valores de ganancia de los nodos para usarlo en la aplicación. Como ejemplo, el archivo topo.txt que se muestra en la tabla 4.6.

Nodo fuente	Nodo destino	Ganancia (dB)
1	2	-54
2	1	-55
1	3	-60
3	1	-60
2	3	-64
3	2	-65

Tabla 4.6 Archivo topo.txt.

Para incluir este modelo en la simulación, el programa de Python se agrega: *f=open("topo.txt","r")*.

Es importante mencionar que si se le asigna al nodo un valor igual o cercano a -110 dBm, el programa nunca generará ruido.

4.5.4 Configuración de la capa MAC

TOSSIM trabaja con base al protocolo CSMA/CA (Carrier Sense Multiple Acces/Collision avoidance). El CSMA/CA, es utilizado para evitar colisiones entre los nodos que quieran utilizar el mismo medio de forma simultánea. Para ello, se crea un objeto en t, es decir, se crea en un conjunto de nodos con un único identificador un modelo de radio llamado *mac=t.mac.* Este, controla un gran número de funciones para configurar el acceso al medio en la simulación como son: el ancho de banda, el número de símbolos por segundo y los bits por símbolo. Sin embargo, el objeto Mac, por default, establece las siguientes características: 4 bits por símbolo, 64K símbolos por

segundo y 256 kbps [20], valores que fueron tomados en cuenta para la ejecución de la simulación. Los parámetros más relevantes se muestran en la tabla 4.7.

Parámetro	Función
symbolsPerSec	Número de símbolos por segundo que el radio puede transmitir (65536)
bitsPerSymbol	Número de bits por símbolo, que multiplicado este valor por los símbolos por segundo entrega el ancho de banda.
preambleLength	Tamaño del paquete.
ackTime	Tiempo que le toma a un nodo transmitir de forma asíncrona un acknowledgment.

Tabla 4.7 Parámetros de la capa MAC.

4.5.5 Configuración de ejecución de eventos

Es posible determinar el tiempo en que se desea ejecutar la simulación de los eventos del nodo. Un ejemplo de ello sería, ejecutar un evento en 15 segundos del objeto t, correspondería con la siguiente instrucción: *t.runNextEvent(), time=t.time(15000).*

Capítulo V

Programación y simulación del protocolo

En este capítulo se explicó el proceso de programación en NesC mediante el uso de diversos algoritmos así como el desarrollo de la simulación en TOSSIM para verificar la funcionalidad de la programación del protocolo. Para ello, se considera una red estática en un terreno plano, cada nodo tiene comunicación con sus demás nodos vecinos a un sólo salto. Se asume que en la red cada nodo tiene una capacidad limitada de procesamiento, almacenamiento y energía, además cuenta con un número de identificación Id.

La explicación se divide en los siguientes apartados:

- Escenario de prueba.
- Algoritmo considerado para una solicitud de ruta.
- Algoritmo considerado para la confirmación de ruta.
- Algoritmo considerado para una respuesta de ruta.

5.1 Escenario de prueba

Se requiere de un escenario de prueba para transmitir información entre los nodos y poder analizar los algoritmos desarrollados, para ello es necesario conocer el número de nodos que contiene la RIS así como identificar el nodo que funcionará como

nodo de recolección, llamado también nodo *sink*, que recolectará los datos de la RIS y los almacenará en una PC.

El escenario elegido sirve para explicar tanto la simulación como el comportamiento de cada uno de los nodos cuando se genera una petición de ruta, este escenario se muestra en la figura 5.1.

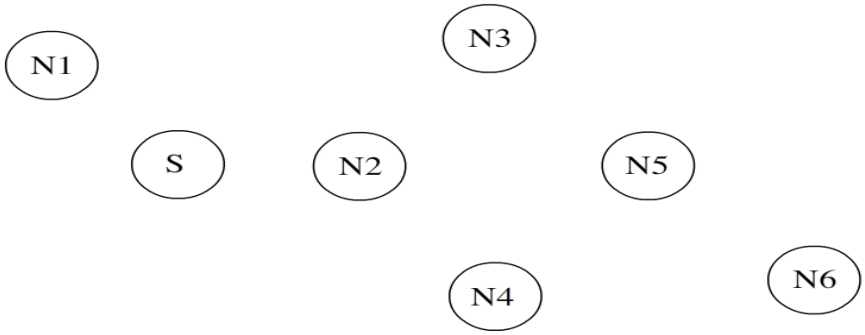


Figura 5.1 Escenario de prueba.

La red está formada por 6 nodos cada uno está etiquetado con una letra N y un número. La numeración se asignará de acuerdo a la proximidad del nodo en cuestión con el nodo fuente (S, *sink*), comenzando la numeración con el más próximo.

El radio de cobertura de cada nodo permite la comunicación con los nodos que se encuentran dentro de ésta, por lo tanto se considera que existe comunicación entre los nodos de acuerdo a la tabla 5.1.

Nodo Emisor	Comunicación con los nodos
S	N1,N2
N1	S
N2	S, N3 y N4
N3	N5 y N2
N4	N2 y N5
N5	N6, N3 y N4
N6	N5

Tabla 5.1 Comunicación entre los nodos.

5.2 Algoritmo para una solicitud de ruta

El protocolo se compone de dos etapas, la primera es inundar por medio de un mensaje en *broadcast* haciendo una solicitud de ruta a la red para conocer la posición de un nodo de destino; la segunda es generar y reenviar la respuesta de dicha solicitud mediante un mensaje de respuesta al nodo fuente en modo *unicast*.

Cada etapa necesita de un paquete para su funcionamiento la primera utiliza el paquete RREQMsg el cual se transmite con el objetivo de descubrir una ruta disponible hacía un nodo destino, este paquete lo genera el nodo fuente. La segunda etapa utiliza el paquete RREPMsg, éste se genera en el nodo de destino y es el encargado de responder a la solicitud. Cuando este paquete llega al nodo fuente se dan por terminados los dos procesos del protocolo implementado, [7].

5.2.1 Etapa de inundación

El objetivo de la etapa de inundación es llenar o actualizar las tablas de enrutamiento de los nodos que en un principio están vacías, comienza con el envío en

broadcast de un paquete RREQMsg desde el nodo fuente hacía sus nodos vecinos haciendo una solicitud de ruta hacia un nodo destino.

En la simulación para enviar un paquete a un nodo se declaran los campos del paquete mediante la interfaz Python, [20]. A continuación, se muestra una parte del código del objeto RREQMsg, uno de los paquetes que se creó para este trabajo.

```
fromRREQMsg import *
msg=RREQMsg()
msg.set_IdFuenteRREQ(1)
msg.set_NoSaltosRREQ(2)
msg.set_NoSeqRREQ(2)
msg.set_ThEnergy(2)
msg.set_ThPower(4)
msg.set_IdDestinoRREQ(2)
msg.set_IdNodoRREQ(3)
pkt=t.new Packet()
pkt set.Data(msg.data)
pkt.setDestination(1)
```

Cuando un nodo recibe un paquete RREQMsg la simulación muestra en pantalla los campos y el contenido del paquete, figura 5.2, el contenido de los campos son valores que pueden ser manipulables a través de Python. Se toma en cuenta que si la simulación arroja los valores del paquete es porque lo ha recibido adecuadamente.

