TRABAJO FINAL DE CARRERA

TÍTULO DEL TFC/PFC : Evaluación de una red de sensores con protocolo AODV y tecnología radio IEEE 802.15.4

TITULACIÓN: Ingeniería Técnica de Telecomunicación, especialidad Telemática

AUTOR: Óscar Alonso Blanco

DIRECTOR: Carles Gomez i Montenegro

FECHA: 5 de septiembre de 2005

Título: Evaluación de una red de sensores con protocolo AODV y tecnología radio IEEE 802.15.4
Autor: Óscar Alonso Blanco

Director: Carles Gómez i Montenegro

Fecha: 27 de junio de 2005

Resumen

Entre las redes inalámbricas ad-hoc, las redes de sensores son las que poseen un futuro más prometedor, gracias a la multitud de usos civiles que se les puede dar. El encaminamiento es un aspecto crítico y de importancia en esta clase de redes. Este documento estudia y cuantifica el efecto que tiene el protocolo de encaminamiento AODV sobre una red de sensores real con tecnología radio IEEE 802.15.4 y sistema operativo tinyOS, caracterizando parámetros tales como el ancho de banda, la latencia, el consumo de batería y la capacidad de reacción respecto a cambios en la topología provocados por la propia red sensora.

Palabras clave: tinyOS, AODV, protocolo de encaminamiento, IEEE 802.15.4, medidas reales, banco de pruebas

Title: Test bed for sensor networks with AODV protocol and IEEE 802.15.4 radio technology
Author: Óscar Alonso Blanco

Director: Carles Gómez i Montenegro

Date: September, 4th 2005

Overview

Whithin the field of ad-hoc wireless networks, sensor networks have the most hopeful future, due to the amount of civilian applications that can give to us. In this kind of networks, routing is a critical and important research subject. This document studies and quantifies the effect of AODV routing protocol in a real sensor network using IEEE 802.15.4 radio technology and tinyOS operating system, characterizing parameters like bandwidth, latency, energy cost and reactivity after topology changes.

Key words: tinyOS, AODV, routing protocol, IEEE 802.15.4, real network measurements, test bed.

Agradecimientos:

Me gustaría dar las gracias a mi director de proyecto, Carles Gomez, que gracias a su inestimable orientación y ayuda ha hecho que este proyecto saliera hacia delante, también a Pere Salvatella autor de nstAODV a su infinita paciencia con mis preguntas y al soporte que me ha prestado para la adaptación de este protocolo.

Dedico este proyecto a toda mi familia, en especial a mis padres que me han tenido bajo su ala durante todos estos años y me han proporcionado la estabilidad necesaria para llegar al final de este camino.

Dedico también este proyecto y de forma muy especial a Emma, por estar ahí, por ser mi cómplice , por entender de mi lo que otros no entienden y darme aliento cuando a veces me ahogaba .

También me quiero acordar de todas las personas y compañeros que a lo largo de esta vida me han ayudado a formarme tanto humanamente como técnicamente.

A todos muchas gracias!

Óscar

P.D:*Nuestra recompensa se encuentra en el esfuerzo y no en el resultado. Un esfuerzo total es una victoria completa." Mahatma Gandhi*

ÍNDICE

INTRODUCCIÓN

La evolución en tamaño y prestaciones que nos ofrece la electrónica hoy en DIA, está dando paso a pequeños nuevos sensores con capacidades de comunicación inalámbricas, capacidad de proceso y autonomía de funcionamiento.

La conjunción de estas 3 propiedades nos va a permitir la creación de redes móviles de sensores Ad-Hoc, es decir, redes sin ninguna infraestructura las cuales nos van a ofrecer un amplio abanico de nuevas utilidades con un coste ambiental y de despliegue más reducido que otras soluciones.

En el presente proyecto hemos realizado la adaptación, instalación y posterior evaluación del protocolo de encaminamiento AODV en dispositivos sensores que implementan el novedoso IEEE 802.15.4. comercialmente llamado ZigBee, los sensores utilizan un sistema operativo (S.O.) para sistemas empotrados de gran aceptación por parte de la industria y de libre distribución llamado TinyOS, hasta el momento no se tiene constancia de un trabajo con características similares, ya que los resultados obtenidos de estas pruebas provienen de situaciones reales y no de aproximaciones obtenidas con simulaciones.

El documento está compuesto por seis capítulos. En el capítulo uno se hará una introducción a las redes Ad-Hoc con especial atención a las redes de sensores. En el capítulo dos se hará una breve introducción a IEEE 802.15.4 dado que es una tecnología inalámbrica novedosa. En el capítulo tres se verá el funcionamiento del sistema operativo de libre distribución TinyOS. En el capítulo cuatro se hará una descripción del funcionamiento del protocolo AODV estándar y analizaremos el protocolo nstAODV. Luego, en el capítulo 5 se hará una descripción de todos los escenarios y su funcionalidad desde un punto de vista teórico. En el capítulo 6 se comentaran los resultados derivados de los escenarios del capítulo anterior. Por último se extraerán las conclusiones ocasionadas de los capítulos anteriores.

CAPÍTULO 1. Introducción a las redes de sensores

En este capítulo haremos una introducción a esta clase de redes comentando lo que son, su historia, sus características y sus todos sus posibles usos en un ámbito civil.

1.1. ¿Qué son?

Las redes de sensores están formadas por un grupo de sensores con ciertas capacidades sensitivas y de comunicación los cuales permiten formar redes inalámbricas Ad-Hoc sin infraestructura física preestablecida ni administración central.

Las redes de sensores es un concepto relativamente nuevo en adquisición y tratamiento de datos con múltiples aplicaciones en distintos campos tales como entornos industriales, domótica, entornos militares, detección ambiental.

Esta clase de redes se caracterizan por su facilidad de despliegue y por ser autoconfigurables, pudiendo convertirse en todo momento en emisor, receptor, ofrecer servicios de encaminamiento entre nodos sin visión directa, así como registrar datos referentes a los sensores locales de cada nodo. Otra de sus características es su gestión eficiente de la energía, que con ello conseguimos unas alta tasa de autonomía que las hacen plenamente operativas.

Cada nodo, como ente individual de una red de sensores, no deja de ser una pequeña computadora, con un pequeño procesador, una memoria de programa y una memoria para almacenar variables, pero al que también agregamos unos pequeños periféricos I/O (entrada/salida) tales como un transceptor radio y un pequeño conversor A/D (Analógico/Digital) que sirve para adquisición de los datos de los sensores locales.

A modo de ejemplo los sensores con los que trabajamos en este proyecto poseen un procesador ATMEL MEGA128 a 7.3Mhz con 128 kB de memoria de programa (Flash) y 4 kB de memoria de variable (RAM), con la novedosa interficie radio ZigBee (IEEE 802.15.4) de la casa Chipcon (cc2420).

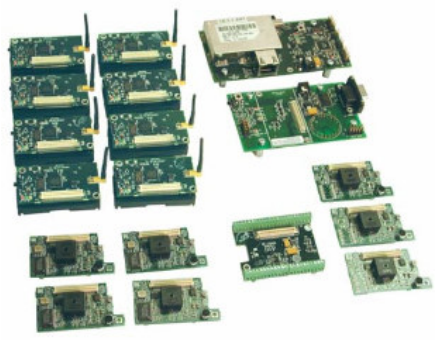


Fig. 1.1 Kit de sensores MICAZ de la casa xbow

1.2. Historia

Las redes de sensores nacen, como viene siendo habitual en el ámbito tecnológico, de aplicaciones de carácter militar.

La primera de estas redes fue desarrollada por Estados Unidos durante la guerra fría y se trataba de una red de sensores acústicos desplegados en el fondo del mar cuya misión era desvelar la posición de los silenciosos submarinos soviéticos, el nombre de esta red era SOSUS (Sound Surveillance System). Paralelamente a ésta, también EE.UU. desplegó una red de radares aéreos a modo de sensores que han ido evolucionando hasta dar lugar a los famosos aviones AWACS, que no son más que sensores aéreos.

SOSUS ha evolucionado hacia aplicaciones civiles como control sísmico y biológico, sin embargo AWACS sigue teniendo un papel activo en las campañas de guerra.

A partir de 1980, la DARPA comienza un programa focalizado en sensores denominado DSN (Distributed Sensor Networks), gracias a él se crearon sistemas operativos (Accent) y lenguajes de programación (SPLICE) orientados de forma específica a las redes de sensores, esto ha dado lugar a nuevos sistemas militares como CEC (Cooperative Engadgement Capability) consistente en un grupo de radares que comparten toda su información obteniendo finalmente un mapa común con una mayor exactitud, existen también un gran número de sistemas de los que solo nombraremos sus siglas y no entraremos en su funcionamiento porque no es el objetivo de este proyecto: FDS, ADS, REMBASS, TRSS.

Estas primeras redes de sensores tan sólo destacaban por sus fines militares, aún no satisfacían algunos requisitos de gran importancia en este tipo de redes tales como la autonomía y el tamaño.

Entrados en la década de los 90, una vez más DARPA lanza un nuevo programa enfocado hacia redes de sensores llamado SensIt, su objetivo viene a mejorar aspectos relacionados con la velocidad de adaptación de los sensores en ambientes cambiantes y en como hacer que la información que recogen los sensores sea fiable.

Es a finales de los años 90 y principios del siglo 21 que los sensores han empezado a coger una mayor relevancia en el ámbito civil, decreciendo en tamaño e incrementando su autonomía. Compañías como Crossbow han desarrollado nodos sensores del tamaño de una moneda de 2 euros con la tecnología necesaria para cumplir su cometido funcionando con pilas que les hacen tener una autonomía razonable y una independencia inédita.

El futuro ya ha empezado a ser escrito por otra compañía llamada Dust Inc compuesta por miembros del proyecto Smart Dust ubicado el Berkeley, que ha creado nodos de un tamaño inferior al de un guisante y que, debido a su minúsculo tamaño, podrán ser creadas múltiples nuevas aplicaciones.

1.3. Características de una red sensora

Las redes de sensores tienen una serie de características propias y otras adaptadas de las redes Ad-Hoc:

- **Topología Dinámica:** En una red de sensores, la topología siempre es cambiante y éstos tienen que adaptarse para poder comunicar nuevos datos adquiridos

- **Variabilidad del canal:** El canal radio es un canal muy variable en el que existen una serie de fenómenos como pueden ser la atenuación, desvanecimientos rápidos, desvanecimientos lentos e interferencias que puede producir errores en los datos.

- **No se utiliza infraestructura de red:** Una red sensora no tiene necesidad alguna de infraestructura para poder operar, ya que sus nodos pueden actuar de emisores, receptores o enrutadores de la información. Sin embargo, hay que destacar en el concepto de red sensora la figura del nodo recolector (también denominados sink node), que es el nodo que recolecta la información y por el cual se recoge la información generada normalmente en tiempo discreto. Esta información generalmente es adquirida por un ordenador conectado a este nodo y es sobre el ordenador que recae la posibilidad de transmitir los datos por tecnologías inalámbricas o cableadas según sea el caso.

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

- **Comunicaciones multisalto o broadcast:** En aplicaciones sensoras siempre es característico el uso de algún protocolo que permita comunicaciones multi-hop, léase AODV, DSDV, EWMA u otras, aunque también es muy común utilizar mensajería basada en broadcast.

- **Consumo energético:** Es uno de los factores más sensibles debido a que tienen que conjugar autonomía con capacidad de proceso, ya que actualmente 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 radio con la misma característica, a esto hay que agregar un software que también conjugue esta característica haciendo el consumo aún más restrictivo.

- **Limitaciones hardware:** Para poder conseguir un consumo ajustado, se hace indispensable que el hardware sea lo más sencillo posible, así como su transceptor radio, esto nos deja una capacidad de proceso limitada.

- **Costes de producción:** Dada que la naturaleza de una red de sensores tiene que ser en número muy elevada, para poder obtener datos con fiabilidad, los nodos sensores una vez definida su aplicación, son económicos de hacer si son fabricados en grandes cantidades

1.4. Aplicaciones

Dentro del campo de las redes móviles Ad-Hoc, las redes de sensores son las que parecen tener un futuro más prometedor.

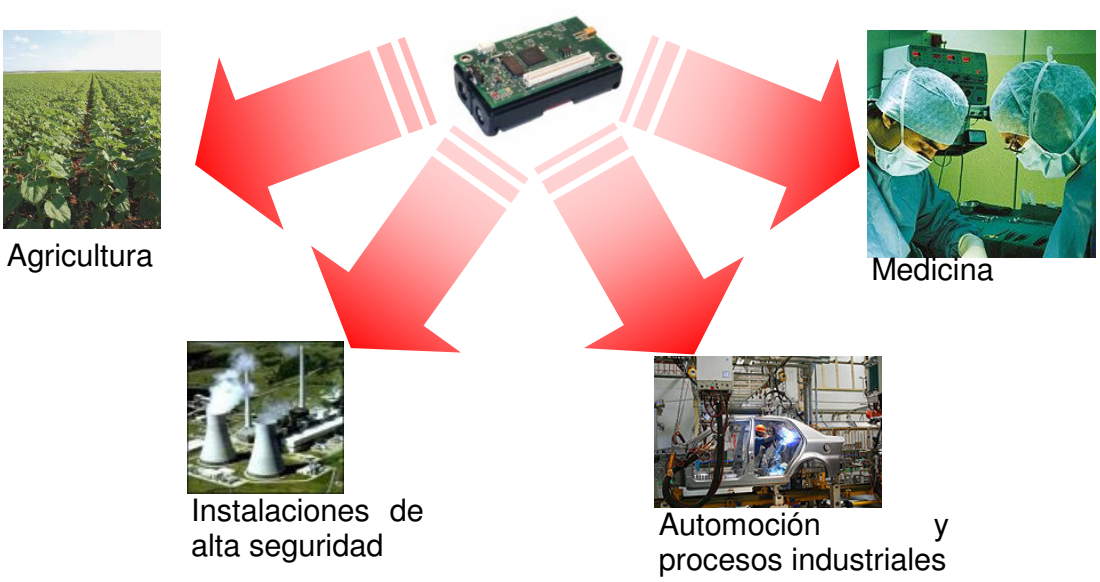


Fig. 1.2. Aplicaciones posibles de una red de sensores.

Pasando de largo las aplicaciones militares que antes hemos comentado en la historia de las redes de sensores (apartado 1.2.), éstas tienen usos civiles interesantes como vemos en la figura 1.2 y también a continuación:

- **Entornos de alta seguridad:** Existen lugares que requieren altos niveles de seguridad para evitar ataques terroristas, tales como centrales nucleares, aeropuertos, edificios del gobierno de paso restringido. Aquí gracias a una red de sensores se pueden detectar situaciones que con una simple cámara sería imposible.

- **Sensores ambientales:** El control ambiental de vastas áreas de bosque o de océano, sería imposible sin las redes de sensores. El control de múltiples variables, como temperatura, humedad, fuego, actividad sísmica así como otras. También ayudan a expertos a diagnosticar o prevenir un problema o urgencia y además minimiza el impacto ambiental del presencia humana.

- **Sensores industriales:** Dentro de fábricas existen complejos sistemas de control de calidad, el tamaño de estos sensores les permite estar allí donde se requiera.

- **Automoción:** Las redes de sensores son el complemento ideal a las cámaras de tráfico, ya que pueden informar de la situación del tráfico en ángulos muertos que no cubren las cámaras y también pueden informar a conductores de la situación, en caso de atasco o accidente, con lo que estos tienen capacidad de reacción para tomar rutas alternativas.

- **Medicina:** Es otro campo bastante prometedor. Con la reducción de tamaño que están sufriendo los nodos sensores, la calidad de vida de pacientes que tengan que tener controlada sus constantes vitales (pulsaciones, presión, nivel de azúcar en sangre, etc), podrá mejorar substancialmente.

- **Domótica:** Su tamaño, economía y velocidad de despliegue, lo hacen una tecnología ideal para domotizar el hogar a un precio asequible.

CAPÍTULO 2: Capa Física MAC IEEE 802.15.4 (ZigBee)

En este capítulo haremos una introducción a este novedoso estándar, viendo las motivaciones que llevaron a su creación, así como un breve repaso de todas sus características y modos de funcionamiento. También hablaremos muy brevemente del transceptor radio utilizado.

2.1. Introducción

En el campo de los estándares inalámbricos disponemos de un gran número de estos (Bluetooth,wi-fi,wi-max...) orientados hacia aplicaciones con altos requerimientos de ancho de banda (redes domésticas y de oficina, videoconferencia, voip, etc...), pero se ha echado a faltar algún estándar inalámbrico específico para redes de sensores.

El inconveniente de utilizar cualquiera de los estándares inalámbricos antes mencionados radica en el gran consumo de energía y ancho de banda que utilizan frente a la baja tasa de bits enviados por cualquier aplicación sensora o de control y sus bajos requerimientos de energía.

En un principio, cada fabricante de nodos sensores ha optado por utilizar soluciones propietarias dadas la presión ejercida por el mercado, lo que trajo problemas de interoperabilidad entre los diversos fabricantes.

La industria entiende que hace falta un nuevo estándar que aúne autonomía, envío de datos de baja capacidad (kbps) y un bajo coste, es con esto con lo que nace el estándar 802.15.4 comercialmente llamado ZigBee. Se ha convenido llamar a esta clase de redes LR-WPAN (Low Rate Wireless Personal Area Network), dado sus bajas tasas de transmisión y su corto alcance.

2.2. Características

2.2.1. Topologías

IEEE 802.15.4 dispone de 2 tipos distintos de nodos:

- *FFD (Full Function Device):* que son dispositivos que pueden realizar cualquier tarea de las indicadas posteriormente.
- *RFD (Reduced Function Device):* dispositivos con funcionalidad reducida.