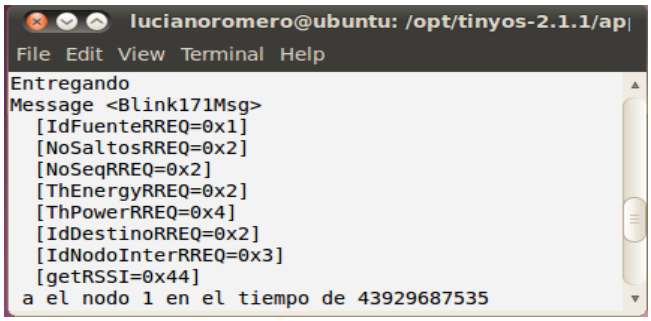


Figura 5.2. Procesamiento de un paquete RREQMsg en la simulación.

Una vez que el paquete es recibido por los nodos que se encuentran en el radio de cobertura del nodo fuente, comienza el proceso de identificación del paquete. Los nodos de la red deben identificar el tipo de paquete que han recibido. La identificación se hace mediante la longitud del paquete recibido dado que los cuatro paquetes tienen diferentes longitudes. Por ejemplo en la figura 5.3, se observa en el simulador el comienzo del procesamiento de un paquete RREQMsg que ha sido recibido, por lo tanto, el simulador lo primero que hace es identificar al paquete y mostrar su longitud.

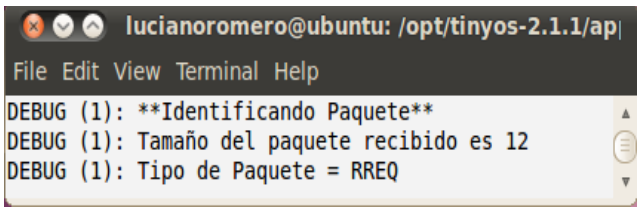


Figura 5.3. Longitud y tipo de paquete en el simulador.

Una vez identificado el tipo de paquete, el nodo elige la función adecuada para su procesamiento se elige el algoritmo correspondiente al tipo de paquete recibido de acuerdo a la figura 5.4.

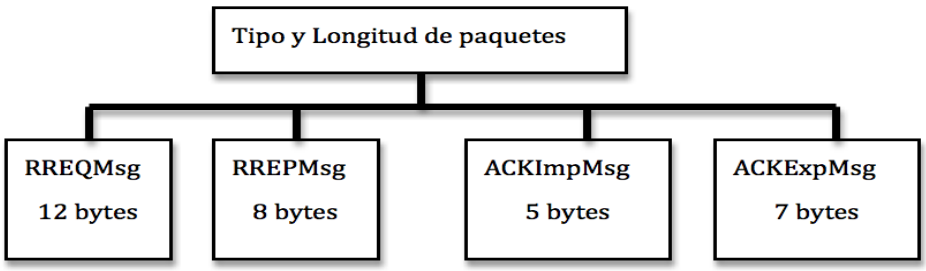


Figura 5.4 Tipo y longitud de paquetes.

El paquete RREQMsg contiene información que es guardada en la memoria del nodo para ser utilizada durante el proceso. Si no es así el nodo guarda los datos

necesarios en su tabla de ruteo para agregar a los siguientes nodos: al nodo vecino que reenvió el paquete y el nodo de recolección que generó la inundación.

Por ejemplo tomando en cuenta el escenario de prueba de la figura 5.1, se hace un envió en *broadcast* del nodo fuente o *sink* (S) hacia los nodos que estén en su radio de cobertura, de acuerdo a la tabla 5.1, N1 y N2 están dentro de su alcance el paquete tiene como nodo de destino a N6. Los nodos guardan en memoria los datos que vienen en el paquete RREQMsg y procesan el paquete de acuerdo al diagrama de la figura 5.5.

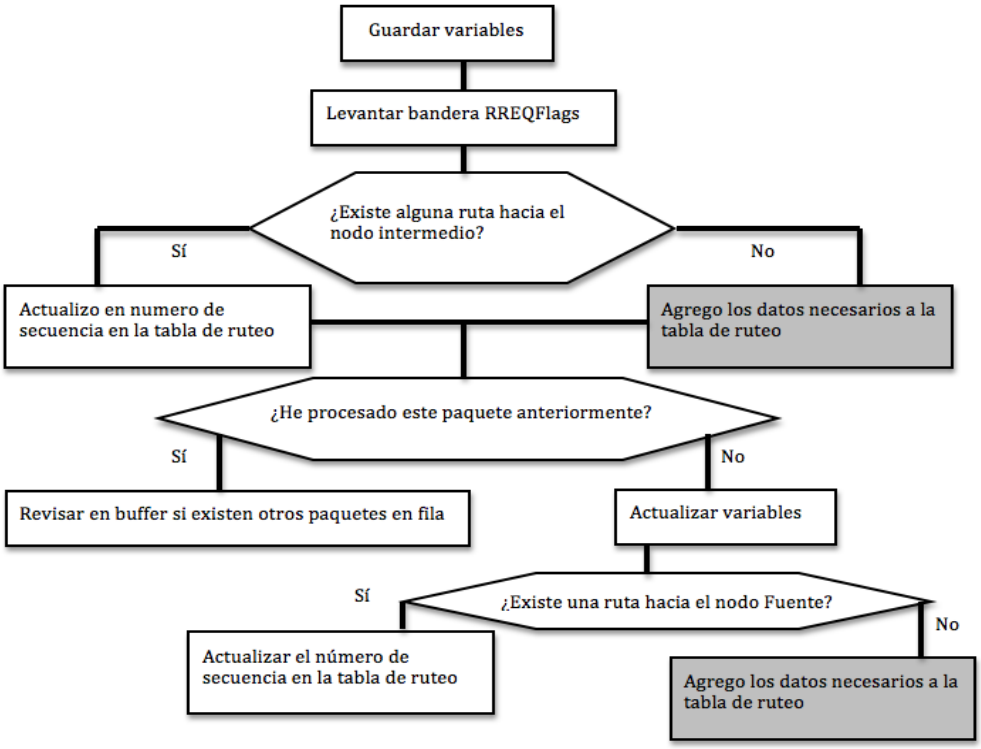


Figura 5.5 Primera parte del algoritmo del proceso de inundación en un nodo.

La tabla de ruteo contiene rutas mediante las que los nodos pueden llegar a sus vecinos y/o al nodo fuente. El protocolo indica dos maneras diferentes para guardar las direcciones, la primera se hace de la forma indicada en el inciso b) de la figura 5.6, en la que el algoritmo guarda la dirección del nodo que envió el paquete y la segunda

indicada en el inciso C) de la figura 5.6, para guardar la dirección del nodo que originó el envío del paquete, en este caso el nodo *sink*.

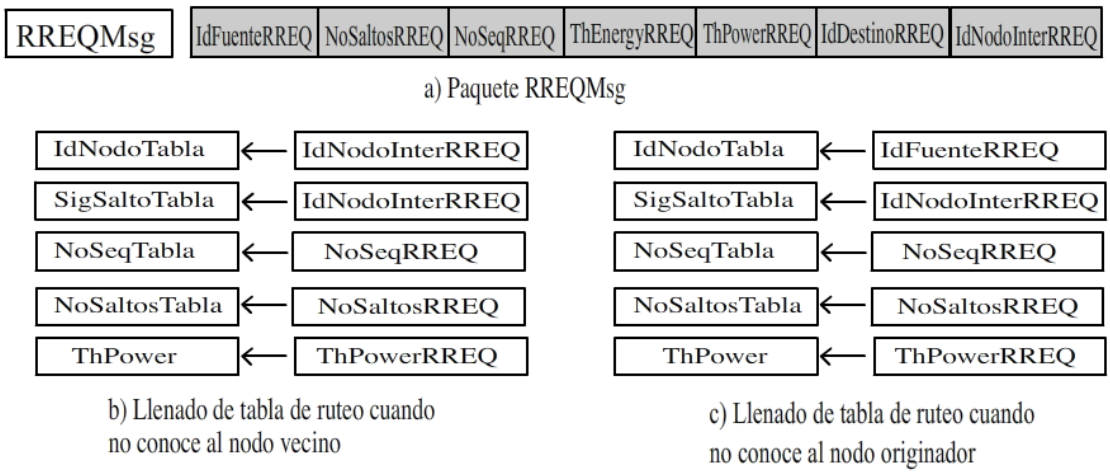


Figura 5.6 Estructura del paquete RREQMsg y llenado de la tabla de ruteo.