Dependiendo de la aplicación que vayamos a realizar existen 2 topologías:

- *Topología en estrella (star):* Todos los nodos de una misma WPAN están coordinados por un único nodo FFD que recibe el nombre de PAN coordinator y entre sus principales tareas se encuentran la de coordinar el acceso al medio.

El PAN coordinator, al tener el papel principal de "organizador", posee unos requerimientos de energía mayores que el resto de nodos y es por ello que este suele ser un nodo con una unidad de energía no agotable (conectado a la red eléctrica).

Esta topología suele ser utilizada en domótica, ordenadores personales, juguetes o para cuidados médicos personales.

- *Topología peer to peer:* Todos sus nodos suelen ser FFD, ya que todos tienen la misma prioridad de acceso al medio, y aunque existe un PAN coordinator, éste no tiene las mismas funciones relevantes. Difiere con una topología en estrella en que cualquier nodo puede contactar con otro sin permiso del PAN coordinator. Se corresponde con arquitecturas de red mesh o Ad–Hoc.

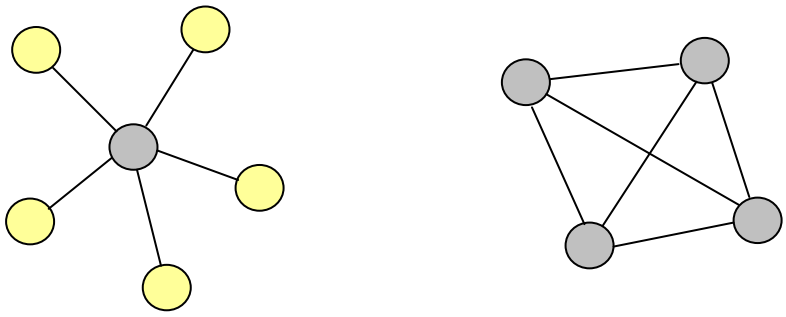


Fig. 2.1 Topología estrella y topología peer to peer

2.2.2. Capa física

La capa física está separada en dos subcapas: PHY data service y PHY management.

Algunas características globales de la capa física son el control del transceptor radio, calidad del enlace (LQI), selección de canal, detector de energía(ED), detección de portadora (CCA) para su uso en CSMA-CA a nivel MAC, etc…

La capa física puede transmitir en 3 bandas distintas en la tabla 2.1. tenemos una descripción de las mismas con sus características:

Tabla 2.1 Parámetros establecidos en función de la banda utilizada.

PHY (MHz)	Frequency band (MHz)	Chip Rate(Kchip/s)	Modulación	Kbps	Ksimbolos/s	2^n
868/915	868–868.6	300	BPSK	20	20	1
	902–928	600	BPSK	40	40	1
2450	2400–2483.5	2000	O-QPSK	250	62.5	4

En la banda de 2,4Ghz, que es la más eficiente en cuanto al uso del ancho de banda, disponemos de un total de 16 bandas o canales de 2 Mhz con una distancia entre canales de 5Mhz para evitar interferencias, tal y como vemos en la figura 2.2.

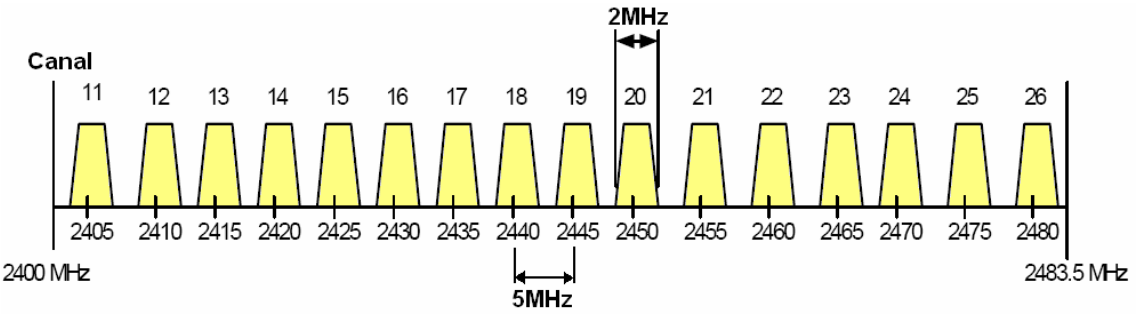


Fig. 2.2 División espectral de canales en la banda de 2.4Ghz

Para ello utiliza Tecnología de espectro expandido de secuencia directa (DSSS).

2.2.3. Capa MAC

La capa MAC se encarga de los siguientes cometidos:

- Generar beacons en el caso de ser un PAN coordinator y que el resto de nodos se sincronicen al ritmo de los beacons.
- Mecanismo de acceso al medio CSMA-CA.
- Asociación o desasociación a una PAN.
- Funciones de seguridad (encriptación AES).
- QoS mediante GTS (Granted Time Slot).
- Mecanismos de fiabilidad entre nodos (ACK's).

La ventaja de este nivel MAC respecto al de otros estándares es que tan solo se dispone de 21 primitivas de servicio o comandos, lo que redunda en un hardware más sencillo y más barato de fabricar.

2.3. Funcionamiento.

Podemos establecer 3 clases de funcionamiento, el que utiliza beacons ,el que no los utiliza y el que utiliza beacons con un tiempo de acceso garantizado (GTS).

2.3.1. IEEE 802.15.4 con beacons

El PAN coordinator se encarga de transmitir beacons cada cierto tiempo, entre beacon y beacon se establece una supertrama compuesta de 16 slots (slots de backoff) y a los 15 que quedan libres los llamamos CAP (Contention Access Period), mediante los cuales los dispositivos podrán transmitir de forma coordinada.

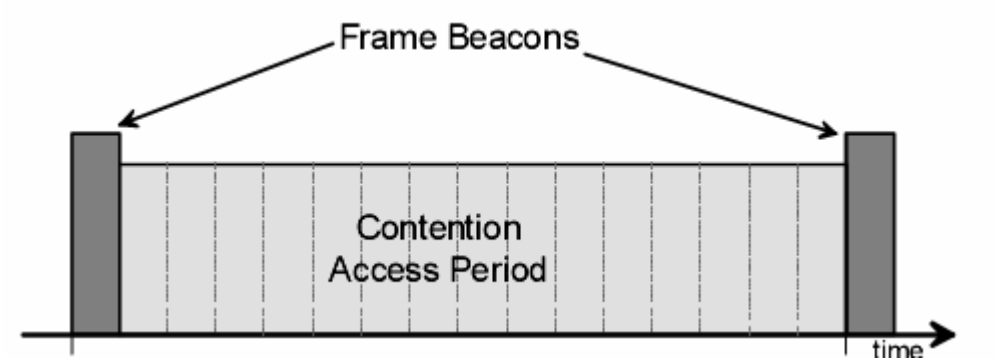


Fig. 2.3 Modo aparente con beacons con periodo de contención de acceso

En este caso, como mecanismo de acceso al medio utilizamos CSMA-CA ranurado en donde las ranuras de backoff están alineadas con el comienzo de un beacon. Cada vez que un dispositivo desea transmitir, primero tiene que alinearse con el siguiente slot de backoff y entonces tiene que esperar un número aleatorio de ranuras de backoff. Si el canal está libre, en el siguiente slot comenzaría a transmitir. Si el canal está ocupado, dejara pasar otro número aleatorio de ranuras de backoff. Los únicos paquetes que no están sometidos a CSMA-CA son los ack's y los beacons.

2.3.2. IEEE 802.15.4 sin beacons

Aquí el mecanismo de acceso al medio es CSMA-CA no ranurado, esto implica que los dispositivos transmiten en el momento que es necesario sin esperar la

pauta (beacons) de un PAN coordinator. Su mecanismo de funcionamiento sería el siguiente: cada vez que un dispositivo desea transmitir datos o comandos MAC, este tiene que esperarse un tiempo aleatorio, si encuentra el canal libre espera un tiempo de backoff, pasado este tiempo intenta transmitir. Si el canal siguiera ocupado después del periodo de backoff volvería a esperar otro tiempo aleatorio y otro de backoff.

Esta es la configuración que nos encontraremos por defecto durante el transcurso de las pruebas que posteriormente realizaremos.

2.3.3. IEEE 802.15.4 con beacons y GTS

Esta tercera modalidad nos proporciona latencia mínima para aquellos dispositivos que necesiten tener este parámetro garantizado, los GTS (Guaranted Time Slot) vendrán definidos en tiempo en la trama de beacon y se sitúan dentro del periodo de libre contención (Contention Free Period (CFP)). Este espacio es reservado para que en caso de haber mucho tráfico ciertos dispositivos tengan siempre prioridad para lograr mínima latencia.

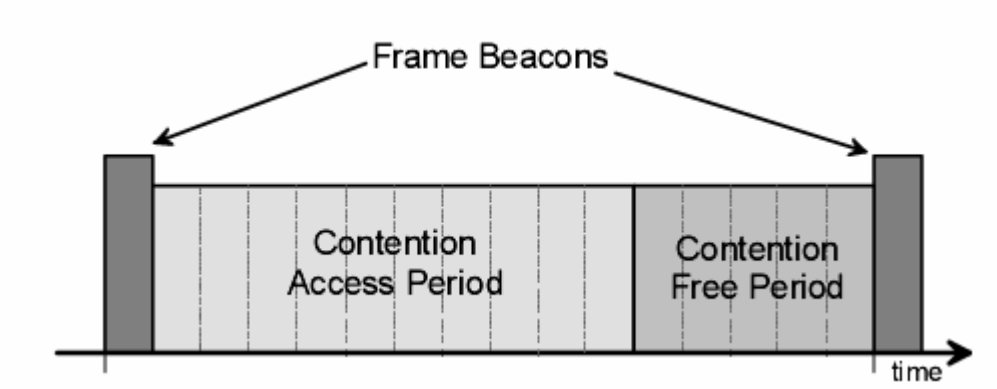


Fig. 2.4 Modo aparente con beacons y zona exclusiva para GTS (los CFP)

2.4. Estructura de trama

2.4.1. Clases de trama

802.15.4 define 4 tipos distintos de trama:

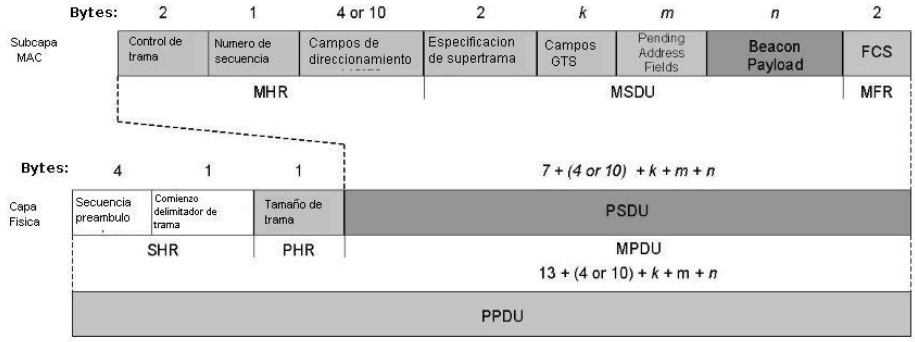


Fig. 2.5 Trama de beacon

Podemos observar diversos campos singulares:

Especificación de supertrama: Este campo especifica diversos parámetros referentes a la supertrama, tales como tiempo de supertrama, si quien transmite el beacon es PAN coordinator o si permite la asociación de nuevos nodos. Nos da información para que el resto de nodos sepan cuando pueden transmitir.

Campos GTS: Aquí se especifica que nodo quiere latencia mínima y slot que ocupará dentro de la trama CFP.

Pending Address Fields: A modo informativo nos dice desde que direcciones se va a transmitir o recibir tramas.

Beacon payload: Este campo se utiliza para llevar información de capas superiores o información encriptada en caso de usar el algoritmo de encriptación AES establecido en el estándar.

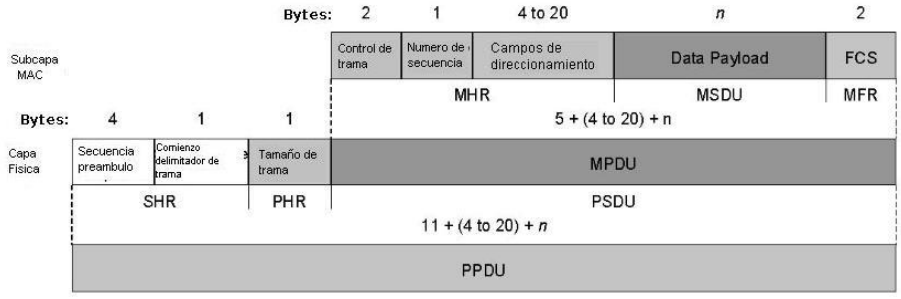


Fig. 2.6 Trama de datos.

Data payload: Únicamente contendrá la información indicada por capas superiores.

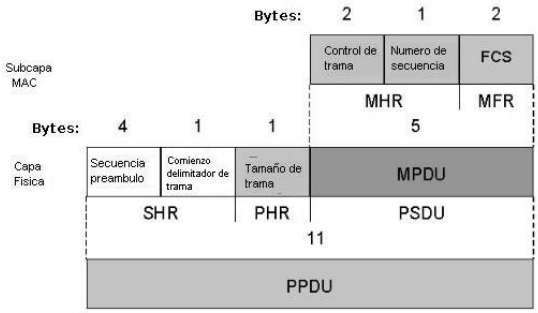


Fig. 2.7 Trama de ack's

La trama de ack's no llevan ningún campo específico, únicamente quien envía una trama de datos o una trama de comandos MAC espera recibir, desde la dirección emisora, una trama ACK para confirmar la llegada.

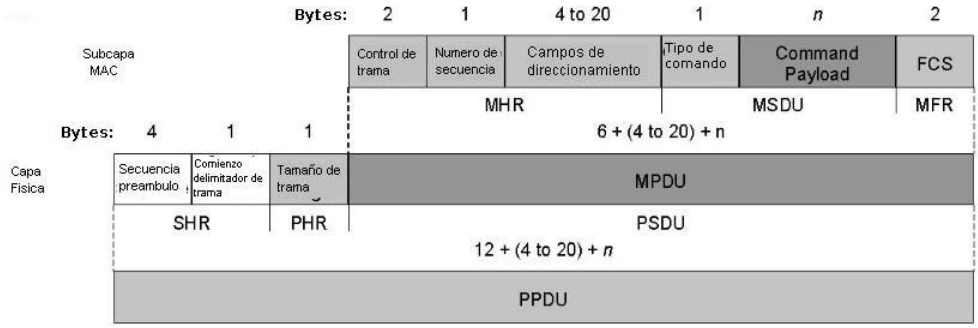


Fig. 2.8 Trama de comandos MAC

La trama de comandos MAC nos permite llevar a cabo todas las funcionalidades primitivas que define el estándar 802.15.4 tales como asociación, desasociación, resincronización con PAN's coordinators, solicitudes de latencia con QoS (GTS), etc...

Campos relevantes:

Tipo de comando: Indica que tipo de comando solicita entre un total de 9 distintos.

Command payload: Su longitud es variable en función del comando utilizado.

2.4.2. Campos comunes en todas las clases de trama:

Como podemos ver, cada una tiene una función distinta muy focalizada, excepto la trama de comandos MAC que puede llevar informaciones muy diversas sobre primitivas de servicio.

En común tienen toda la cabecera de sincronismo (SHR) y cabecera física (PHR), con 3 campos estipulados:

Una secuencia de preámbulo: Secuencia de unos y cero que nos ayuda a determinar el inicio de una trama.

El delimitador de trama : Otra secuencia de unos y ceros que ayuda a diferenciar entre tramas.

Tamaño de trama: Determina el tamaño en bytes de la trama MPDU.

En la cabecera MAC (MHR) encontramos también campos comunes en todos los casos como:

Control de trama: Nos permite escoger entre las 4 clases de tramas descritas.

Número de secuencia: Este campo irá en función del paquete de origen y se incrementará para cada clase de trama que sea enviada.

Campos de direccionamiento: En función de si usamos direcciones cortas (16 bit) o largas (48bits) de origen y de destino.

2.5. Mecanismos de robustez:

Como IEEE 802.15.4, se va a mover en entornos hostiles, se han definido una serie de mecanismos para que así sea más robusto enfrente de estos entornos:

- CSMA-CA: Sistema anteriormente comentado basado en la detección de portadora evitando colisiones. Su esquema de funcionamiento lo hace excelente compartiendo el medio.

- Paquetes con confirmación (ACK): Cuando enviamos paquetes, se nos devuelve un paquete ACK confirmando que el paquete de datos o cualquier otro ha sido recibido correctamente.

- Verificación de los datos (CRC): Mediante un polinomio generador de grado 16 obtenemos la redundancia y podemos comparar el CRC enviado con el calculado en destino y de esta manera verificar los datos.

- Restricciones de consumo: IEEE 802.15.4 esta pensado para aplicaciones que utilicen una batería o una unidad de energía agotable, ya que estas aplicaciones transmitirán información de forma muy esporádica por lo que la cantidad de energía que consume cuando escucha el canal es ultra baja.

- Seguridad: Implementa seguridad de clave simétrica mediante el standard de encriptación AES, el manejo y gestión de la claves es derivado a capas superiores.

2.6. Implementación usada en el proyecto

Para este nuestro proyecto, el transceptor radio que ha sido utilizado, es el modelo CC2420 perteneciente a la casa Chipcon, que cumple con el estándar IEEE 802.15.4.

Este chipset cumple con todas las características mencionadas en los apartados anteriores, destacando que funciona por defecto en el modo IEEE 802.15.4 sin beacons y utiliza el mecanismo de acceso al medio CSMA-CA.

Para consultar alguna de sus parámetros por defecto y la forma de establecer esos cambios consultar anexo K.

CAPÍTULO 3: Entorno de trabajo

En este capítulo veremos el kit de sensores que hemos utilizado, haremos un pequeño resumen teórico de lo que es tinyOS y su lenguaje de programación, también comentaremos alguna de las aplicaciones que hemos realizado para el desarrollo de nuestro proyecto.

3.1. Kit de XBOW

Hemos utilizado un kit de sensores de la casa crosbow proporcionado por i2cat, estos constan de diversos componentes que describiremos de forma breve:

- 8 dispositivos sensores modelo MICAZ con interficie radio 802.15.4.(fig 3.1.)
- 1 placa programadora con interficie por puerto serie DB9 modelo MIB510.(fig 3.1.)
- 1 Placa programadora con interficie ethernet modelo MIB600. (fig 3.1.)
- 4 placas sensoras MTS310 (Sensor magnético, aceleración, luz, temperatura, acústico y un zumbador)
- 3 placas sensoras MTS300(Sensor de luz, temperatura y sonido, y un zumbador)
- 1 placa MDA300 Placa de adquisición de datos con sensores de temperatura y humedad.

Los elementos que aparecen en la figura 3.1, son los que usaremos, el resto no nos serán necesarios para nada.

Fig. 3.1. Material utilizado durante el proyecto.

3.2. TinyOS

TinyOS es un sistema operativo basado en componentes para sistemas empotrados inalámbricos y de libre distribución. Su principal característica es que utiliza muy poca memoria y maneja múltiples tareas y eventos.

La dinámica de los programas realizados bajo TinyOS está conducida por tareas las cuales pueden ser interrumpidas para atender eventos. Los eventos disponen de una prioridad mayor para ser atendidos.

El sistema operativo guarda una cola con las tareas que tiene que realizar (fig 3.2). En caso de que el nodo no tenga ninguna tarea, entrará en modo standby para ahorrar batería.

Muchos de los programas a los que llamamos mediante tareas tienen un evento indicando que su cometido ha sido realizado. Generalmente este evento puede desencadenar otra parte del código. Otros son inicializados y se activan cada vez que se realizan algo para lo que han sido programados (p.ej: La recepción de un paquete).

Muchas operaciones de bajo nivel son finalmente gestionadas por el sistema operativo y así nos permite centrarnos en la aplicación. Se puede entender que TinyOS es como una pirámide en la que a bajo nivel nos encontramos componentes de sistema y en capas superiores encontramos componentes de uso habitual que llaman de forma implícita y encapsulada a componentes del sistema.

Para entender TinyOS, debemos de tener presentes 3 clases de abstracción que son la esencia para su entendimiento y posterior programación:

- Los comandos son las llamadas hacia abajo (fig 3.2). Al llamar a un comando, éste dentro de su componente llama a otros componentes de capas inferiores.

- Los eventos conllevan el efecto inverso, una llamada hacia arriba (fig 3.2). Un componente de bajo nivel avisa a uno de alto nivel de que ha sucedido algo. Si sucede un evento, éste tiene mayor prioridad que cualquier tarea y pasaría a ejecutarse. En caso de haber una interrupción dentro de una interrupción lo más probable es que no sea ejecutada esta hasta que acabara la anterior, siendo esto algo crítico.

- Las tareas son porciones de código que se ejecutan de forma asíncrona siempre y cuando la CPU no tenga que ejecutar ningún evento. Estas se ejecutan por orden de llamada (FIFO) y tal y como antes hemos comentado en caso de que la cola este vacía el procesador entrara en standby hasta que un nuevo evento haga despertar al procesador.

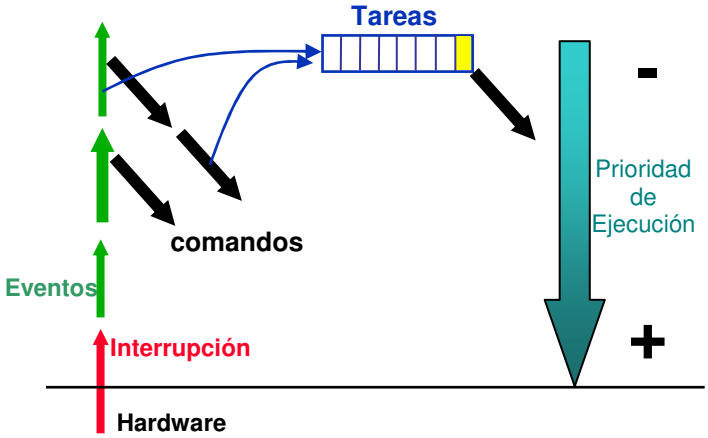


Fig. 3.2 Esquema de funcionamiento

Cuando programamos y compilamos una aplicación, esta va siempre unida de forma implícita al sistema operativo, de forma que no existe el sistema operativo (previamente precargado) y el programa compilado, sino el conjunto que es subido al sistema empotrado cada vez que necesitamos actualizar el programa. Todo y eso su tamaño en Bytes sigue siendo minúsculo.

Para programar mediante TinyOS utilizamos una extensión del lenguaje C llamada nesC, la cual nos permite el uso de interficies que van conectadas a componentes.

Para crear un programa en nesC, únicamente tenemos que escoger los componentes (las partes funcionales) previamente programadas y después "linkarlos" mediante otro fichero de configuración a las interficies que serán utilizadas como código en nuestra futura aplicación.

3.3. NesC

Como hemos comentado antes NesC (Network Embedded Systems C), es una extensión del lenguaje C pero no siempre fue así. Inicialmente y hasta la versión 0.6 de TinyOS, todo el sistema y la programación de aplicaciones fue en C normal. Luego se rescribió todo el código en nesC para la versión 1.0 de TinyOS en adelante.

La adopción de esta extensión contrajo algunas ventajas:

- La aparición de interficies.
- Detección de errores de conexión de componentes a interficies (wiring) en tiempo de compilación.
- Generación automática de documentación referida a funciones.
- Optimización del código utilizado mediante componentes.
- Una mayor limpieza con lo que conseguimos más sencillez.

En nesC cada aplicación está construida a base de componentes enlazados para formar un ejecutable. Los componentes pueden proporcionar o utilizar interficies de otros componentes. La forma de acceder al uso en otro contexto de estos componentes es mediante estas interficies que son bidireccionales es decir mediante eventos y mediante comandos. Con estas interficies podemos acceder a la funcionalidad de los componentes.

En nesC definimos dos clases de componentes: los módulos y las configuraciones. Los módulos contienen el código en sí del componente mediante interficies. Por otra parte las configuraciones enlazan (wiring) las interficies de los componentes que vamos a necesitar en nuestro módulo. Este fichero es crucial porque si no está bien configurado, la aplicación no nos compilará. Tanto módulos como configuraciones utilizan la misma extensión *.nc, la diferencia entre ambos está en la sintaxis que utilizan y es por ello que siguen un código de letras para diferenciar unos de otros. Por ejemplo cuando trabajamos con un programa “ejemplo”, ejemploM.nc seria el módulo y ejemplo.nc seria el fichero configuración.

3.4. Aplicaciones desarrolladas

Todas las aplicaciones que hemos realizado (latencia, ancho de banda, tiempo de gap y pruebas de batería) para micaz han salido a partir de dos aplicaciones denominadas como conjunto Zping.

Aquí explicaremos mediante algorítmica como funciona estas 2 aplicaciones, hay que recordar que estamos delante de un sistema orientado a eventos, así que como veremos de cada programa existe un inicio y luego una serie de eventos que se activaran según sean necesarios.

3.4.1. PrimerizoM.nc

Esta aplicación es la que se encarga de enviar paquetes (fig 3.3.), tiene programado un evento timer.fired que en función del tiempo establecido va enviando paquetes.
Este tiempo lo podremos variar en las pruebas para conseguir distintas cadencias de envió. Como vemos para poder enviar paquetes previamente hay que reservar un espacio de memoria y a continuación efectuar el envío. Entre estas 2 acciones preparamos los datos para ser enviados, entre ellos incrementamos el número de secuencia y adquirimos el valor actual de Tabsoluto (Tabsoluto1).

El evento receive actuara en caso de recibir un nuevo paquete AODV, volverá a adquirir el instante de Tabsoluto actual (Tabsoluto2) y realizara la resta entre tiempos absolutos para obtener un RTT, por ultimo enviara una orden de lectura de voltaje (ADCC.getdata).

El evento ADCC.dataready, se activara cuando tenga una lectura de voltaje válida y la almacenara en la variable voltaje para ser enviada en el próximo paquete.

Por último el evento sendTTL.SendDone no realiza nada relevante pero para que funcione el conjunto tiene que estar.

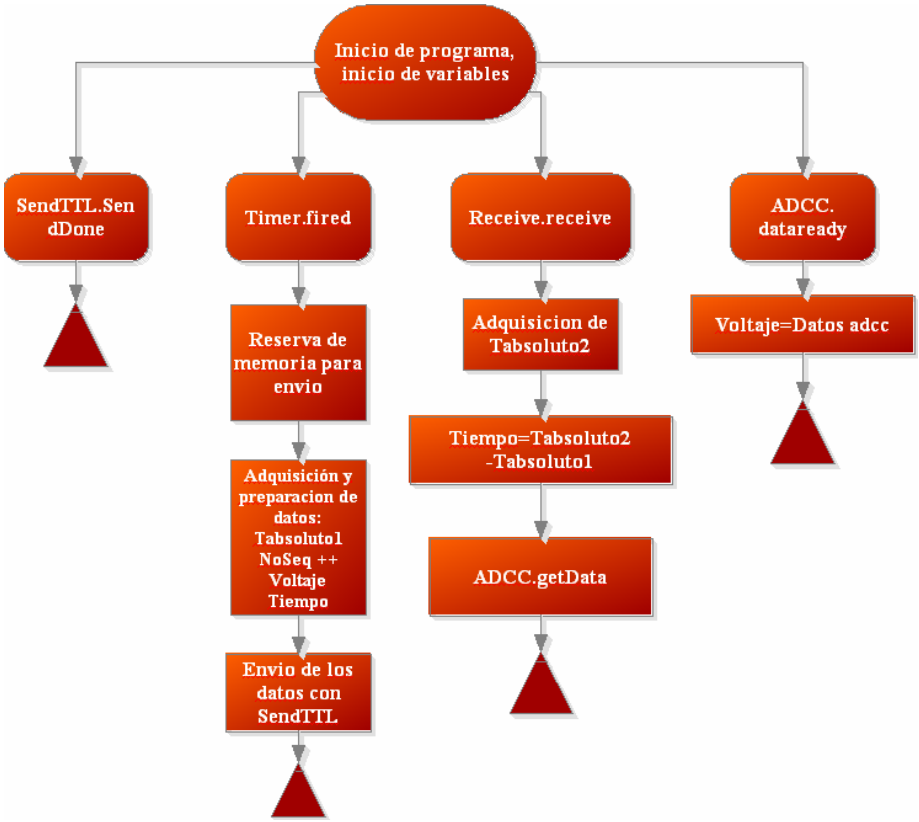


Fig 3.3. Algoritmo de funcionamiento de PrimerizoM

3.4.2. SegundoM.nc

SegundoM es parecido a PrimerizoM (fig 3.4.) pero con menos funcionalidades, mientras que Primerizo envía paquetes de forma periódica, SegundoM, únicamente envía paquetes si a el le llegan paquetes, como veréis definimos el evento receive.receive que tiene la misma funcionalidad que timer.fired en el programa anterior.

En este programa enviaremos en los paquetes de datos, un número de secuencia que se va incrementando con cada nuevo paquete y tiempo absoluto.

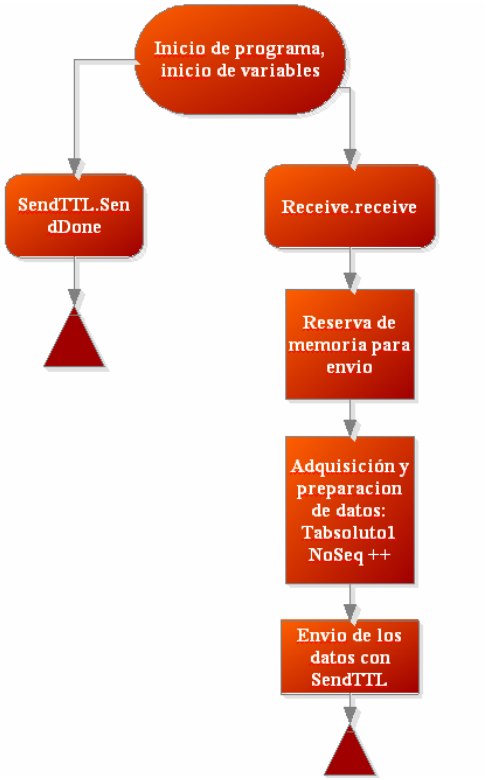


Fig 3.4. Algoritmo de funcionamiento de SegundoM

3.4.3. Interfaz PC

Esta es la aplicación de PC que utilizamos para capturar los datos que nos sirven el conjunto de aplicaciones Zping, mediante serialforwarder (anexo G), para mas información sobre ella consultar el anexo J.

Capítulo 4: El protocolo AODV

4.1. Visión general

El protocolo AODV (Ad-Hoc On-demand Distance Vector) es un protocolo de encaminamiento para redes móviles Ad-Hoc creado por Charles E.Perkins.

Tal y como indica su nombre se trata de un protocolo perteneciente a la familia de los protocolos reactivos, en función del tipo de información que se intercambian los nodos y con la frecuencia con la que lo hacen, los protocolos de encaminamiento en redes Ad-hoc son divididos en protocolos *proactivos* que mantienen sus rutas actualizadas y consistentes en todo momento y calculadas a priori, teniendo como ventaja su baja latencia y como inconveniente una alta sobrecarga de trafico en la red. Los protocolos *reactivos* como el analizado, se basan en calcular la ruta óptima hacia un determinado destino solamente cuando es necesario, estos protocolos, intentan reducir así la sobrecarga generada por los mensajes de actualización de rutas periódicas de los protocolos proactivos, el inconveniente que tienen es el retardo inicial que puede ocasionar el envío de mensajes por una nueva ruta.

Como pequeña reseña y que luego profundizaremos en ella, el estudio que vamos a realizar, está basado en el RFC3561 y nos servirá de base para luego abordar el protocolo empleado en este proyecto que difiere en mucho aspectos con respecto a la aquí analizada.

AODV es un protocolo de carácter minimalista ya que procura sobrecargar poco la red ad-hoc sobre la que esta montado y consume poca memoria en comparación con otros protocolos.

Hay que destacar que casi todas las versiones de AODV funcionan con el protocolo de red IP, sin embargo el caso que más adelante expondremos prescinde de esta capa de red. Nosotros haremos la descripción del estándar incluyendo esta capa IP.

4.2. Descripción del protocolo

4.2.1. Números de secuencia

El objetivo de los números de secuencia es evitar bucles de encaminamiento que pueden afectar a cualquier red pero afectan especialmente a las redes inalámbricas dada su naturaleza de compartición del medio.

Cada nodo dispondrá de su propio número de secuencia, el cual aumentará a cada nuevo paquete que sea recibido.

La utilización de números de secuencia nos permite tener siempre la información de las rutas lo más actualizada posible ya que si nos llega información desactualizada, sería descartada por el simple hecho de tener un número de secuencia inferior.

4.2.2. Descubrimiento de rutas

Cuando queremos transmitir una información entre 2 nodos origen y destino, si no tenemos ninguna ruta previamente establecida, pílale protocolo AODV comienza un proceso de descubrimiento de ruta.

En este procedimiento enviamos un paquete RREQ (*Route Request*) en modo Broadcast (Fig 4.1.) con el objetivo de solicitar una ruta indicando el identificador del nodo que queremos alcanzar, el identificador de nodo origen, el número de saltos inicializado a 0 y su número de secuencia.

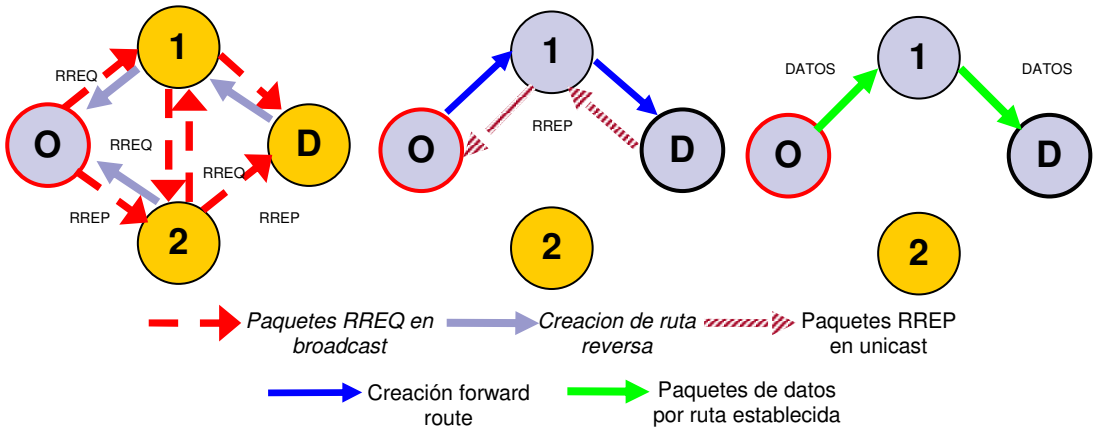


Fig 4.1 Proceso completo de descubrimiento

Si cualquier nodo que no sea destino recibe este paquete, primero comprueba si lo ha recibido previamente en un registro de RREQ recibidos (*rreq_record*) y en caso de no estar registrado, lo vuelve a retransmitir, incrementamos el número de saltos y creamos la ruta reversa (ruta al nodo por el que a llegado este RREQ (fig 4.1.)). Gracias a esto vamos estableciendo nuestra tabla de rutas hacia un destino o varios.