De esta manera el nodo N6 después de la inundación conocerá el camino para llegar a su vecino el nodo N5, así como también tendrá en la tabla de ruteo una ruta para llegar al nodo fuente por medio de N5, que se encuentra a cuatro saltos del nodo fuente de acuerdo a la tabla de este nodo, figura 5.7 a). N5 tiene además otros dos vecinos, los nodos N4 y N3, por lo tanto en su tabla de ruteo tiene rutas de cómo llegar a dichos nodos y cómo llegar por medio de ellos al nodo fuente, por ambas rutas se encuentra a 3 saltos, figura 5.7 b).

Los nodos cuentan con la capacidad de hacer actualizaciones de sus rutas en el proceso de inundación o en el proceso de respuesta de ruta. Al recibir un mensaje el nodo verifica si existe alguna ruta en su tabla de ruteo hacía el nodo que envió el mensaje, si existe una ruta en su tabla de ruteo sólo actualiza el número de secuencia, sino existe simplemente la agrega a su tabla.

IdNodoTabla	SigSaltoTabla	NoSeqTabla	NoSaltosTabla	ThPower
N5	N5	1	1	-40
S	N5	1	4	-30

a) Tabla de ruteo del nodo N6

IdNodoTabla	SigSaltoTabla	NoSeqTabla	NoSaltosTabla	ThPower
N4	N4	1	1	-40
N3	N3	1	1	-30
S	N4	1	3	-40
S	N3	1	3	-30

b) Tabla de ruteo del nodo N5

Figura 5.7 Ejemplo de la Tabla de ruteo del nodo 6 después de la inundación.

El algoritmo del proceso de inundación cuenta con la capacidad de verificar si el paquete recibido fue procesado con anterioridad y descartar su procesamiento en caso de que así haya sido, en la simulación se muestra una serie de etiquetas que corroboran el proceso anterior figura 5.8, en este ejemplo no se ha procesado este paquete por lo tanto lo va a procesar de acuerdo al algoritmo que le corresponda.

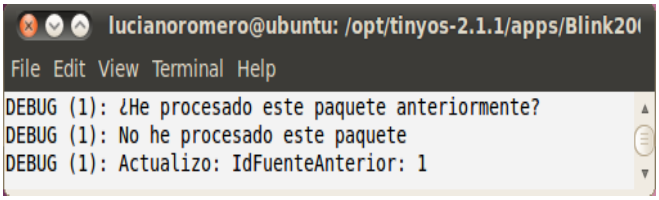


Figura 5.8 Verificación en la simulación, si ha sido procesado el paquete con anterioridad.

Un nodo puede saber si el paquete es actual o no revisando el ID del nodo originador así como el número de secuencia de la inundación actual.

Posteriormente, los nodos que recibieron este paquete deben revisar si son el nodo de destino como viene en el diagrama de la figura 5.10. En caso de que lo sean comenzarán la etapa de respuesta de ruta enviando un paquete RREPMsg. Por otro lado, si un nodo recibe un paquete y no es el nodo de destino, no retransmite el paquete

RREQMsg inmediatamente, antes de enviar el paquete el nodo compara su nivel de energía con el valor indicado en el paquete RREQMsg recibido en su campo ThEnergyRREQ. Si la energía disponible del nodo es menor que la requerida, el nodo descarta el paquete y no hará el reenvío en la simulación se muestra una etiqueta haciendo referencia a la comparación entre las variables ThEnergyRREQ y ThEnergyNODO, por ejemplo en la figura 5.9 el nodo tiene la energía suficiente para seguir transmitiendo el paquete que está procesando. Si cuenta con la energía necesaria entonces deberá entrar a un proceso de conteo que se describe en el siguiente apartado.

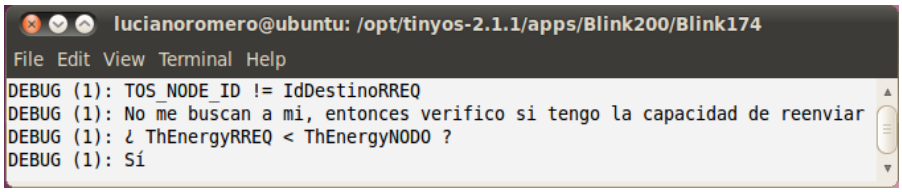


Figura 5.9 El simulador compara la energía registrada del paquete contra la del nodo.

La variable ThEnergy representa la cantidad necesaria de energía que requiere un nodo para hacer el reenvío de los paquetes desde el nodo de destino hasta el nodo fuente una vez establecida la ruta de transmisión.

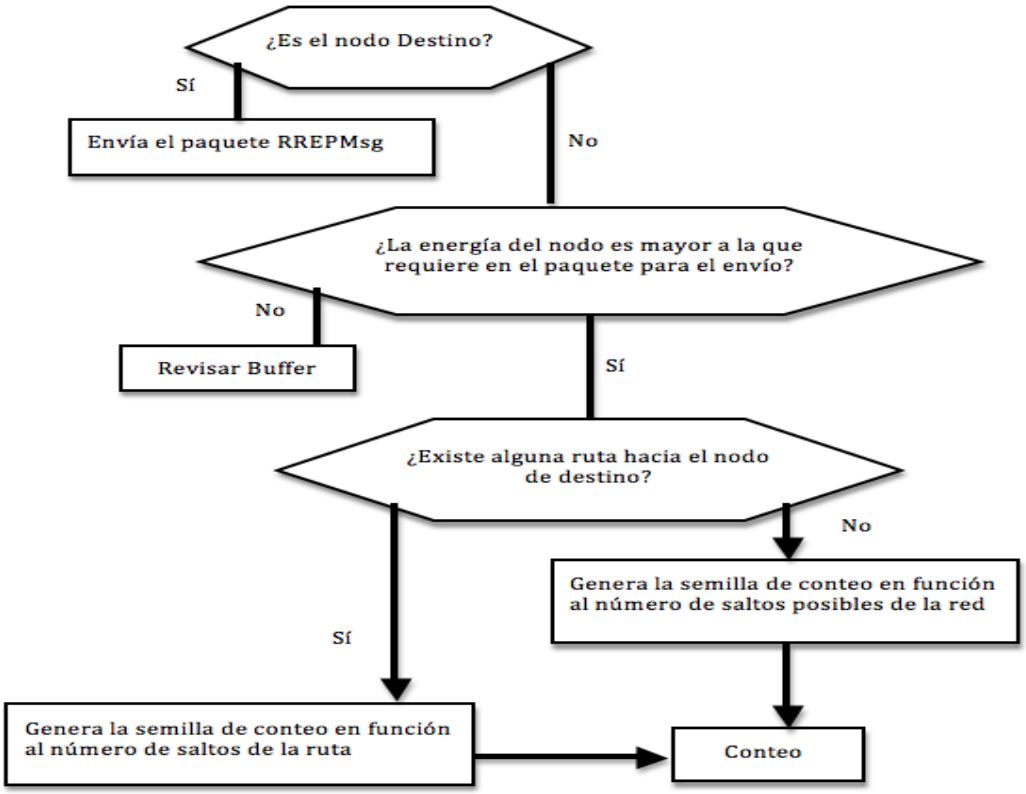


Figura 5.10 Segunda parte del algoritmo de inundación en un nodo.

La ventaja de este protocolo es que cada vez que se recibe un paquete RREQMsg o un paquete RREPMsg en los nodos, se generan nuevas rutas para conocer cómo llegar a los nodos vecinos o al nodo fuente, o bien, los actualiza. De esta manera cada nodo conoce la ubicación de cada uno de los nodos que integran la red. Sin embargo, el protocolo sólo tiene la capacidad de actualizar las rutas mediante el número de secuencia más no de investigar si los nodos continúan activos.

5.2.1.1 Proceso de Conteo

En la etapa de inundación existe el proceso de conteo que se encarga de generar un retardo aleatorio previo a la retransmisión. Esta función sirve para evitar que, en el caso de que más de un nodo esté procesando un mismo paquete, intente transmitir al

mismo tiempo que otro provocando una colisión. En la simulación y en la implementación la función conteo genera un valor aleatorio antes comenzar el conteo, en la figura 5.11 se muestra el comportamiento de la etapa de conteo dentro de la simulación.

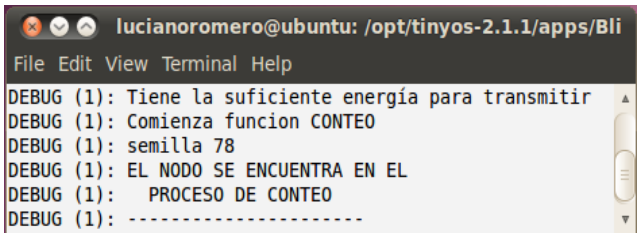


Figura 5.11 Función Conteo en la simulación.

En caso de que un nodo termine el conteo, éste debe de enviar una notificación avisando que realizará la retransmisión del paquete RREQMsg. Dicha notificación se hace mediante el paquete ACKImpMsg que se transmite en *broadcast*. Cualquier nodo que se encuentre procesando el mismo paquete RREQMsg y escuche la notificación deberá detener el procesamiento de dicho paquete. En la figura 5.12 se observa la simulación al momento de terminar el conteo y la preparación para hacer el envío de un paquete ACKImpMsg hacia los demás nodos de la red, esto para avisar que dicho nodo se encargara de retransmitir el paquete RREQMsg.

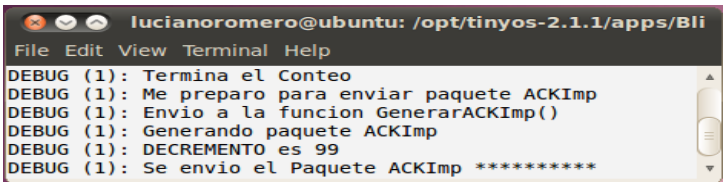


Figura 5.12 Fin de la función Conteo, se ejecuta el envío de un paquete ACKImp.

El conteo máximo se calcula considerando dos supuestos: si el nodo no conoce una ruta al nodo de destino entonces el valor aleatorio se genera al considerar el máximo número de saltos que podría darse en la red o si el nodo si conoce una ruta al nodo de destino entonces el valor aleatorio se genera al considerar el máximo número de saltos para llegar al nodo destino si se cuenta con una ruta a éste. Por lo tanto la posibilidad de que los nodos hagan el mismo conteo existe pero con probabilidad muy baja. Este mecanismo para la retransmisión evita el reenvió innecesario de paquetes RREQMsg por lo que es un elemento para el ahorro de energía.

5.2.1.2 Intensidad de la señal

Una parte importante en la que se basa el protocolo EARWSN para tomar decisiones en los envíos y ahorrar energía es mediante la lectura del RSSI. Para diseñar esta parte del algoritmo se hacen lecturas de RSSI de los nodos para guardar dicha información en las tablas de ruteo, de esta manera cada nodo puede elegir al nodo más alejado que se encuentre dentro de su cobertura para hacer el envío (aquel que tenga el RSSI más bajo) y evitar saltos innecesarios. Sin embargo, la versión de TOSSIM utilizada no permite simular las lecturas de RSSI de los motes MICAz, así que se recurrió a generar números aleatorios para tomarlos en lugar de los valores de RSSI y así verificar el algoritmo en este sentido.

Para entender un poco más esta parte del algoritmo se muestra un ejemplo en figura 5.13. Se considera a un nodo que tiene como vecinos a los nodos A, B y C, todos se encuentran dentro de su radio de cobertura, pero elegirá al que esté más alejado para reducir el número de saltos, en este ejemplo elegirá al nodo B ya que es el nodo con menor RSSI.

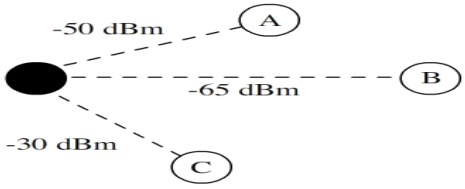


Figura 5.13 escenario con nodos con diferentes RSSI.

5.2.2 Algoritmo considerado para procesar los mensajes de confirmación

En caso de que algún nodo reciba un paquete de confirmación ACKImpMsg durante el conteo debe terminar con el procesamiento del paquete RREQMsg. En la figura 5.14 se muestra el diagrama de la función fired la cual es la encargada de verificar si existe un ACKImpMsg en buffer.

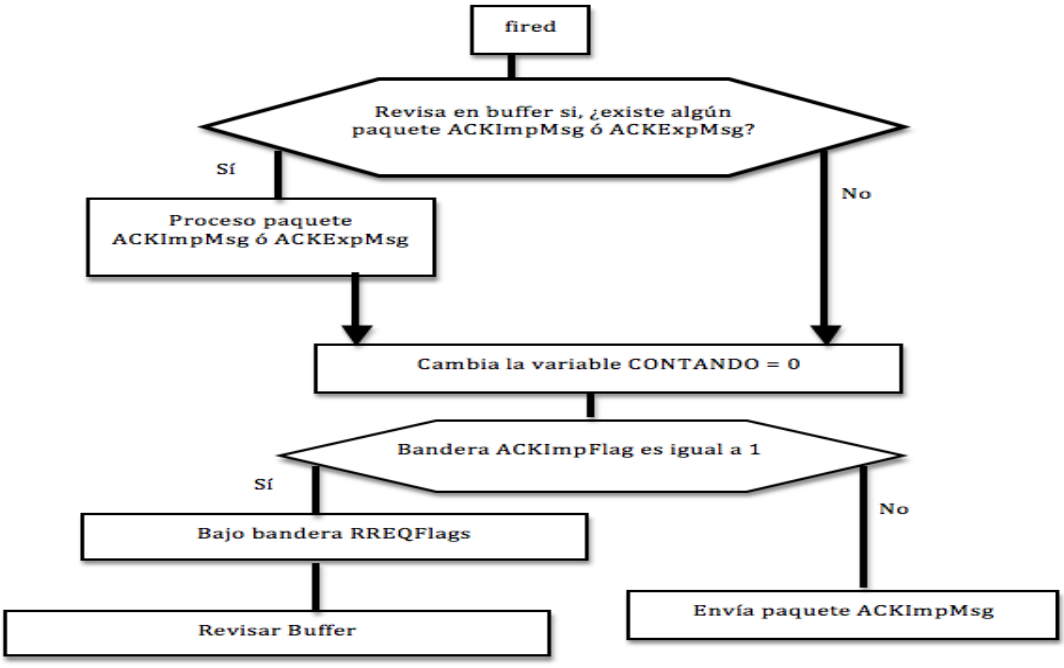


Figura 5.14 Diagrama de la función Fired.