Un nodo tan sólo podrá responder con un paquete RREP (*Route Reply*) para confirmar una ruta en 2 casos: si el nodo es ya el destino o también en el momento en que el nodo que ha recibido un RREQ disponga de ruta hacia destino.

En el momento de responder con un RREP, se envía en modo unicast hacia el vecino que nos había mandado el RREQ de la ruta correspondiente, añadimos la dirección de destino, el número de secuencia de destino así como la dirección del nodo origen y el número de saltos entre origen y destino. Y así sucesivamente hasta conseguir llegar a origen.

Cuando los RREP llegan al siguiente nodo vecino, incrementamos el valor de saltos hop-count del RREP y generamos una ruta de reenvío(fig 4.1.) (forward route) hacia el destino del RREQ y hacia el nodo vecino por el que hemos recibido el RREP. Así también actualizamos la ruta inversa antes establecida mediante RREQ y todos los temporizadores asociados.

Una vez llega un RREP a origen se confirma la ruta. El nodo origen puede recibir diversos RREP de diversos nodos como confirmación de posibles rutas distintas a origen, en este caso el nodo origen tiene 2 criterios para escoger la mejor ruta, por una parte la que tenga el menor número de saltos y por otro el que tenga el número de secuencia más alto.

4.2.3. Mantenimiento y gestión de rutas

AODV se mueve en el medio radio que es un entorno bastante hostil por sus continuas oscilaciones en la calidad del canal y sobretodo la movilidad de los nodos que provoca la rotura de los enlaces. Es por ello que se establecen distintos mecanismos para mantener o eliminar las rutas según su calidad y fiabilidad.

4.2.3.1. Conectividad local entre vecinos

En el estándar se definen 2 métodos:

El primer método consiste en dejar esta tarea a la capa MAC del estándar radio utilizado en ese momento. Mediante confirmaciones (ACK) podemos saber el estado del enlace, en caso de no recibir esta confirmación sabremos que nuestro vecino o no está en condiciones de transmitir o el canal a perdido mucha calidad.

Las confirmaciones se envían de forma automática cada vez que enviamos un paquete de datos, podemos decir que es un método bastante rápido.

El segundo método es el más habitual y se basa en el envío de mensajes HELLO. Estos mensajes son enviados por los nodos que forman parte de una ruta activa para indicar al resto de nodos vecinos su existencia y así mantener la conectividad con ellos.

Los paquetes HELLO son enviados en modo Broadcast, contienen la dirección de su emisor, su número de secuencia, el tiempo de vida del enlace y un TTL de 1. La cadencia de envío de estos paquetes funcional determina el parámetro HELLO_INTERVAL.

La forma de actuar del resto de nodos que reciben estos paquetes HELLO, es buscar en su tabla de rutas si alguna de estas tiene algún nodo vecino (next-

hop) necesario para alcanzar un destino, en caso afirmativo se actualizará el temporizador asociado a esta ruta.

Un nodo invalidará un nodo vecino o la ruta/s activas en el momento que deje de recibir paquetes HELLO en un tiempo superior al producto de las constantes ALLOWED_HELLO_LOSS y HELLO_INTERVAL. ALLOWED HELLO LOSS es el número máximo de paquetes HELLO que se pueden dejar de recibir antes de descartar una ruta activa.

Existe un tercer método comentado en el RFC, el denominado passive ack, que consiste en verificar el estado de las rutas escuchando todas las confirmaciones que provienen de otros nodos a nivel de enlace, de esta forma lograríamos un mayor control de las rutas invalidando de forma inmediata aquellas que no han recibido el ack correspondiente.

4.2.3.2. Paquetes RERR

Los paquetes Route Error forman parte de un mecanismo, que nos permite detectar cuando un enlace radio está roto y por tanto cuando una ruta esta caída en cierta sección.

Cuando esto sucede, el nodo que detecta la rotura invalida todos los destinos que han pasado a ser inalcanzables por culpa de este error en el enlace y seguidamente crea y envía el paquete RERR.

Dependiendo del enlace que se haya roto, puede significar que distintas rutas hacia el nodo destino inalcanzable se hayan visto afectadas, así que según esta naturaleza los paquetes RERR serán enviados en modo unicast o multicast. Este paquete se utilizará para avisar del fallo del enlace y contendrá una lista de los nodos que han pasado a ser inalcanzables.

Bajo el punto de vista del nodo que recibe el paquete RERR, primero mira si el destino del paquete RERR es el next-hop de alguno de los destinos que vienen en el paquete. En caso afirmativo y siempre que el número de secuencia sea mayor al almacenado en tabla de rutas del nodo local, marca la ruta hacia destino como inválida y le suma a su Tiempo de Vida la constante DELETE_PERIOD restante para que expire. Una vez hecho esto vuelve a propagar el paquete RERR repitiendo el proceso mencionado anteriormente hasta llegar a origen.

Si un nodo recibe un RERR anunciando un destino que quiere a continuación alcanzar, tiene que volver a empezar el mecanismo de descubrimiento de ruta del que hemos hablado anteriormente.

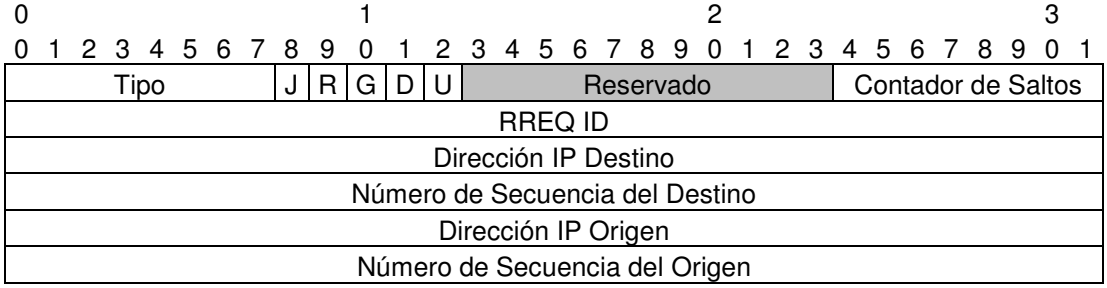
4.2.4. Reinicio de un nodo

Cuando un nodo se desconecta y reconecta por el motivo que sea, perderá el número de secuencia y los números de recuento de las rutas hacia sus destinos, pudiendo provocar bucles en el enlace. Lo que hace el nodo al reiniciar es vaciar la tabla de rutas y evita contestar los RREQ durante un tiempo definido por DELETE_PERIOD, que es tiempo necesario para vaciar la tabla de encaminamiento.

4.3. Paquetes usados por AODV

4.3.1 Paquetes RREQ

El mensaje RREQ se envía para solicitar una ruta hacia un determinado nodo destino. Sigue el formato siguiente:



Tipo – 1

Flags:

J – El flag de Join se usa cuando el nodo origen quiere participar en un grupo multicast.

R – Repair flag. El flag de Reparación se usa cuando un nodo quiere iniciar una reparación de dos partes del árbol Multicast que hayan sido desconectadas.

G – Gratuitous RREP flag. El flag Gratuito indica si se debe enviar un RREP Gratuito en modo Unicast al nodo indicado en el campo *Dirección IP Destino.*

D – Destination Only flag. El flag de Solo Destino indica que tan solo puede responder el destino a un paquete RREQ.

U – Unknow Sequence Flag. El flag de Número de secuencia desconocido indica que no se conoce el número de secuencia del nodo destino.

Reservado – Bits reservados para uso futuro. Si se envía como 0, se ignora en la recepción.

Contador de saltos (Hop Count) – El número de saltos desde el nodo origen al destino.

RREQ ID – Número de secuencia que junto con la dirección IP del nodo origen identifican unívocamente a un RREQ en particular.

Dirección IP Destino – Dirección IP del nodo con el que queremos establecer la comunicación.

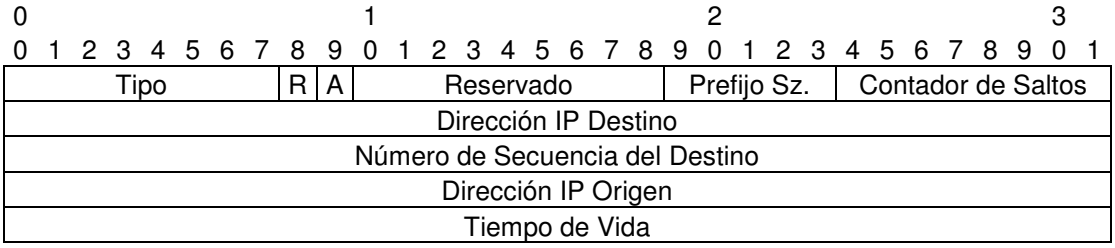
Número de Secuencia del Destino – Último número de secuencia recibido en la fuente de cualquier ruta hacia el destino.

Dirección IP Origen – Dirección IP del nodo que ha originado la petición de ruta.

Número de Secuencia del Origen – Número de secuencia actual del nodo origen.

4.3.2 Paquetes RREP

El mensaje RREP se envía como contestación a un RREQ confirmando un establecimiento de ruta. Sigue el formato siguiente:



Tipo – 2

Flags:

R – Join Flag. El flag de Join se usa cuando el nodo origen quiere participar en un grupo multicast.

A – Acknowledgment Required. El flag de Reconocimiento de Peticiones se activa cuando se quiere comprobar si un enlace es unidireccional.

Reservado – Bits reservados para uso futuro. Si se envía como 0, se ignora en la recepción

Longitud del Prefijo – Si no es cero, especifica que el next-hop indicado puede ser usado, por nodos con el mismo prefijo, como si fuese el destino de la petición.

Contador de Saltos (Hop Count) – Número de saltos desde el nodo origen al destino.

Dirección IP Destino – Dirección IP del nodo para el que se solicita ruta

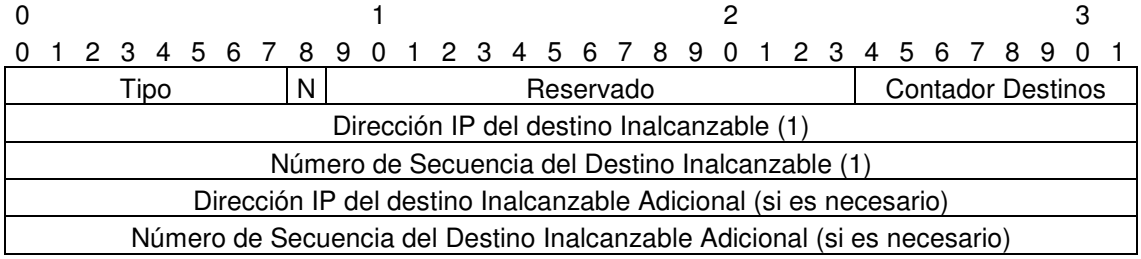
Número de secuencia del Destino – Número de secuencia del destino asociado con la ruta.

Dirección IP Origen – Dirección IP del nodo creador del RREQ y el cual ha solicitado la ruta.

Tiempo de Vida (Lifetime) – Tiempo en milisegundos durante el cual los nodos que han recibido un RREP deben considerar válida la ruta.

4.3.3 Paquetes RERR

El mensaje RERR se envía si un error en el enlace causa que uno o más destinos se vuelvan inalcanzables desde ciertos nodos vecinos.



Tipo – 3

Flag N – El flag de No Borrado (No Delete) se usa cuando se está realizando un Local Repair de un enlace y el nodo no debe borrar la ruta.

Reservado – Bits reservados para uso futuro. Si se envía como 0, se ignora en la recepción

Contador de Destinos – Número de destinos inalcanzables que se encuentran en el mensaje. Como mínimo será 1.

Dirección IP del destino inalcanzable (n) – Dirección IP del destino que ha resultado inalcanzable por culpa del error en el enlace.

Número de secuencia del Destino Inalcanzable (n) – Último número de secuencia conocido, incrementado en 1, del destino que se encuentra en el campo *Dirección IP del Destino Inalcanzable*.

4.4. Que implementación vamos a usar:

Existen muchas implementaciones del protocolo AODV (ver anexo E), aunque no todas ellas funcionan con la capa de red IP, el caso que vamos a estudiar se trata de una variante denominada TinyAODV en su origen y es un AODV pensado para dispositivos empotrados con bajos requerimientos de recursos.

Primeramente intentamos trabajar mediante el *TinyAODV* original, pero dado que no obtuvimos soporte alguno del autor (Jasmeet Chabra) y que no conseguimos que su funcionamiento fuera el adecuado ya que únicamente funcionaba hacia el nodo recolector (sink) y de forma parcial, decidimos dejar esta versión de lado y coger una variante denominada *nstAODV*, que tiene como base el mismo tinyAODV pero con algunas modificaciones y más funcionalidades. En esta variante hemos tenido la suerte de contar con el soporte de su autor Pere Salvatella de la UPC y gracias a ello hemos descubierto bastantes bugs en el código que nos han beneficiado a ambos.

Finalmente y una vez analizado todo su código, hemos llegado a la conclusión que incluye más funcionalidades que tinyAODV.

4.5. La implementación utilizada: nstAODV

Como hemos ido comentando en distintas partes de este documento, nstAODV no corre sobre la capa de red IP pero a cambio se ocupa tanto del nivel de red como del nivel de aplicación. A nivel de red se encarga de la gestión y del mantenimiento de todas las rutas.

Después de analizar todo el código de nstAODV, hemos podido ver que su funcionamiento es mucho más sencillo que todo lo anteriormente explicado, con características que se diferencian al estándar AODV o a sus implementaciones habituales:

- Las rutas no expiran porque no disponen de un temporizador: La única forma de eliminar una ruta es si recibimos algún RERR referente a algún destino en concreto, esta característica la ha heredado de tinyAODV porque este presupone que todos los nodos van a permanecer quietos y por lo tanto las mismas rutas van a ser siempre validas.

- El tiempo de descubrimiento de ruta siempre es estático (TIME_DISCOVERY), no es adaptativo como sucede en otras implementaciones. Aunque tengamos una ruta confirmado siempre tendremos que dejar pasar este tiempo.

- La detección de rutas rotas se realiza a nivel de enlace mediante reconocimientos (ack) a nivel 2. En el momento que dejamos de recibir un reconocimiento a nivel 2, establecemos la ruta como inválida, lo que hace que el siguiente paquete que enviemos establezca un proceso de descubrimiento de ruta desde el mismo, antes de empezar a enviar RERR.

- Existen 3 tablas distintas la tabla de rutas habitual, la tabla de rutas inversas, estas están contempladas en el estándar, y una tercera denominada tabla de broadcast (AODV_Data_Cache), que gestiona los paquetes broadcast de tal manera que no se dupliquen paquetes, esto consiste en que cuando se envía un paquete broadcast y este es reenviado, no se reenvíe más de una vez que caso de llegarle el mismo paquete.

El formato de los paquetes en nstAODV es muy parecido al descrito aquí, diferenciándolo únicamente el formato de las direcciones que para el resto de implementaciones es IP y para nuestro sistema son direcciones de 8 bits.

4.6. Parámetros de nstAODV

Igual que existen unos parámetros por defecto para AODV también existen unos parámetros en nstAODV cuya funcionalidad vamos a comentar :

- *DISCOVERY_PERIOD:* Es el tiempo que deja la pila nstAODV cuando se inicia un procedimiento de descubrimiento, este tiempo suele ser el doble del nominal como mínimo, ya que así está definido en el programa.
- *MAX_INTENTS_GLOBAL*: Número de intentos para poder enviar un mensaje con descubrimiento en caso de no haber ruta antes de enviar paquete RERR.
- *SEND_QUEUE_MAX_RETRIES:* Número máximo de intentos de envío pertenecientes a la cola de envío SimpleQueueM.
- *AODV_MAX_RAND:* Elemento aleatorizador. En función de la probabilidad 1/AODV_MAX_RAND se retrasará la emisión de paquetes RREP entre 0 y 250ms.
- *AODV_MAX_METRIC:* Máximo número de saltos al que puede llegar un paquete AODV y si se rebasa, los paquetes serán eliminados.
- *SEND_QUEUE_SIZE:* Tamaño de la cola de envío de paquetes.
- *QUEUE_SIZE:* Tamaño de la cola AODV.
- *AODVR_NUM_TRIES:* Número de intentos de envío de paquetes RREQ, RREP y RERR dentro de una misma petición de envío de estos paquetes.
- *AODV_RTABLE_SIZE:* Tamaño máximo de la tabla de rutas (AODV_Route_Table) dentro de cada nodo.
- *AODV_RQCACHE_SIZE:* Tamaño máximo de la tabla de RREQ (AODV_Route_Cache).
- *AODV_DATACACHE_SIZE:* Tamaño máximo de la tabla de datos de broadcast (AODV_Data_Cache).

Cuando se empezó a trabajar con nstAODV los parámetros que se encontraron por defecto fueron estos:

Tabla 4.2. Valores por defecto de nstAODV

DISCOVERY_PERIOD	1000
MAX_INTENTS_GLOBAL	4
SEND_QUEUE_MAX_RETRIES	3
AODV_MAX_RAND	3
AODV_MAX_METRIC	10
SEND_QUEUE_SIZE	10
QUEUE_SIZE	10
AODVR_NUM_TRIES	2
AODV_RTABLE_SIZE	5
AODV_RQCACHE_SIZE	5
AODV_DATACACHE_SIZE	5

CAPÍTULO 5: Escenarios implementados:

Para la realización de este proyecto, se han realizado 2 fases muy diferenciadas: en la primera hubo una adaptación de la pila nstAODV a nuestro entorno de trabajo (ver anexo F) y en una segunda fase se han realizado pruebas que seguidamente se van a detallar.

En este capítulo podremos ver todos los escenarios que se han montado así como el funcionamiento de todos ellos. También describiremos todos los parámetros que tiene por defecto el tanto TinyOS, como la pila AODV y la pila radio IEEE 802.15.4.