Para evitar el caso de que algún otro nodo fuera de la cobertura del nodo que envió el ACKImpMsg esté procesando aún el paquete RREQMsg actual, el nodo que

previamente había transmitido el paquete RREQMsg, al enterarse mediante el paquete ACKImpMsg de que algún nodo se encargará de la retransmisión, debe enviar un mensaje de confirmación explícito ACKExpMsg para avisar a los nodos dentro de su radio de cobertura que detengan el proceso. En la figura 5.15 se observa un ejemplo de los paquetes ACKImpMsg y ACKExpMsg. En esta figura se observan cuatro nodos, el nodo N que transmitió el paquete RREQMsg previamente, el nodo N1 que se encargará de retransmitir nuevamente y los nodos N3 y N2 que continúan procesando el paquete RREQMsg. N1 transmite la notificación de que éste se encargara de la retransmisión, misma que escucha N y N2. Sin embargo, N3 no puede escuchar dicha notificación dado que no está en la cobertura de N1. Por tanto N, al enterarse de la notificación, envía una confirmación "explicita" para los posibles nodos que continúen procesando la solicitud de ruta, como N3 en la figura. Si hubiese más nodos procesando la solicitud de ruta al recibir la notificación terminarían con su proceso. Dado que los nodos N1 y N2 están dentro de la cobertura de N también recibirán el paquete ACKExpMsg pero no lo procesaran ya que el programa está diseñado para que se deseche esta acción.

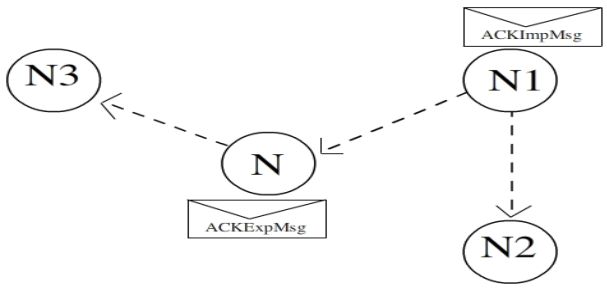


Figura 5.15 Envío del paquete ACKExpMsg.

Cuando la simulación procesa un paquete ACKImpMsg, al igual que para los demás paquetes, muestra en pantalla el valor de sus campos por lo que indica que se está procesando en el algoritmo adecuado, figura 5.16.

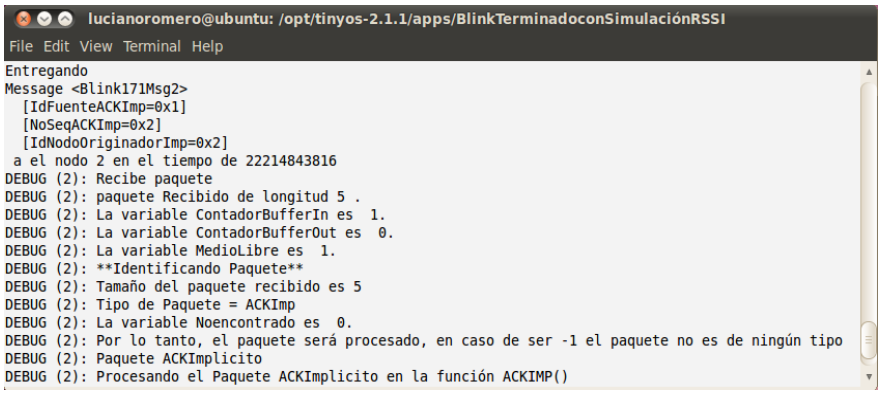


Figura 5.16 Procesamiento del paquete ACKImpMsg en la simulación.

Los paquetes ACKImpMsg y ACKExpMsg, al momento de ser procesados por los nodos, sólo cumplen con la función de confirmar los reenvíos del paquete RREQMsg y no de generar nuevas rutas debido a la poca información que contienen.

5.2.3 Respuesta de ruta

Cuando el paquete RREQMsg llega al nodo de destino, el nodo crea un nuevo mensaje para difundir, el cual tiene como objetivo responder a la solicitud de ruta. El mensaje es el RREPMsg, los nodos lo procesan de acuerdo al diagrama de la figura 5.17.

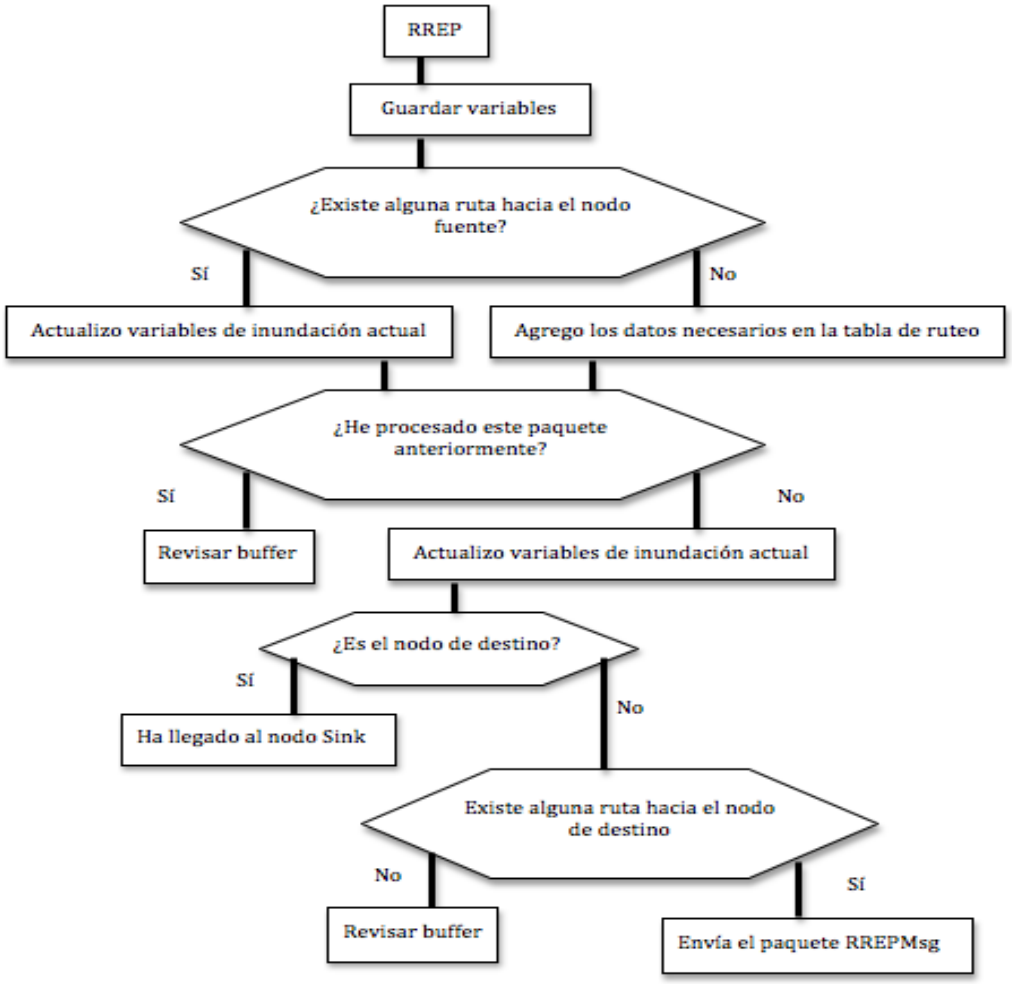


Figura 5.17 Algoritmo de respuesta de ruta (RREP)

El procesamiento de este paquete está basado en el procesamiento del paquete de respuesta de ruta RREP del protocolo de enrutamiento AODV [8], el cual hace énfasis en tomar en cuenta las siguientes indicaciones para la ruta de regreso:

- El nuevo paquete RREPMsg solo lo podrá generar una vez que llegó al nodo de destino.
- El nuevo paquete RREPMsg será enviado hacía el nodo que originó la inundación usando la misma ruta descubierta mediante la inundación, sólo que en sentido contrario.

- Se conserva el número de secuencia ya que la inundación y la respuesta se toman como una sola inundación.

- El nuevo paquete RREPMsg genera nuevas rutas a los nodos que lo procesen, se invierten los papeles ya que el nodo de fuente es el que anteriormente era el nodo de destino y el nodo de destino ahora será el nodo que anteriormente era el nodo fuente.

- Se hace una actualización en su campo número de saltos para indicar a cuántos saltos se encuentran los nodos que lo reciban.

El llenado del paquete RREPMsg en el nodo de destino se hace mediante los campos del paquete RREQMsg como en la figura 5.18. Se requiere de una variable para indicar el número de saltos que lleva el nuevo paquete en el camino de regreso, misma que se inicializa en 1.

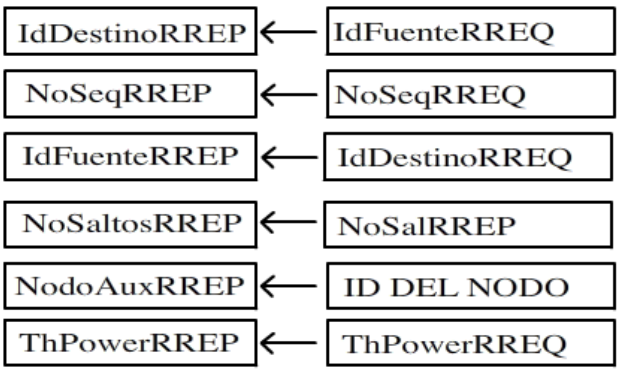


Figura 5.18 Llenado del paquete RREPMsg en el nodo de destino.

La respuesta de ruta mediante el paquete RREPMsg se puede observar en el ejemplo de la figura 5.19 en el cual se considera que N6 fue el nodo de destino. Una vez que el paquete llegó al nodo de destino se crea la respuesta de ruta y se envía por el camino por el que llegó. Si la inundación comenzó en el nodo *sink* y pasó por los nodos

N2, N3 y N5 para llegar a su destino, el paquete regresará al origen mediante envíos *unicast* a cada nodo de la ruta.

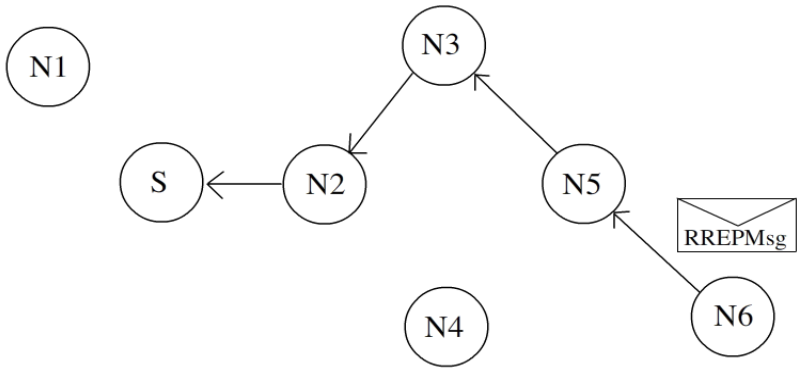


Figura 5.19 Ruta de regreso del paquete RREPMsg.

Los nodos intermedios como N5, N3 y N2 generan un nuevo paquete RREPMsg con los campos del paquete RREPMsg que les llegó anteriormente. Éstos al no ser el nodo de destino incrementan el número de saltos en el nuevo paquete de respuesta y retransmiten el paquete hasta que llega al nodo fuente. Este proceso les permite conocer que el nodo N6 es su vecino o en caso de no serlo, agregaran una ruta para poder llegar a éste. Es decir el reenvío del paquete RREPMsg también genera nuevas rutas para la tabla de enrutamiento de cada nodo. Por ejemplo, en la figura 5.19 el nodo N4 que no forma parte de la ruta de regreso, al recibir la retransmisión del paquete RREPMsg por parte de N5, agregara a N6 en su tabla de enrutamiento.

En la siguiente figura se observa la simulación ejecutando el procesamiento de un paquete RREPMsg cuando el nodo que lo recibió es el nodo de destino automáticamente crea otro paquete RREPMsg para enviarlo, figura 5.20.

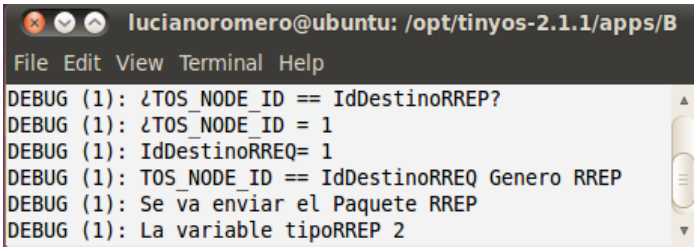


Figura 5.20 Procesamiento de un paquete RREPMsg en la simulación.

Por último cabe señalar que el protocolo no toma en cuenta las funciones del nodo fuente, solo se encarga de procesar los datos que se reciben de la RIS en cada inundación o petición.

Capítulo VI

Conclusiones y Trabajo Futuro

6.1 Conclusiones

Se logró implementar en el sistema operativo TinyOS un protocolo de enrutamiento eficiente en el uso de energía. Se verificó su funcionamiento mediante la simulación. Además se logró concretar los siguientes puntos:

- Se generaron los algoritmos necesarios para la programación del protocolo basados en el funcionamiento de los protocolos de EARWSN y AODV. Estos algoritmos se pueden utilizar para implementar el protocolo en cualquier otro lenguaje de programación y utilizarse en diferentes plataformas.

- Se generó un código con el lenguaje de programación NesC que podrá instalarse en los motes de la familia MICAz, este código fue ejecutado en la plataforma sin presentar ningún error de compilación, basado en los algoritmos del protocolo aquí presentados.

- Se genero un código en el lenguaje Python que permite la ejecución de la simulación, dicho código puede seguir siendo utilizado para futuras pruebas.

6.2 Trabajo a futuro

Se pretende a corto plazo, realizar la implementación física en motes que trabajen bajo la tecnología MICAz o cualquier mote con transceptor C2024 para crear una RIS. La implementación física hará posible verificar ciertos aspectos del protocolo implementado como la generación adecuada de las tablas de enrutamiento.

También se debe de verificar la vida útil de los motes, tomando en cuenta el número de transmisiones que hace un nodo hasta agotar su batería, por lo tanto, se conocerán cuántos envíos puede hacer cada nodo y con ello el momento en que un nodo debe reemplazar su batería. Además debe de compararse el protocolo implementado con otros protocolos usados en otras tecnologías existentes que también estén enfocadas al bajo consumo de energía en las RIS para verificar sus prestaciones.

Apéndice A

Instalación de TinyOs en Ubuntu 10.4

El sistema operativo TinyOs se instaló en la distribución de Linux Ubuntu versión 10.4, [21]. En los siguientes párrafos se detallan los pasos para la instalación.

A1. Añadir los enlaces en donde se encuentran los archivos de TinyOs.