También podemos consultar el anexo I, en el cual tenemos el programa base a partir del cual hemos desarrollado todos los escenarios.

5.1 Parámetros por defecto en las medidas

Todas las medidas han sido realizadas con una serie de parámetros prefijados y comunes a todos los escenarios, éstos están regulados por IEEE 802.15.4 y AODV y vienen por defecto configurados en nstAODV y TinyOS.

Los parámetros indicados a continuación se encuentran en la librería *cc2420const.h* y están analizados en el anexo K, estos parámetros son fijos (tabla 5.1.) con la excepción de CC2420_DEF_RFPOWER, que en función del escenario puede cambiar.

Tabla 5.1. Valores por defecto del transceptor radio

Parámetro	Valor
CC2420_DEF_RFPOWER	0x01(-25dBm)- 0x1F(0dBm)
CC2420_DEF_CHANNEL	26
CC2420_DEF_PRESET	2405
CC2420_ACK_DELAY	75 (microsegundos)

Otro parámetro importante por defecto, es que ningún nodo actúa como coordinador PAN por defecto, lo que nos lleva a decir que utilizamos un sistema de acceso al medio CSMA-CA sin ranurar, esta función esta en un bit del registro MDMCTRL0

NstAODV también disponen de unos parámetros por defecto, pero estos ya han sido comentados en el capítulo anterior.

5.2 Escenario 1: Pruebas de tiempo de latencia y descubrimiento de ruta:

Estas pruebas han consistido en la creación de diversos escenarios con los nodos en una topología en cadena y una cantidad de nodos de entre 2 y 6, con una cadencia de envío de paquetes de 0.25 pps, tamaño de paquete de 25 bytes a la ida y a la vuelta, potencia de transmisión de -25dBm (3uW) y distancia entre nodos de 7,5 cm.

Para los valores de RTT con descubrimiento se han hecho 15 medidas en una tanda mientras que para RTT sin descubrimiento se han hecho entre 120 y 80 medidas, no se ha hecho el mismo número de medidas por dificultades técnicas. En cada uno de los nodos hemos programado la pila nstAODV con identificadores de red diferenciados y entre sus extremos hemos hecho 2 programas que harían las mismas funciones que un ping dentro de un sistema operativo, al conjunto lo hemos llamado Zping. El objetivo es emplear la aplicación Zping para medir el tiempo de ida y vuelta de un paquete , algo más conocido como RTT (Round Trip Time).

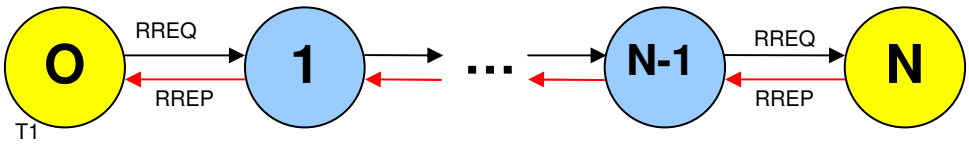


Fig 5.1. Primera referencia (T1) cogida justo antes de proceso de descubrimiento

Cuando nuestro programa envía un paquete en origen primeramente toma una referencia de tiempo absoluto T1, entonces este nodo al ver que no posee ruta hacia el destino solicitado comienza un proceso de descubrimiento de ruta en el que intercambia mensajes de petición de ruta (RREQ) y de confirmación de ruta (RREP). Una vez establecida la ruta, envía el mensaje que hemos intentado enviar antes y una vez llega a destino, otro programa se encarga de devolverlo hacia origen. Cuando el mensajes emitido desde destino, llega de nuevo hasta origen, se toma una segunda referencia de tiempo absoluto T2. Finalmente, el RTT lo obtenemos de resta T2-T1.

Cabe destacar que obtendremos 2 clases de medidas por una parte mediremos un RTT con tiempo de descubrimiento y por otra un RTT sin tiempo de descubrimiento ya que la ruta ya estará establecida.

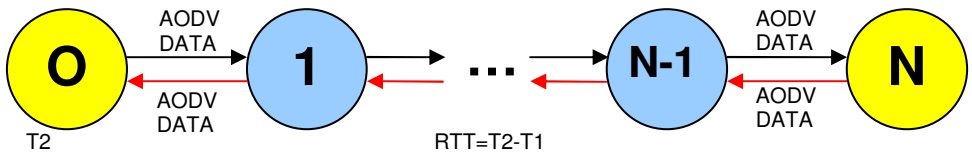


Fig 5.2 Segunda referencia (T2), a la llegada del primer paquete AODV data

5.3 Escenario 2: Pruebas de tiempo de Gap

Esta prueba ha sido realizada con 4 nodos dispuestos como se muestra en las figuras 5.3 y 5.4 y ha consistido en el establecimiento de un ping a 2 saltos entre 3 nodos.

El objetivo es medir la reactividad del protocolo a cambios en la topología de la red es decir, el GAP de conectividad derivado de la rotura de un enlace y el posterior descubrimiento de una ruta alternativa (si esta existe).

Una vez ya se ha establecido una ruta activa entre origen y destino, forzamos la caída del enlace entre O (origen) y 1 quitando el nodo 1 y se añade el nodo 2 como posible ruta alternativa hacia D (Destino) . Mediante un reloj local establecido en origen, hemos determinado el tiempo en el que llega el último paquete hacia destino antes del GAP y el primero después del GAP una vez la ruta a sido restablecida.

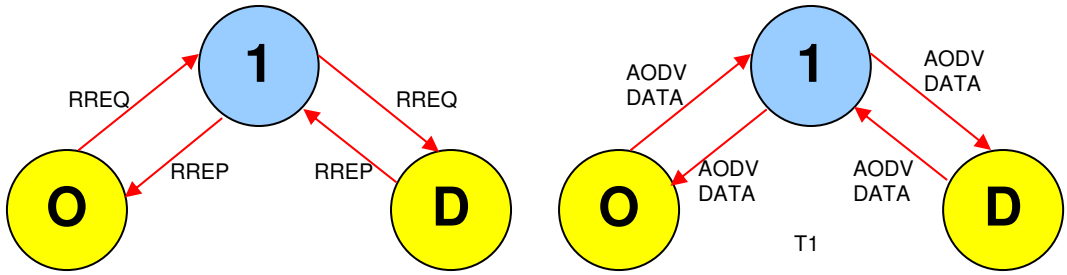


Fig 5.3 Descubrimiento de ruta y envío de datos.

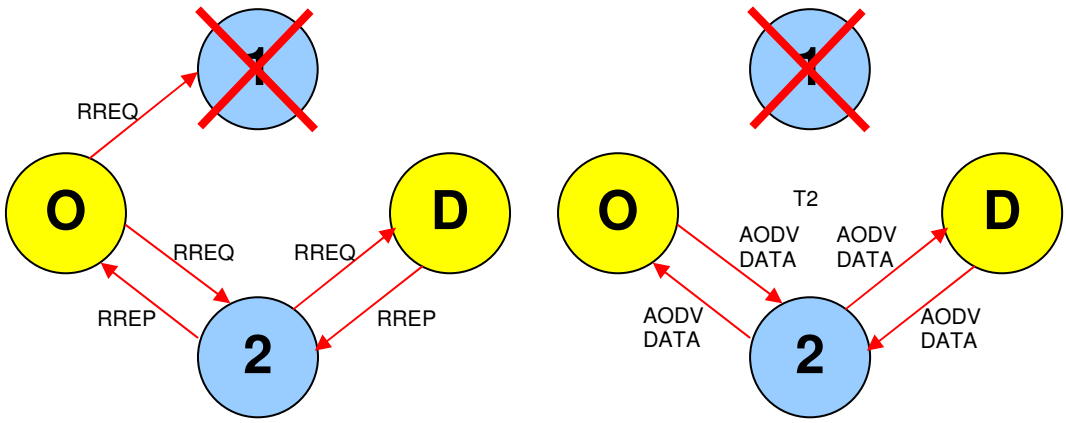


Fig 5.4 Caída forzada del nodo 1, establecimiento de la nueva ruta.

En el momento en que el nodo origen detecta la caída del enlace, reacciona invalidando la ruta actual hacia destino, lanza un nuevo proceso de descubrimiento y a su vez observamos el tiempo absoluto del último paquete de datos llegado antes del cambio de nodo intermedio (instante T1, ver figura 5.3). Una vez establecida la ruta por el camino alternativo, se vuelven a enviar paquetes de datos y tomamos la segunda referencia de tiempo absoluto (instante T2, ver figura 5.4.). Su resta nos dará el tiempo de GAP.

El tiempo de GAP lo hemos medido utilizando el conjunto de aplicaciones Zping pero agregando un generador de números de secuencia para controlar la perdida de paquetes, con una cadencia de envío de 10 pps, tamaño de paquete de 29 bytes a la ida y a la vuelta, potencia de transmisión de -25dBm (3uW) y distancia entre nodos de 7,5 cm.

Con esta configuración y con 15 muestras por prueba hemos realizado 3 pruebas distintas (Fig 5.5.):

-Envío bidireccional mirando tiempo absoluto y número de secuencia desde origen hacia destino, es decir hemos filtrado el paquete que contiene los datos de tiempo absoluto y número de secuencia emitidos por origen para llegar a destino para poder establecer diferencias de tiempo y pérdidas de paquetes.

-Envío bidireccional mirando tiempo absoluto y número de secuencia desde destino hacia origen, en este caso hemos filtrado los datos de tiempo absoluto y número de secuencia, emitidos desde destino por que previamente ha llegado un paquete desde origen y así también veremos las diferencias de tiempo y de pérdida de paquetes.

-Envío unidireccional mirando tiempo absoluto y número de secuencia desde origen hacia destino, para este caso únicamente establecemos la emisión de paquetes de origen a destino sin respuesta por parte de destino y filtramos estos paquetes para poder obtener las diferencias de tiempo y de pérdida de paquetes.

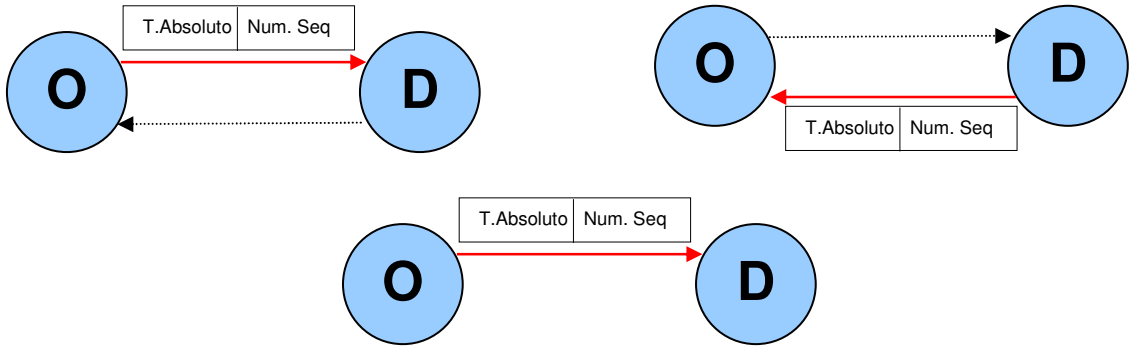


Fig 5.5 Las 3 modalidades con las que ha estado probado el tiempo de Gap

5.4 Escenario 3: Pruebas de ancho de banda:

Se han realizado unas pruebas de ancho de banda y para ello hemos intentado saturar el enlace de alguna forma. Para conseguirlo, hemos recurrido a dos métodos distintos: por un lado hemos aumentado el tamaño de los paquetes y por otro hemos variado la cadencia de envío.

El tamaño máximo de los datos de paquete nstAODV esta limitado a longitud de un byte (0..255), aunque estructuralmente el límite viene dado por el buffer hardware del chipset radio (CC2420) ya que disponemos de una FIFO de tan solo 128 bytes. Después de hacer un estudio de lo que lleva un paquete de tamaño máximo, quedaría de esta forma:

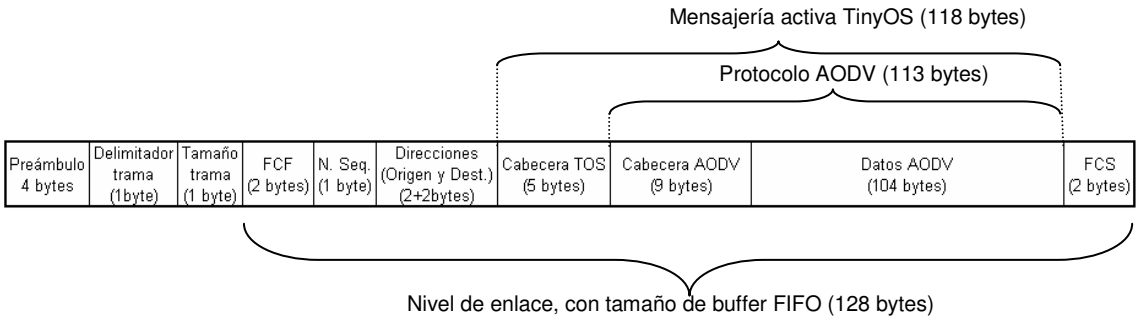


Fig 5.6 Formato de una trama completa desglosada.

Como conclusión podemos ver que el tamaño de un paquete de datos queda bastante reducido, tan sólo 104 bytes, aunque si tenemos en cuenta las aplicaciones que posiblemente están a su abasto se revelan como suficientes.

La cadencia máxima la hemos determinado haciendo pruebas y para ello hemos implementado un generador de números de secuencia que nos permite

observar los paquetes que no han sido entregados y por tanto se han perdido. Entendemos como cadencia máxima el momento en que no perdemos ningún paquete. Como veremos en la tabla de pruebas establecemos distintas cadencias asignando una probabilidad de éxito (5.1).

$$\Pr obabilidad\_de\_éxito = Paquetes\_recibidos \Big/ Paquetes\_enviados \qquad \textbf{(5.1)}$$

Las pruebas las hemos realizado de 1 a 3 saltos con una topología en cadena para poder ver la influencia entre el tiempo de proceso y el ancho de banda y enviando paquetes de tamaño máximo de 104 bytes de forma unidireccional (Fig 5.7.), con una cadencia de variable que oscila entre los 50 pps y los 28,6 pps, potencia de transmisión de -25dBm y una distancia entre nodos de 7.5 cm.

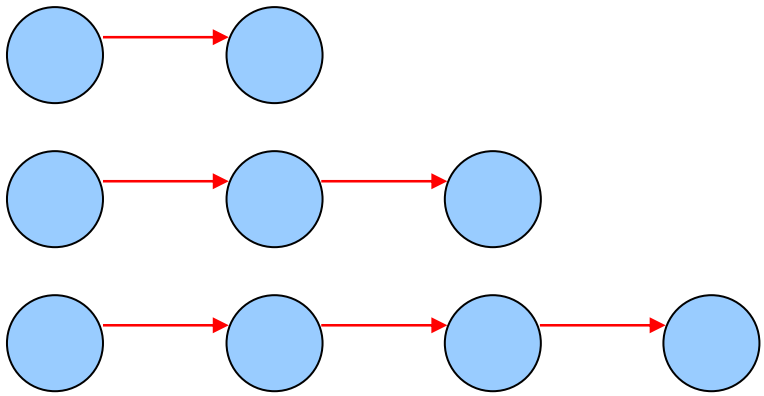


Fig 5.7 Envíos unidireccionales de 1 a 3 saltos.

5.5 Escenario 4: Pruebas de batería

Para realizar las pruebas de autonomía hemos cogido un juego de baterías recargables.

Somos conscientes de que las baterías recargables no son fiables en caso de que se haga un uso inconstante, es por ello que hemos basado las pruebas en un único juego de estas baterías las cuales han sido sometidas a un continuo proceso de carga y descarga, es decir han sido descargadas y cargadas completamente al inicio de cada prueba, cosa que hace que las medidas sean bastante fiables.

El uso de un juego de baterías recargables nos va a servir para poder determinar el consumo medio en mA, ya que a priori sabemos la capacidad de estas (1300mAh), en caso de utilizar unas pilas alcalinas no dispondríamos de este dato, aparte que la duración de una batería depende mucho de la marca de pilas y su capacidad es tratada prácticamente como un secreto industrial.

Primero hemos hecho pruebas de 2 horas aproximadamente para determinar la caída de tensión que se da, pero dado que esto puede llevar al engaño y ya que no es una prueba irrefutable, decidimos realizar una segunda prueba de estrés que pusiera a prueba la autonomía de los dispositivos sensores.

Estas 2 pruebas las hemos realizado en dos modalidades distintas: una a potencia máxima (0dBm), para la cual únicamente hemos utilizado 2 nodos (1 salto)y otra a potencia mínima (-25dBm) en la que hemos utilizado 3 nodos (2 saltos), también se podría haber hecho esta prueba a potencia máxima sin ningún problema siempre que hubiéramos separado los 3 nodos los suficiente como para que no se vieran los extremos (Todos equidistantes 10 metros).

También se ha variado el tamaño de paquete para ver su influencia en el gasto de batería haciendo oscilar su tamaño entre 37 bytes y 128 bytes..

Así las pruebas han consistido en lo siguiente:

-2 pruebas de desgaste con duración de 2 horas cada una entre 2 nodos (1 salto) en una topología en cadena, con cadencia de envío de 0.25 pps, tamaño de paquete de 37 bytes , potencia mínima (-25dBm) y máxima (0dBm) y distancia entre nodos de 7.5 cm, en estas pruebas y en ambas hemos filtrado el paquete que lleva la información de origen a destino para sacar el muestras de tensión cada 4 segundos.