El usuario debe ejecutar en una terminal de línea de comandos las instrucciones para abrir el archivo source.list, que se encuentra en la ruta: *etc/apt/.* Para abrir el archivo se utiliza la instrucción *sudo gedit source.list.* Consecutivamente, se debe teclear la contraseña del usuario o administrador según sea el caso, para continuar el proceso de apertura del mismo. Una vez abierto el archivo, se añade al final de éste, el siguiente link que contiene los repositorios: *deb http://hinrg.cs.jhu.edu/tinyos karmic main.*

A2. Actualizar aplicaciones

Se debe actualizar la lista de aplicaciones mediante la instrucción *sudo apt-get update,* como se muestra en la figura A1. El comando *apt-get* se usa para administrar paquetes instalables disponibles en repositorios.

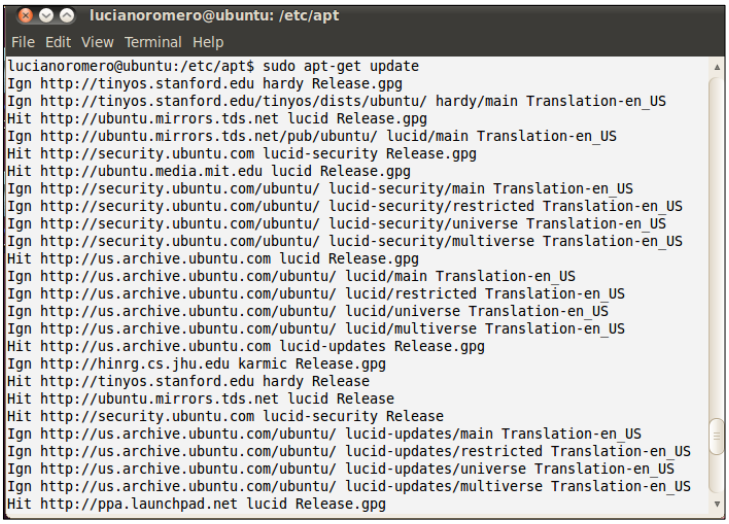


Figura A1 Instalación de repositorios.

A3. Instalación de TinyOs

A continuación, se ejecuta el comando: *sudo apt-get install tinyos,* para instalar el sistema operativo, tal como se muestra en la figura A2.

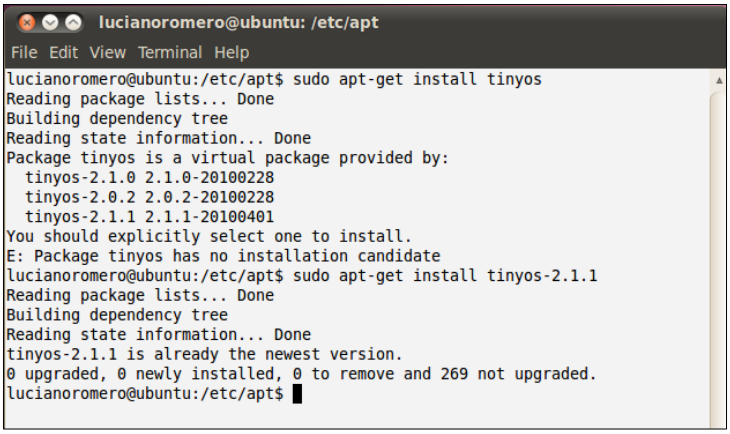


Figura A2 Comando para instalar TinyOs.

Posteriormente, se debe elegir la versión que se desea instalar de TinyOs. En este caso se usó la versión 2.1.1. Esto se realiza con el comando *sudo apt-get install tinyos-2.1.1.*

A4. Evitar errores de ejecución de TOSSIM

Se deben colocar los siguientes enlaces tal como se realizó en el punto 1, para evitar errores en la ejecución del software TOSSIM, una vez que haya finalizado la instalación de TinyOs. Después, ejecutar nuevamente el punto 2.

deb http://ppa.launchpad.net/fkrull/deadsnakes/ubuntu lucid main

deb-src http://ppa.launchpad.net/fkrull/deadsnakes/ubuntu lucid main

deb http://ubuntu.mirrors.tds.net/pub/ubuntu/ lucid main

A5. Instalación de Python

Para instalar Python versión 2.5, así como sus herramientas, será necesario ejecutar las siguientes líneas de código en la terminal:

sudo apt-get install python2.5 python2.5-dev python2.5-old-doctools python2.5-gdb

sudo apt-get install cvs subversion autoconf automake1.9 python-dev

Otra aplicación que requiere ser instalada para compilar TinyOs es g++:

sudo apt-get install g++ gperf swig graphviz alien fakeroot

Una forma diferente de instalar Python 2.5 y el compilador g++, es por medio de la aplicación llamada Gestor de paquetes Synaptic, del menú Sistema, opción Administración, el cual le permite al usuario visualizar en modo gráfico el proceso de instalación de los programas en el equipo, además de gestionar los que se están instalando, tal como se muestra en la figura A3.

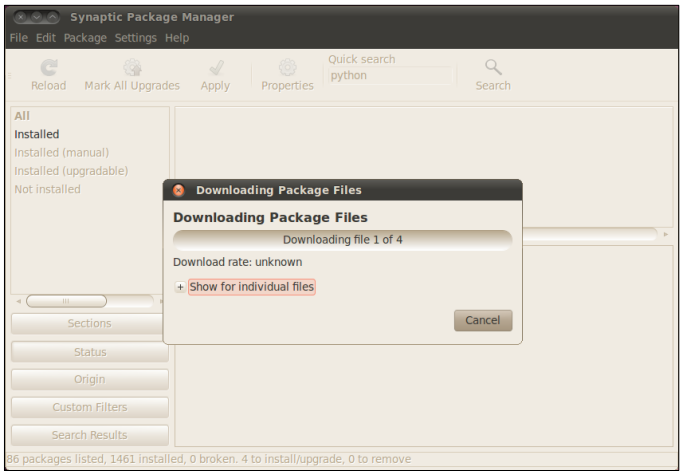


Figura A3 Instalación por medio de Synaptic.

A6. Configuración del entorno de desarrollo TinyOs

Para realizar la configuración y visualizarla cuando el usuario abre una terminal de línea de comandos, se deberá abrir el archivo llamado .bashrc o .profile que es un archivo oculto y posteriormente añadir las siguientes líneas:

source /opt/tinyos-2.1.1/tinyos.sh

export CLASSPATH=/opt/tinyos-2.1.1/support/sdk/java/tinyos.jar:.

Se puede utilizar el comando *dpkg –l | grep bashrc,* para buscar el archivo en los directorios del sistema.

Después de instalar el entorno de desarrollo, la terminal se visualiza como la figura A4.

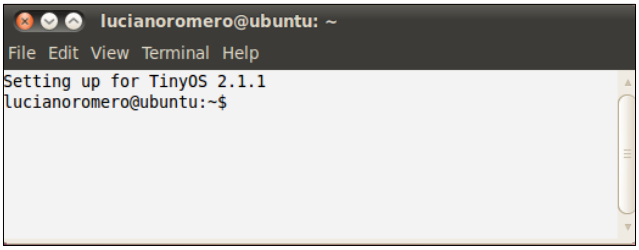


Figura A4 Entorno de desarrollo TinyOs.

A7. Cambio de versión de Python

Hasta el momento se han instalado y actualizado los archivos ejecutables de TinyOs, sin embargo TOSSIM, el simulador de TinyOs funciona únicamente bajo la versión 2.5 o 2.4 de Python. Si no se conoce la versión que se dispone, el usuario deberá acceder a la siguiente ruta desde el directorio raíz: *usr/lib/python/config/*. La carpeta "config" deberá contener más de dos archivos, sino es así, la instalación no está completa y se deberá reinstalar nuevamente el paso 7. En caso contrario, se visualizarán varias carpetas, y una de ellas especificará la versión de Python.