-2 pruebas de desgaste indefinido entre 3 nodos (2 saltos) en una topología en cadena, con cadencia de envío de 10 pps, tamaño de paquete de 37 bytes, potencia mínima (-25dBm) y distancia entre nodos de 7.5 cm, dado que no hay que filtrar ningún paquete para obtener información, únicamente hemos tenido que coger un cronómetro y medir el tiempo que ha necesitado el dispositivo sensor para dejar de ser operativo, la diferencia entre estas 2 pruebas radican en su posición, en una de las pruebas el tiempo se mide en el nodo extremo y en la otra en el nodo intermedio, al cual se le presupone una mayor carga de trabajo.

-2 pruebas de desgaste indefinido entre 2 nodos (1 salto) en una topología en cadena, con cadencia de envío de 10 pps, tamaño de paquete de 37 bytes en una prueba y de 128bytes en otra, potencia mínima (-25dBm) y distancia entre nodos de 7.5 cm, igual que en caso anterior únicamente será necesario tener un cronómetro para poder medir el tiempo de autonomía.

Capítulo 6: Resultados obtenidos y conclusiones

6.1. Introducción

En este capítulo expondremos los resultados obtenidos en los escenarios que han sido descritos en el capítulo anterior así como la justificación de los mismos.

6.2. Tiempos de latencia y descubrimiento de ruta:

En estas gráficas elaboradas con el escenario 1 del capitulo 5, podemos comparar las latencias existentes con descubrimiento y sin descubrimiento (Fig 6.1. y Fig 6.2.).

Para cualquier tiempo de latencia con descubrimiento (RTT_DR) puede ser caracterizado con estas 2 formulas, en caso de un solo intento (6.1) y en caso de más de un intento, en el utilizaremos esta formula (6.2).

$$RTT\_DR=(1+No\_Int) \times DISCOVERY\_PERIOD + RTT \quad \textbf{(6.1)}$$

$$RTT\_DR = No\_Int\frac{(No\_Int+3)}{2}\times DISCOVERY\_PERIOD + RTT \quad \textbf{(6.2)}$$

RTT_DR →Tiempo de latencia con descubrimiento
RTT → Tiempo de latencia sin descubrimiento
No_Int → El número de intentos necesarios para establecer una ruta

Para Zping de 2 a 6 nodos con descubrimiento (fig 6.1) podemos observar que cumple esta premisa tanto para tiempos de descubrimiento de 1000 ms como para 500 ms cumplen con la primera ecuación lo que significa que únicamente a sido necesario un intento, sin embargo para tiempos de descubrimiento de 100ms no se cumple, todas las muestras tomadas cumplen con la ecuación (6.2) aunque la media que se refleja en la gráfica no cumple con esta ecuación.

Para Zping de 2 a 6 nodos sin descubrimiento (fig 6.2) podemos observar un crecimiento lineal de 12 ms aproximadamente a cada nuevo salto que añadimos, esto es debido al tiempo de proceso, este mismo crecimiento también se da en Zping de 2 a 6 nodos con descubrimiento (fig 6.1) pero se nota porque esta un orden de magnitud por debajo de los tiempo de descubrimiento.

Aparte podemos observar las desviaciones medias derivadas de las pruebas Zping (tabla 6.1 y tabla 6.2).

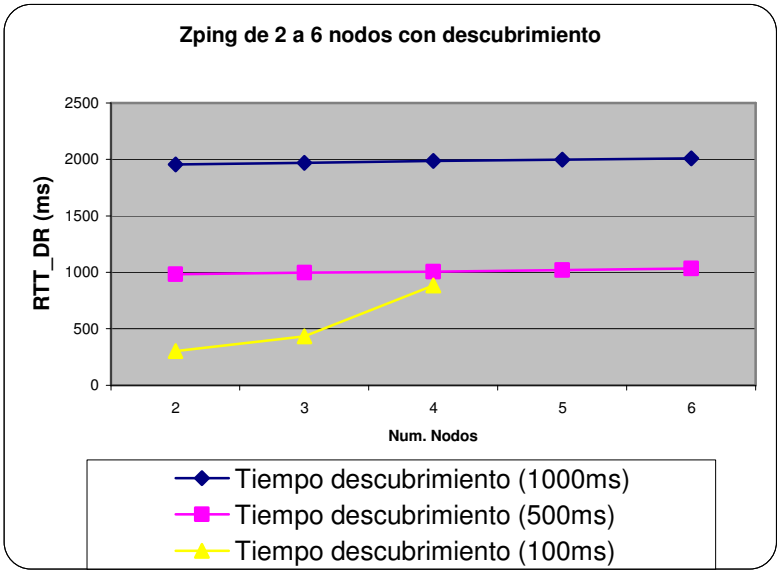


Fig 6.1 Zping de 2 a 6 nodos con descubrimiento

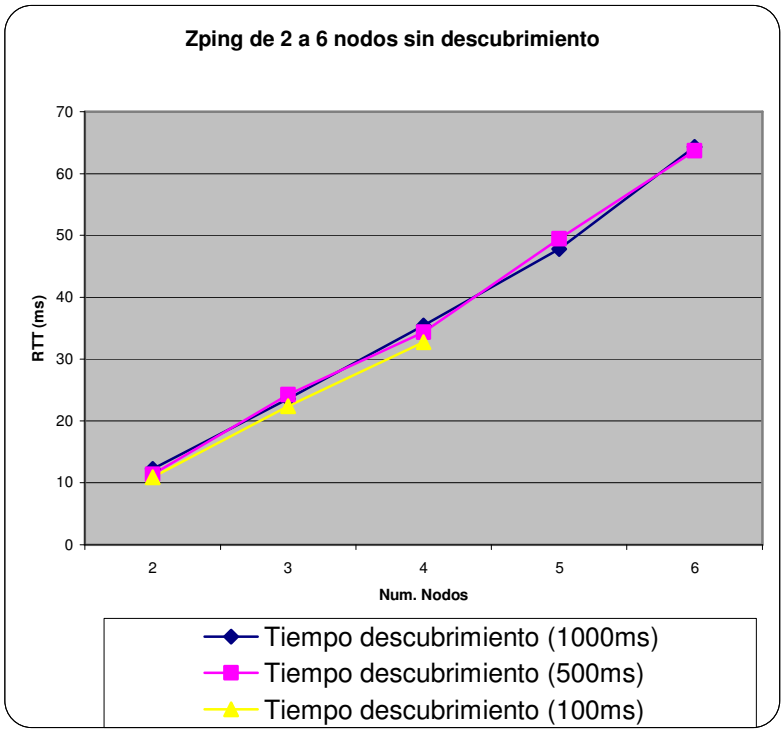


Fig 6.2 Zping de 2 a 6 nodos sin descubrimiento

Para tiempo de descubrimiento de 100ms podemos ver que las cosas han cambiado (fig 6.1) y no se cumple el mismo comportamiento que para 500 y para 1000.

Tabla 6.1. Desviación media de Zping con descubrimiento en ms

NODOS/ DISCOVERY_PERIOD	2	3	4	5	6
1000 ms	0,05722488	2,415998855	1,771615815	3,336476842	10,44917541
500 ms	0,13200944	3,425252277	2,8953256	3,1569329	9,523987626
100 ms	200,2031768	324,7165738	640,8681578	-	-

Tabla 6.2. Desviación media Zping sin descubrimiento en ms

NODOS/ DISCOVERY_PERIOD	2	3	4	5	6
1000 ms	2,927207083	5,545541074	5,450247384	6,943953786	16,74068999
500 ms	2,747262622	3,35672726	3,626747272	7,946928202	16,62347243
100 ms	3,098224245	18,27860254	9,469885584	-	-

La causa es que con un tiempo máximo de descubrimiento de 200ms muchas veces las rutas no se establecen a la primera. Vemos que a medida que vamos agregando nuevos nodos, las latencias se incrementan, esto es debido al mayor número de intentos necesarios ya que a cada nuevo intento el tiempo de descubrimiento se va incrementando.

La razón que hay para realizar nuevos intentos de descubrimientos es que el tiempo de descubrimiento ha expirado antes de obtener la ruta. Esto puede ser debido a varios factores, entre los más importantes es que existe un factor aleatorio de acceso al medio que puede retrasar la entrada al medio de un RREQ en 250ms, tiempo ya de por sí que es superior a los 200ms, este efecto se ve agravado al agregar más nodos. Este hecho lo podemos ver en las siguientes graficas de intentos especialmente hechas para un tiempo de descubrimiento de 100ms (Fig 6.3. Fig 6.4. y Fig 6.5.).

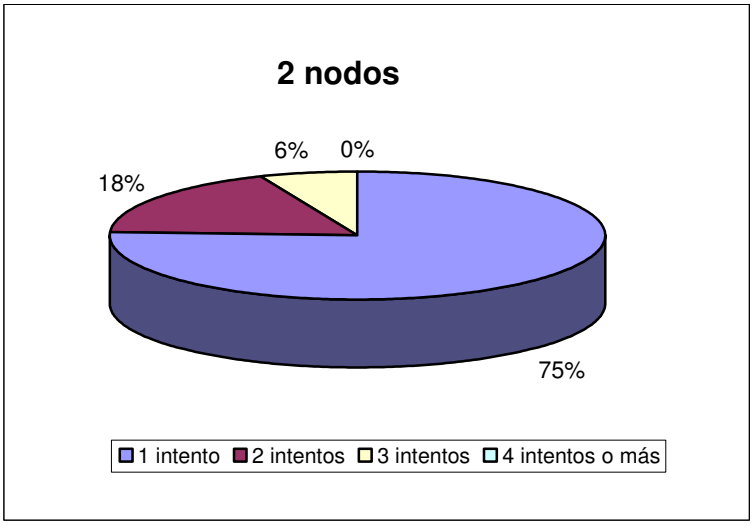


Fig 6.3 Porcentaje de intentos de petición de ruta a 2 nodos

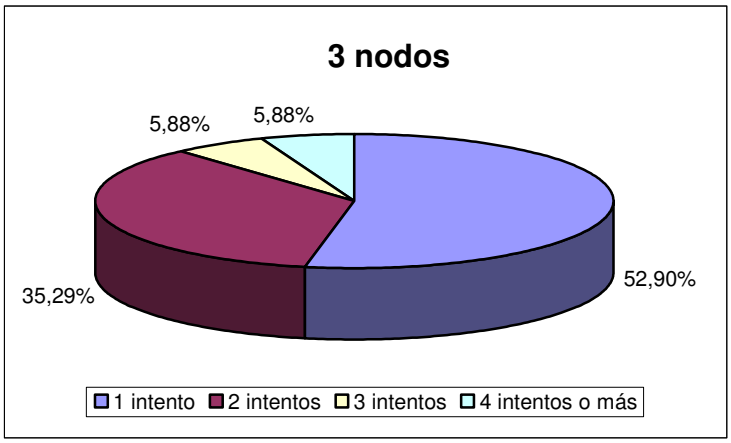


Fig 6.4 Porcentaje de intentos de petición de ruta a 3 nodos

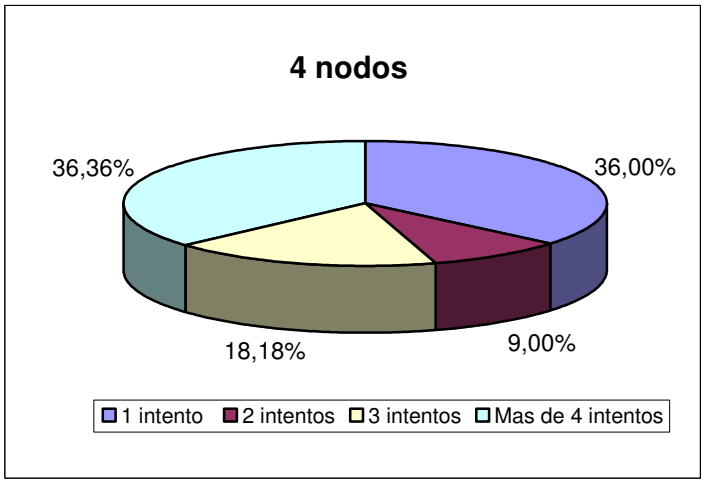


Fig 6.5 Porcentaje de intentos de petición de ruta a 4 nodos

6.3. Tiempo de GAP

Para el tiempo de GAP de conectividad que esta relacionado con el escenario 2 del capítulo 5 (Fig 6.6), podemos ver una relación directa entre el tiempo de descubrimiento (TIME_DISCOVERY) y el tiempo que tarda en restablecerse una ruta, cabe destacar las diferencias entre envíos bidireccionales y unidireccionales, vemos que en envíos bidireccionales desde el origen no existe un incremento lineal del tiempo como sucede en el caso anterior, es probable que al ser el escenario con una mayor carga de trabajo, si agravamos el tiempo de descubrimiento se acumule más tareas y se tarde más tiempo, hay que recordar que en este caso el dispositivo sensor aparte de enviar tareas tiene que recibir paquetes cosa que conlleva un gasto de proceso.

Es importante leer el anexo A para entender este apartado ya que explicamos el funcionamiento completo de todo el componente AODV.

¿Qué sucedería si perdemos una ruta hacia destino?

Supongamos la situación de nuestro escenario en que enviamos paquetes con una cadencia de 10pps y tenemos una ruta ya establecida.

En el momento de la caída del enlace, primero pasaría un tiempo entre 0 y 100ms hasta el envío del siguiente paquete y como aún el nodo origen no habrá detectado que la ruta está rota, realizará el envío mediante el componente que hace las funciones de cola y envío (*SimpleQueueM*). Este envío se intentará el número de veces que indica la variable SEND_QUEUE_MAX_RETRIES sin consumir tiempo significativo. Debido a que el nodo origen no recibe ninguna confirmación (ack), en ese momento detecta que ha caído el enlace, el mensaje es devuelto por *SimpleQueueM* ya que no ha conseguido realizar el envío. Seguidamente invalida la ruta con el componente AODV_TablesM y almacena el mensaje en la cola específica de AODV (*BufferQueueM*).

Ahora nos tendremos que esperar otros 100ms hasta que sea ordenado otro envío. El nodo origen puede ver que no dispone de ninguna ruta hacia destino. Entonces el paquete de datos se almacenará en la cola AODV y se iniciará el proceso de descubrimiento de ruta así como el temporizador correspondiente (TIME_DISCOVERY). Durante este tiempo, todos los mensajes que lleguen serán puestos en la cola AODV hasta que finalice el tiempo de descubrimiento.

En caso de haber establecido correctamente la ruta, todos los mensajes almacenados en la cola AODV serán enviados hacia destino. En caso de no poderse establecer la nueva ruta, se volverá a empezar otro proceso de descubrimiento de ruta con un tiempo superior equivalente al tiempo inicial de descubrimiento de ruta multiplicado por el número de intentos de descubrimiento más uno.

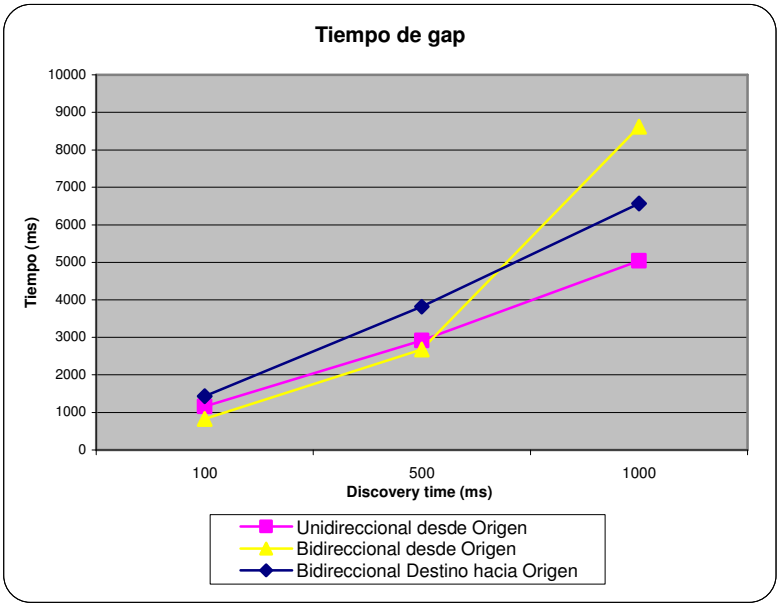


Fig 6.6 Tiempo entre gaps

En la gráfica de la figura 6.7 podemos ver una relación de paquetes perdidos respecto a cada tipo de descubrimiento de rutas. Reflexionando el porqué de estas pérdidas, inicialmente pensamos que se trataba de una saturación de la cola AODV pero posteriormente, analizando el instante en el que se perdían los paquetes, se ha llegado a la conclusión de que la culpa era de la cola de tareas. Seguramente en el instante en que se pierde la conectividad, se producen un número elevado de tareas que hace que la cola de tareas del sistema operativo se desborde y aunque se posteen las tareas de envío, éstas no son atendidas y se pierden. Otra cosa que podemos observar es que tanto el número de paquetes que se han perdido cuando el envío de paquetes es de destino hacia origen de forma bidireccional es muy inferior a las otras medidas, es probable que una menor llegada de paquetes en sentido contrario al haberse perdido implique un número menor de tareas para el nodo destino, por lo que el efecto de perdida de paquetes se ve frenado.

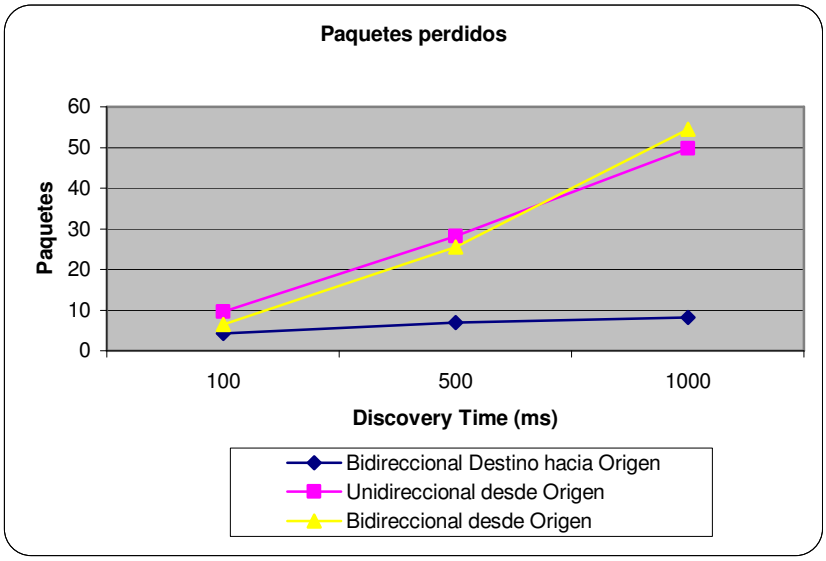


Fig 6.7 Paquetes perdidos entre gaps.

6.4. Pruebas de ancho de banda.

Una vez hechas las pruebas de ancho de banda (Fig 6.8.) que corresponde al escenario 3 del capitulo 5, hemos podido determinar que la velocidad máxima de envío sin que se pierdan paquetes, se da a 34,5 paquetes por segundo para 2 nodos, lo que nos da una velocidad a nivel 3 de 32,568 kbps y una velocidad de total de 37,536 kbps. Esto implica que como máximo IEEE 802.15.4 sobre TinyOS únicamente ha sido capaz de trabajar al 15% de su capacidad.

Trabajando con rutas multi-salto de 3 y de 4 nodos, hemos tenido que rebajar la velocidad a 30 paquetes por segundo para conseguir que el 98% de los paquetes enviados llegaran a su destino por lo que la velocidad de datos AODV ha quedado establecida en 23.8Kbps y la velocidad total en 31,11Kbps, un 12,44% de la capacidad del enlace.

Ateniéndonos a estos resultados, podemos decir que posiblemente el problema que tienen nuestros dispositivos sensores es de proceso, bien seguro que si consiguiéramos aumentar la capacidad de proceso podríamos aprovechar un mayor ancho de banda que nos ofrece IEEE 802.15.4

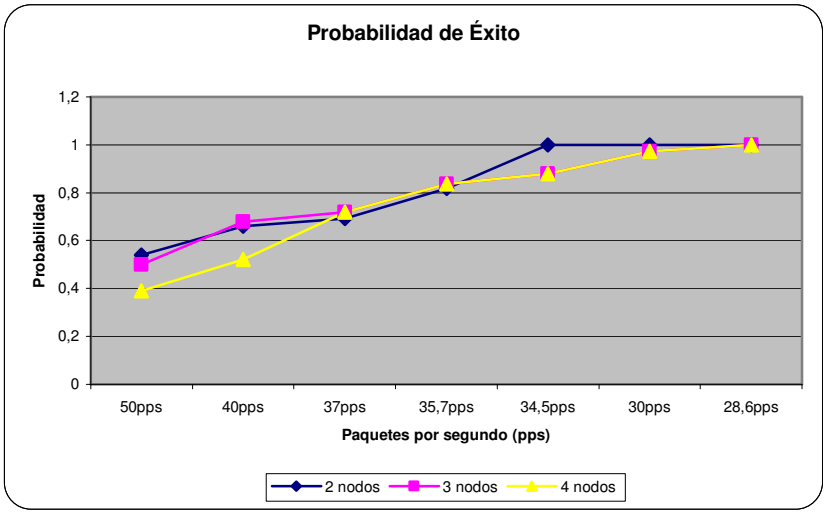


Fig 6.8 Ancho de banda en pps respecto a probabilidad de recepción

6.5. Pruebas de batería:

Estas pruebas de batería se corresponde con el escenario 4 del capitulo 5.

En estas pruebas basadas en tiempo limitado (Fig 6.9) no vemos una caída lineal del voltaje, para 0dBm existe un caída de tensión de 104mV mientras que para -25dBm la caída de tensión es más suave 41mV.

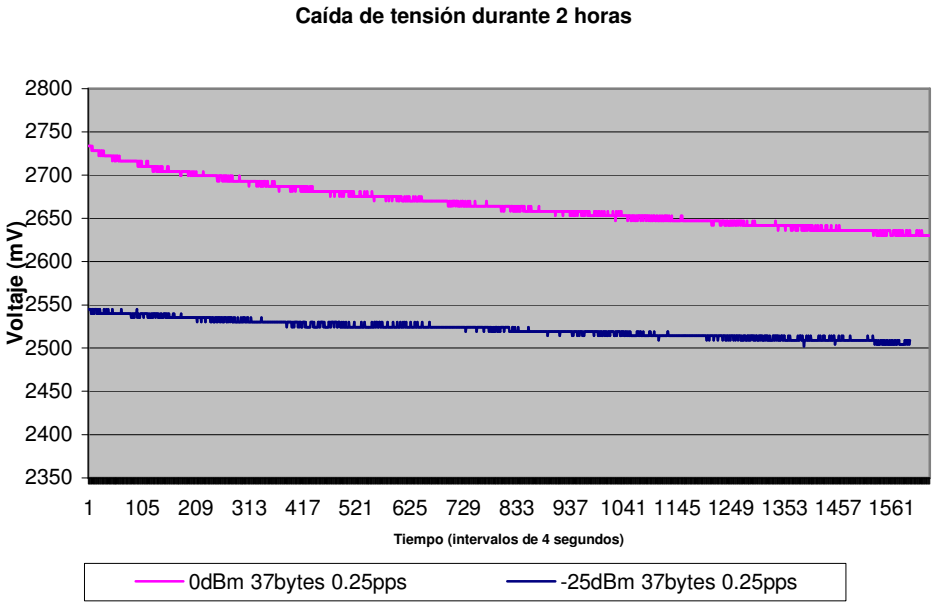


Fig 6.9 Caída de tensión durante 2 horas.

Tabla 6.3 Prueba de estrés.

Posición	Batería	Duración	Tamaño paquete en bytes	Cadencia	Potencia	Consumo medio
(O)-O-O	1300mAh	51:45:00	37	10pps	25dBm(3uW)	25,16mA
O-(O)-O	1300mAh	43:10:00	37	10pps	25dBm(3uW)	30,12mA
(O)-O	1300mAh	51:30:00	37	10pps	0dBm(1mW)	25,24mA
(O)-O	1300mAh	51:10:00	128	10pps	0dBm(1mW)	25,4mA

En las pruebas de estrés es donde pudimos ver que la caída de tensión se estabilizaba en 2.4 voltios y cuando nos acercábamos al final se pudo observar que la caída de tensión volvía a ser cuantiosa, dejando el dispositivo sensor de ser operativo cuando llegamos a los 1.8 voltios.

En las pruebas a 3 nodos vemos que quien ha realizado un mayor esfuerzo es el nodo intermedio ya que es el que debe recibir y enviar información en ambos sentidos. Podemos ver que los nodos laterales han durado una media de 51 horas y 45 minutos en el mejor de los casos, disminuyendo sensiblemente la capacidad en el nodo intermedio en el que vemos que la duración baja a 43 horas y 10 minutos.

Otra cosa que podemos observar, es el nulo efecto producido en el desgaste de las pilas al usar la potencia mínima y la máxima. En todos los nodos laterales, que para 2 y 3 nodos hacen exactamente la misma función, el consumo medio es muy similar para todos habiendo lógicas variaciones en función de la potencia utilizada pero con muy poca variabilidad.

La observación del consumo medio (tabla 6.4) nos lleva a la conclusión de que el gasto de energía del microcontrolador atmel mega 128 es menor que el del transceptor radio. Viendo estos consumos vemos que el uC atmel mega 128 consume 8mA, mientras que la interfaz radio consume el resto de corriente, si miramos los resultados podemos deducir que el transceptor radio se pasa gran parte del tiempo en modo recepción de esta forma podemos justificar que todos los nodos laterales consumen por igual, sin embargo el nodo intermedio tiene un mayor consumo al tener que transmitir un mayor número de veces que los nodos laterales.

Tabla 6.4 Consumos de las partes elementales de MICAZ

Microcontrolador Atmel mega 128	
Consumo de corriente(activo)	8mA
Consumo de corriente (inactivo)	<15 uA
Transceptor radio CC2420	
Consumo de corriente(modo recepción)	19,7mA
Consumo de corriente(modo transmisión a 0dBm)	17,4mA
Consumo de corriente(modo transmisión a -25dBm)	<11mA
Consumo de corriente(modo reposo)	1uA

CONCLUSIONES

Cada día que pasa existe un mayor número de redes de sensores ya que dentro de las redes ad-hoc estas son las que se les está dando un mayor número de usos civiles tal y con destacamos en el capítulo 1, esta progresión ira a más durante los próximos años.

Es por esto que es bastante importante, poder caracterizar el funcionamiento de estas redes como se ha hecho en el presente proyecto, otra motivación ha sido la escasez de documentación y pruebas con sensores reales bajo la circunstancias descritas en el proyecto. Las pruebas que hemos realizado son relevantes, ya que no hemos encontrado hasta la fecha ningún test de esta índole.

Hemos podido constatar que nstAODV es una versión que se separa del estándar en muchos aspectos, la forma de trabajar de nstAODV es distinta a otras implementaciones ya que el sistema operativo que lo soporta está orientado a eventos, lo que hace que tenga momentos libres en los que puede entrar en standby y ahorrar batería. El resto de implementaciones AODV necesitan tener un proceso abierto de forma constante para el control del protocolo. Otra cosa que prima en este AODV es la sencillez, por poner un ejemplo, el número de temporizadores es muy pequeño, sin embargo en el resto de implementaciones existen gran cantidad de temporizadores.

También hemos podido constatar el funcionamiento de IEEE 802.15.4, y podemos decir que funcionaba en modo Peer to Peer, mediante el mecanismo CSMA-CA.

Hemos tenido que desarrollar aplicaciones mediante el lenguaje nesC propio de tinyOS y mediante el lenguaje C# para el filtrado de paquetes, estas nos han ayudado a desarrollar el conjunto de pruebas que aquí han sido relatadas.

Con respecto a las pruebas, hemos podido comprobar que a medida que hemos ido agregando nuevos nodos, el tiempo de latencia ha subido a causa del tiempo de proceso y del tiempo de descubrimiento que se haya definido. También hemos visto que por causa del sistema operativo TinyOS y la baja capacidad de proceso de los dispositivos sensores, el ancho de banda de ve afectado con lo que solo conseguimos aprovechar hasta un 15 % de la capacidad que nos ofrece IEEE 802.15.4.

Otro aspecto es la relación directa entre el tiempo de GAP y el tiempo de descubrimiento definido, con una pérdida de paquetes causada por el propio sistema operativo TinyOS que se ve incapaz de atender todas las tareas en el momento que se detecta la rotura de una ruta ya que la cola de tareas se desborda.

Por último las pruebas de autonomía realizadas, nos revelan que podrían aguantar más de 2 días seguidos con un uso intensivo aunque siempre limitado

por la capacidad de proceso y el sistema operativo. También analizando los consumos proporcionados en los datasheet, podemos afirmar que gran parte del consumo medio lo realiza el transceptor radio cuando esta en modo recepción y transmisión. Así estamos en condiciones de afirmar que para una aplicación sensora que active sus sensores de forma esporádica la autonomía será larga ya que no habrá un uso intensivo ni de la cpu ni del transceptor radio IEEE 802.15.4.

Respecto a las líneas futuras, cabe la posibilidad de optimizar el protocolo nstAODV optimizando los tiempos de descubrimiento de nuevas rutas, que como sabemos son tiempos estáticos, lo que hace que se pierda mucho tiempo para esta tarea. Si se realizasen procesos de descubrimiento dinámicos, es decir que una vez una ruta ha sido establecida, a continuación pudiéramos enviar el paquete de datos correspondiente a la ruta demandada. Con este cambio lograríamos que los tiempos de gap y los tiempos de latencia RTT, se vieran mejorados, proporcionando así un mejor servicio de encaminamiento.

IMPLICACIONES AMBIENTALES

Eficiencia energética:

-El uso de una tecnología radio de última generación, hace posible una mayor eficiencia energética respecto otras tecnologías (802.11), ya que el bajo consumo es una de sus mejores bazas.

Estudio energético: [13] IEEE 802.15.4 vs WLAN

Para demostrarlo compararemos estas 2 tecnologías. Pongamos que disponemos de 50000 hogares por ciudad y 100 dispositivos sensores por cada hogar para su control domótico.

Para este planteamiento evaluaremos 3 casos distintos:

Caso 1: IEEE 802.11

Potencia consumida para tx: 667mW (uso intensivo)
Potencia dedicada: 667mW*100*50000=**3.335MW**

Caso 2: IEEE 802.15.4

Potencia consumida para tx: 30mW (uso intensivo)
Potencia dedicada:30mW*100*50000=**150KW**

Caso 3: IEEE 802.15.4

Potencia consumida para tx: 30mW(0.1% de duty cycle)
Potencia dedicada: 30mW*100*50000*0.001=**150W**

Como podemos observar, si implementamos IEEE 802.15.4 a gran escala, la energía requerida es mínima lo que implica una menor emisión de gases nocivos de efecto invernadero

BIBLIOGRAFIA

[1] E. Belding-Royer, C. Perkins, “Evolution and future directions of the ad hoc on-demand distance-vector routing protocol”, 2003.

[2] Estándar IEEE 802.15.4
http://standards.ieee.org/getieee802/download/802.15.4-2003.pdf

[3] Datasheet chipset de comunicaciones CC2420
www.chipcon.com/files/CC2420_Data_Sheet_1_2.pdf

[4] RFC 3561 - Ad hoc On-Demand Distance Vector (AODV) Routing
http://www.faqs.org/rfcs/rfc3561.html

[5] TinyOS Tutorial
http://www.tinyos.net/tinyos-1.x/doc/tutorial/

[6] David Gay, Philip Levis, David Culler, Eric Brewer, “nesC 1.1 Language Reference Manual” http://nescc.sourceforge.net/papers/nesc-ref.pdf

[7] Philip Buonadonna,JasonHill,DavidCuller Active “Message Communication for Tiny Networked Sensors” http://www.tinyos.net/papers/ammote.pdf

[8] “Getting Started Guide” de xbow
http://www.xbow.com/Support/Support_pdf_files/Getting_Started_Guide.pdf

[9] Andrew S. Tanenbaum “Redes de computadoras” cuarta edición.

[10] TinyOS mailing list, https://mail.millennium.berkeley.edu/pipermail/tinyos/

[11] TinyOS installation guide,
www.spp.co.jp/xbow/02_TinyOS_Installation_j.pdf

[12] System Architecture Directions for Networked Sensors
www.cs.virginia.edu/~qc9b/cs851/SADofNS_2.ppt

[13] Patrick Kinney, “ZigBee Technology: Wireless Control that Simply Works “
http://www.zigbee.org/imwp/idms/popups/pop_download.asp?contentID=5162

[14] Georgia Institute of technology ,“A survey on sensor networks”

Escola Politècnica Superior
de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

ANEXOS

TÍTULO DEL TFC/PFC : Evaluación de una red de sensores con protocolo AODV y tecnología radio IEEE 802.15.4

TITULACIÓN: Ingeniería Técnica de Telecomunicación, especialidad Telemática

AUTOR: Óscar Alonso Blanco

DIRECTOR: Carles Gomez i Montenegro

FECHA: 5 de septiembre de 2005

Anexo A: Funcionamiento de TinyAODV

De forma precisa, wsnAODV es un componente que esta formado por diversos subcomponentes, cada uno de los cuales tiene tareas que son esenciales para su correcto funcionamiento.

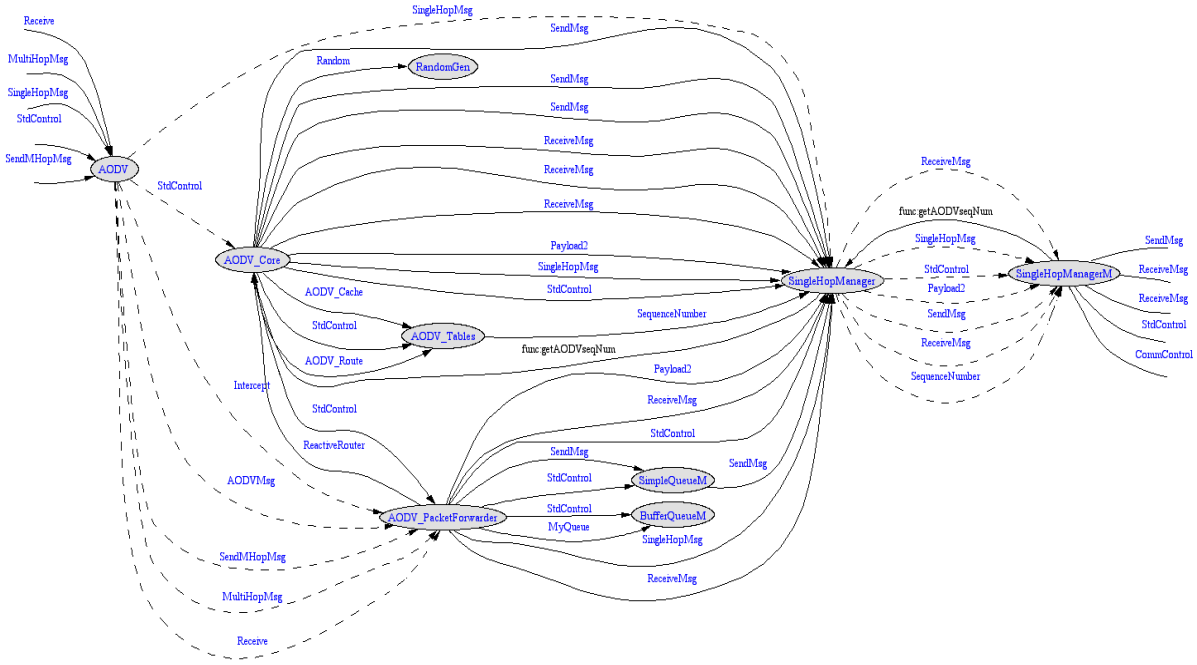


Fig A.1 Componente AODV y todos sus sub-componentes.