Una vez verificado lo anterior, procedemos a cambiar la versión, accediendo al archivo *sim.extra,* con el comando, *sudo gedit opt/tinyos-2.1.1/support/make/sim.extra*. Allí se cambiará el valor numérico que se comprobó en el paso anterior, es decir, se cambiará la versión enseguida de la palabra PYTHON, ejemplo PYTHON=2.5.

A8. Compilación en TinyOs

Se deberá realizar una primera prueba para verificar que la plataforma de TinyOs se ha instalado correctamente. Para ello se compila el archivo llamado Blink, que es un programa simple escrito en NesC, que se encuentra en la ruta *etc/Tinyos-2.1.1/apps/Blink.* Una vez posicionados en ésta, se deberá compilar el archivo con el comando *make micaz.* Cabe resaltar que Make compila específicamente aplicaciones programadas mediante el lenguaje NesC y Python.

Se utiliza el comando *make micaz* por que la tecnología que se propuso para este trabajo es con base a Micaz, sin embargo es posible realizar la compilación bajo Telosb: *make*

telosb, Micadot: *make mikadot*, o Mica2: *make mica2*. Como se explicó en el punto anterior, TOSSIM es el simulador de TinyOs y se ejecuta bajo el comando: *make micaz sim.*

En la figura A5, se muestra la correcta compilación del archivo Blink. El resultado de la terminal arroja el tamaño en bytes de la aplicación Blink en memoria RAM y ROM, cuando se ejecuta *make telosb*.

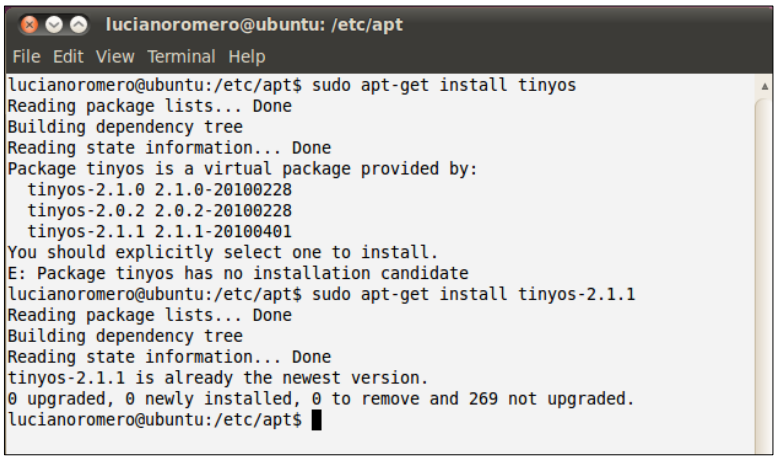


Figura A5 Compilación de Blink.

A9. Errores comunes al compilar.

Un error común es que no se encuentre la información completa del microcontrolador que tiene integrado el mote. En dicho caso, se deben considerar los siguientes pasos:

Si el microcontrolador es del tipo AVR, remover los archivos relacionados con el microcontrolador del mote con el siguiente código: *sudo apt-get remove avr_binutils-tinyos*

Verificar que se encuentren las siguientes carpetas: *usr/avr/bin* y *usr/bin/lib.*

Abrir el siguiente archivo:

sudo apt-get opt/tinyOs-2.1.1/doc/nesdoc/micaz/index.html y añadir el siguiente link: *http://packages.ubuntu.com/ hardy/all/avr-liba/download*.

Ejecutar en la terminal *sudo apt-get update* para actualizar.

Enseguida, instalar un nuevo paquete: sudo apt-get install avr-libc_1.4.7_all.deb

Nuevamente verificar las carpetas del segundo punto.

Posteriormente, ejecutar en la terminal: *make telosb*, en la ruta mencionada en el punto 8.

Finalmente, compilar el programa Blink y verificar que los datos arrojados son similares a los de la figura A5.

Referencias.

[1] Akyildiz F. I. and Mehmet V. C, (2010). Wireless Sensor Networks, 1ª Edition, Singapur: Wiley.

[2] Dharma P. A, De Morais C. C, (2011). Ad-Hoc and Sensor Networks Theory and Applications, 2ª Edition, Scientific Publishing.

[3] B.Warneke, M. Last, B. Liebowitz, K. S. J. Pister, (2001). Smart Dust: Communicating with a Cubic- Milimeter Computer, IEEE Computer, pp 44-51.

[4] J. Urraza, Smart Dust, Facultad de Ciencias y Tecnologías, UCNSA, Paraguay, (2008), pp 4-8.

[5] Tanenbaum S. A., (2003). Redes de Computadoras, 4ª Edición, México: Pearson Educación.

[6] Mahgoub I., & Ilyas M., (2006). Sensor Network Protocols. US: Edited by Library of Congress Cataloging in Publication Data.

[7] R Vidhyapriya, Drptvanath, (2007). EnergyAware Routing for Wireless Sensor Networks, Department of information Technology.

[8] The Internet Engineering Task Force (IETF), Ad hoc On-Demand Distance Vector (AODV) Routing, [en línea]; Julio 2003, [consulta: 1 Septiembre 2013], disponible en: < http://www..ieft.org/rfc/rfc3561.txt>

[9] IEEE 802.11n-2009; Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, (2012 revision), pp 8.

[10] IEEE 802.15.1 – 2005; Wireless medium access control (MAC) and physical layer (PHY) specifications for wireless personal area networks (WPANs), 2005, pp. 6.

[11] IEEE 802.15.2 – 2003; Coexistence of Wireless Personal Area Networks with other Wireless Devices Operating in Unlicensed Frecuency Bands, 2003, pp. 7.

[12] IEEE 802.15.3 – 2003; Wireless medium access control (MAC) and physical layer (PHY) Specifications for High Rate Wireless Personal Area Networks (WPANs), pp 5.

[13] IEEE 802.15.4 – 2003; Wireless medium access control (MAC) and physical layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs), pp 23-29.

[14] Aura Ganz, Zvi Ganz, Kitti Wongthavarawat (2003), Multimedia Wireless Networks: Technologies, Standards, and QoS, Publisher Prentice Hall, Upper Saddle River, NJ, pp 179-187.

[15] Wayne Tomasi, (2003). Sistemas de Comunicaciones Electrónicas, 4ª edición, México: PEARSON EDUCACIÓN, pp 606.

[16] Kazem Sohraby, Daniel Minoli, and Taieb Znati (2007). Wireless Sensor Networks, 1ª edition, US: Wiley-Interscience, pp 177-188.

[17] Micaz Datasheet, Crossbow Micaz Wireless Measurements System, (2010), Document Part Number 6020 0060 04 Rev A.

[18] Vetelino J.& Reghu A., (2011), Introduction to Sensors, NY: Taylor and Francis Group LLC.

[19] TinyOs,TinyOS, [en línea]; 11 de Noviembre de 2006, [consulta: 15 de Octubre de 2013], disponible: http://www.tinyos.net/

[20] TinyOs,TinyOS, [en línea]; 11 de Noviembre de 2006, [consulta: 15 de Octubre de 2013], disponible: *http://tinyos.stanford.edu/tinyos-wiki/index.php/TOSSIM*

[21] TinyOs,TinyOS, [en línea]; 11 de Noviembre de 2006, [consulta: 15 de Octubre de 2013], disponible: http://tinyos.stanford.edu/tinyos-wiki/index.php/Installing_TinyOS_2.1.1