En el esquema podemos ver la interrelación de todos estos componentes.
Los más importantes en esencia son los siguientes:

- *AODV_Core*: Este componente se encarga de forma directa, entre otras cosas, del proceso de descubrimiento de rutas así como de intermediar con el componente AODV_Tables.nc para la correcta gestión de las rutas, podríamos asemejarlo a la capa de red.
- *AODV_PacketForwarder:* Se encarga de definir las interficies necesarias para el envío y recepción de paquetes por las rutas solicitadas. También en caso de ver de que no existe una ruta, se encarga de iniciar el proceso de descubrimiento que continuará en el componente AODV_Core.nc.
- *AODV_Tables*: Con este componente controlamos las 3 tablas que dispone cada nodo y nos permite realizar búsquedas, eliminar, insertar dentro de las rutas, en definitiva, nos permite la gestión completa de las mismas.
- *SimpleQueueM y BufferQueueM*: Son dos componentes que gestionan dos colas pero con objetivos un poco distintos. En el caso de BufferQueueM guardamos mensajes que no se han podido enviar por no disponer de ruta y SimpleQueue, es un pequeño buffer de envío de

paquetes una vez ya confirmada la ruta, ambos son de carácter FIFO y son manejados por AODV_PacketForwarder.

- *SingleHopManager*: Es el encargado de las comunicaciones salto a salto y de las comunicaciones UART, como podemos observar en el gráfico de arriba tiene múltiples vinculaciones con AODV_Core y de forma indirecta con AODV_PacketForwarder.
- *RandomGen:* Es un componente generador de números aleatorios, utilizado para aleatorizar la entrada al medio dentro del proceso de descubrimiento.

A.1. AODV_Core.nc

Como antes hemos dicho en el breve corolario de la página anterior, de forma esencial AODV_Core se encarga de llevar a cabo el proceso de descubrimiento y es por ello que, en el fichero se definen tres tareas que serán ejecutadas cuando cualquiera de las otras interficies haga una llamada (post) a estas.

```
task void sendRerr()
task void sendRreq()
task void sendRreply()
```

Cualquiera de estas tareas tiene asociado un evento de confirmación de envío (SendDone) y un evento de recepción (receive) diferenciado según sea un paquete RREQ, RERR o RREP:

```
event result_t SendRreq.sendDone(TOS_MsgPtr sentBuffer, bool success)
event result_t SendRreply.sendDone(TOS_MsgPtr sentBuffer, bool success)
event result_t SendRerr.sendDone(TOS_MsgPtr sentBuffer, bool success)
event TOS_MsgPtr ReceiveRreq.receive(TOS_MsgPtr receivedMsg)
event TOS_MsgPtr ReceiveRreply.receive(TOS_MsgPtr receivedMsg)
event TOS_MsgPtr ReceiveRerr.receive(TOS_MsgPtr receivedMsg)
```

En los eventos receive es donde se gestan o se eliminan rutas y en donde existe un elemento aleatorizador de entrada al medio (únicamente en ReceiveRreq), que sirve para ejecutar de forma inmediata un envío o postponerlo hasta 250mSg. La constante que regula el porcentaje de veces que un mensaje se pospone se llama AODV_MAX_RAND, así un mensaje se pospondrá en el 1/AODV_MAX_RAND de las veces.

Con respecto al mecanismo para definir o no los envíos de forma inmediata, disponemos de una serie de estados, uno para cada acción (RREQ,RREP y RERR).

En cada una de estas acciones, existen 3 variables llamadas rreqTaskPending, rreplyTaskPending, rerrTaskPending respectivamente que son iteradas en diversas ocasiones y en especial cada vez que el Timer lanza el evento enunciando su expiración cada 250mSg.

Estas tres variables pueden tener 3 estados distintos:

- TASK_DONE cuando no tenemos tarea alguna.
- TASK_PENDING: Que nos informa de que en estos momentos tiene una tarea pendiente.
- REPOST_REQ: En este caso cada 250 msg se preguntará a las variables si están en este estado y se encargará el envío de una tarea de las anteriormente comentadas según el estado de la variable.

Existe un tercer grupo de funciones que son las que interaccionan con AODV_packet_forwarder, AODV_tables. Son capaces de poner en marcha un proceso de descubrimiento, invalidar una ruta, comenzar un procedimiento RERR u obtener el siguiente nodo perteneciente a un destino definido en la tabla local de rutas:

```
command wsnAddr ReactiveRouter.getNextHop(wsnAddr dest)
command result_t ReactiveRouter.generateRoute(wsnAddr dest, uint8_t flag)
command result_t ReactiveRouter.SendRouteErr(wsnAddr dest)
command result_t ReactiveRouter.invalidateRoute(wsnAddr dest)
```

A.2. AODV_PacketForwarder.nc

Como antes hemos dicho, este componente se encarga tanto de la emisión como de la recepción de mensajes una vez ya tenemos una ruta establecida.

Como el principal proceso que realizamos es el envío de mensajes, definimos 2 tareas: una claramente definida para el envío de mensajes y la otra para la cola de mensajes no enviados:

```
task void enviarMsg()
task void buidarCua()
```

En enviarMsg lo que hacemos es comprobar si tenemos alguna ruta que nos permita llegar al destino que nos han pedido.

En caso afirmativo y habiendo obtenido el siguiente nodo que nos permita alcanzar nuestro destino, mandamos a nuestra cola de envío el mensaje con su destino y a partir de aquí es este componente el que se encarga de la emisión.

En caso de no obtener la dirección del siguiente nodo para llegar a nuestro destino quiere decir que la ruta ha sido invalidada o jamás ha sido descubierta.

Si es así, miraremos el número de intentos que lleva el mensaje (puede suceder que ya se haya intentado enviar otras veces), si esta es superior a MAX_INTENTS_GLOBALS, enviaremos un mensaje RERR anunciando la ruta que no se ha conseguido establecer mediante procesos de descubrimiento anteriores. En caso que no hayamos sobrepasado el número máximo de intentos, incrementaremos los intentos, encolaremos el mensaje en el buffer AODV y comenzaremos un proceso de descubrimiento (ReactiveRouter.generateRoute).

Después de esto establecemos un tiempo de descubrimiento mediante un timer, este tiempo de descubrimiento viene marcado por la variable DISCOVERY_PERIOD y en función del número de intentos se esperará lo siguiente: DISCOVERY_PERIOD*(SendIntent+1), donde “SendIntent” es el número de intentos.

Después de este tiempo de descubrimiento, se desencola el siguiente paquete mediante la tarea “buidarCua()” y se postea de forma automática una nueva tarea “enviarMsg()”, siempre y cuando tengamos mensajes esperando en la cola, en caso contrario no se postearán nuevos mensajes hasta que la interficie sendTTL no diga lo contrario.

Como hemos dicho en el último párrafo, en AODV_PacketForwarder para poder crear tareas son necesarias las interficies:

```
command result_t SendMHopMsg.sendTTL[uint8_t app](uint16_t dest, uint8_t len, TOS_MsgPtr
msg, uint8_t ttlfield)
```

Esta interficie de comando prepara la cabecera del paquete de datos AODV para el momento en que pueda ser enviado. El momento dependerá de si existe una ruta creada o no y de que la llamada del mensaje no coincida con un proceso de descubrimiento o un envío pendiente, en cuyo caso encolaremos el mensaje correspondiente, en caso contrario invocaríamos de forma directa la tarea enviarMsg().

En AODV_PacketForwarder también tenemos otras interficies que, aunque son de menor relevancia, si que son importantes para el funcionamiento general de nuestro componente wsnAODV:

```
command result_t SendMHopMsg.forwardTTL[uint8_t app](TOS_MsgPtr msg)
default event result_t SendMHopMsg.sendDone[uint8_t app](TOS_MsgPtr sentBuffer, result_t
success)
command void * SendMHopMsg.getBuffer[uint8_t app](TOS_MsgPtr msg, uint16_t *len)
```

Las interficies referentes a SendMhopMsg son eventos y comandos entre los que nos encontramos con:

- forwardTTL que nos permite el reenvío de mensajes sin que sea tocada la cabecera AODV del mismo
- getBuffer que nos permite ubicar el lugar de memoria en donde almacenaremos los datos que posteriormente enviaremos mediante sendTTL.
- sendDone, que no es más que la confirmación del envío.

Con esto parece lógico que deberíamos tener una cuarta interficie a modo de evento de recepción, pero esto no hará falta ya que como veremos en el siguiente párrafo, de esto se encarga la interficie comunicaciones salto a salto SingleHopReceive:

```
event TOS_MsgPtr SingleHopReceive.receive(TOS_MsgPtr msg)
```

Desde esta interficie decidimos varias cosas como determinar si un mensaje es para el nodo local o tiene que enviarlo hacia su destino, si un mensaje broadcast ha sido enviado o no (discrimina paquetes de broadcast una vez han sido enviados por primera y única vez).

Otra interficie importante es:

```
event result_t SingleHopSend.sendDone(TOS_MsgPtr sentBuffer, result_t success)
```

Ésta nos indica si un mensaje que ha sido enviado ha recibido con éxito la confirmación a nivel de enlace (ack). Si se produjera una confirmación negativa o que el propio evento sendDone nos devolviera un resultado no exitoso, la ruta

quedaría invalidada ya que quitaríamos la ruta defectuosa y encolaríamos el mensaje en el buffer de AODV para así no tener que perder el mensaje.

También tenemos otras interficies de comandos que únicamente atienden a la encapsulación de ciertas variables para que su uso posterior sea más sencillo y rápido:

```
command wsnAddr MultiHopMsg.getSource(TOS_MsgPtr msg)
command wsnAddr MultiHopMsg.getDest(TOS_MsgPtr msg)
command uint8_t MultiHopMsg.getApp(TOS_MsgPtr msg)
command uint8_t MultiHopMsg.getLength(TOS_MsgPtr msg)
command uint8_t AODVMsg.getSequenceNum(TOS_MsgPtr msg)
command uint8_t AODVMsg.getTTL(TOS_MsgPtr msg)
command uint8_t AODVMsg.getFlag(TOS_MsgPtr msg)
command uint8_t AODVMsg.getNext(TOS_MsgPtr msg)
```

Todas son referentes al mensaje que vamos a enviar tanto por parte de la cabecera directa de AODV (AODVMsg) como por la sub-cabecera (multihopMsg).

Por último tenemos estas 2 interficies:

```
event result_t DiscoveryTimer.fired()
event TOS_MsgPtr UARTReceiveMsg.receive(TOS_MsgPtr msg)
```

La primera es referente a la interficie del evento del temporizador de descubrimiento que únicamente nos cambia el flag de descubrimiento de ruta a falso y llama a un subprograma que llama a la tarea buidarcua().

El segundo se encarga de las comunicaciones con el puerto serie que, a su vez, delega las funciones de AODV a su control aunque en nuestro estudio no entraremos en su uso dado que en la práctica no ha sido utilizada.

A.3. AODV_Tables.nc

Este componente se ocupa de la gestión de las distintas tablas que posee cada nodo. Para ser más exactos, cada nodo tiene 3 tablas:

- La tabla de rutas (routeTable): En esta tabla guardamos todas las rutas con la dirección del siguiente nodo (nexthop) la cual nos permitirá llegar a nuestro destino.
- La tabla de route request (RREQ) (rreqCache): Cuando inundamos el medio con mensajes de RREQ, esta tabla nos permite almacenar la ruta de destino y la de origen. La razón de ser de esta tabla es no tener que repetir de forma innecesaria mensajes que hayamos recibido con anterioridad evitando así retransmisiones innecesarias y bucles debido a que los RREQ son mensajes en broadcast.
- La tabla de paquetes broadcast (aodvCache): Los mensajes broadcast también son enviados por inundación por lo tanto para evitar de nuevo las retransmisiones innecesarias, verificamos esta tabla y así sabemos si nuestro mensaje ya ha sido retransmitido o no.

Así, para la gestión de la tabla de rutas utilizamos

```
command uint8_t AODV_Route.RemoveRoute(wsnAddr nextnode)
command result_t AODV_Route.invalidateRoute(wsnAddr node)
command result_t AODV_Route.updateRouteInfo(wsnAddr dest, wsnAddr nextHop, uint16_t
destSeq, uint8_t numHops, uint16_t mySeq, uint8_t flag)
```

Como podemos ver por los propios nombres, gestiona entradas completas invalidándolas, modificándolas o borrándolas. Se trataría de un encapsulado a nivel de tabla.
Disponiendo del destino tenemos interficies de consulta a nivel de ruta:

```
command wsnAddr AODV_Route.getNextHop(wsnAddr dest)
command uint8_t AODV_Route.getRouteSeqNum(uint8_t ind)
command uint8_t AODV_Route.getRouteIndex(wsnAddr node, bool dest)
command uint8_t AODV_Route.getReverseRoute(wsnAddr nodeSrc)
command uint8_t AODV_Route.getReverseHops(wsnAddr nodeSrc)
```

Como vemos esto nos permite obtener directamente y de forma encapsulada, un campo que cierto momento tenga relevancia.
Para la tabla de RREQ, disponemos de tres interficies:

```
command result_t AODV_Cache.checkCache(wsnAddr src, uint16_t rreqID, uint8_t numHops)
command result_t AODV_Cache.updateCache(wsnAddr dest, wsnAddr src, wsnAddr nextHop,
uint16_t rreqID, uint16_t destSeq, uint8_t numHops, uint8_t flag)
command result_t AODV_Cache.deleteCache()
```

Como vemos el manejo es un poco más sencillo que con la tabla anterior, y nos da muchas menos opciones de tratamiento individual de datos, aunque vemos que no es necesario, viendo como esta planteado su uso. Cada vez que iniciemos un nuevo procedimiento de descubrimiento borraremos esta tabla con la tercera interficie.

Para la tabla de broadcast, únicamente necesitaremos estas 2 interficies:

```
command result_t AODV_Cache.checkCacheData(wsnAddr dest, wsnAddr src, uint8_t app,
uint8_t aodvSeq)

command result_t AODV_Cache.updateCacheData(wsnAddr dest, wsnAddr src, uint8_t app,
uint8_t aodvSeq, uint8_t flag)
```

Por ultimo tenemos una interficie de evento que nos avisa cuando tenemos que actualizar el número de secuencia correspondiente a todas las rutas:

```
event void SequenceNumber.updateSeqNum(wsnAddr addr, uint8_t seqNum, uint16_t mySeq)
```

Aquí el subprograma se encarga de actualizar el número de secuencia de la tabla de rutas (routeTable).

A.4. SingleHopManagerM.

Este componente se encarga de las comunicaciones a un salto :

Para ello tenemos una serie de interficies que se encargan del envío y la recepción de mensajes que andan directamente enlazados con el componente de comunicaciones:

```
command result_t SendMsg.send[uint8_t id](uint16_t addr, uint8_t length, TOS_MsgPtr msg)
default event result_t SendMsg.sendDone[uint8_t id](TOS_MsgPtr msg, result_t success)
event result_t RadioSend.sendDone[uint8_t id](TOS_MsgPtr msg, result_t success)
default event result_t SendMsg.sendDone[uint8_t id](TOS_MsgPtr msg, result_t success)
event TOS_MsgPtr RadioReceive.receive[uint8_t id](TOS_MsgPtr msg)
default event TOS_MsgPtr PromiscuousReceiveMsg.receive[uint8_t id](TOS_MsgPtr msg)
default event TOS_MsgPtr ReceiveMsg.receive[uint8_t id](TOS_MsgPtr msg)
```

También algunas como SendMsg.send , SendMsg.SendDone y Radioreceive.receive son las interficies de entrada i salida del componente, el resto que aquí destacamos son interficies de evento que andan relacionadas con el componente de comunicaciones llamado *PromiscuousCommNoUART*.

Otras interficies cubren como en ocasiones anteriores encapsulación de variables, para que su acceso sea más cómodo:

```
command wsnAddr SingleHopMsg.getSrcAddress(TOS_MsgPtr msg)
command wsnAddr SingleHopMsg.getDestAddress(TOS_MsgPtr msg)
command wsnAddr SingleHopMsg.getSeqNum(TOS_MsgPtr msg)
```

También tenemos otra clase de interficies en modo comando que nos cubren funciones de evaluación de longitud de los datos:

```
command void * Payload2.linkPayload(TOS_MsgPtr msg, uint8_t *len)
default command void * SubPayload2.linkPayload(TOS_MsgPtr msg, uint8_t *len)
```

Por ultimo ponemos de forma obligada el evento de actualización de número de secuencia:

```
default event void SequenceNumber.updateSeqNum(wsnAddr addr, uint8_t seqNum, uint16_t
mySeq)
```

Este evento esta creado en este componente pero se utiliza propiamente en AODV_Tables, actualiza el número de secuencia cuando recibe algún mensaje por la el componente singlehopmanager.

A.5. SimpleQueueM y BufferQueueM:

Se trata de dos colas FIFO, pero con propósitos muy distintos , SimpleQueueM es una cola de envío directo de mensajes y BufferQueue es una cola diseñada para almacenar mensajes que no han podido ser enviados por AODV.

SimpleQueueM

Como hemos dicho antes se trata de una cola FIFO, con capacidad de envío, existen 2 parámetros AODV que dependen directamente de este componente,

por un lado SEND_QUEUE_MAX_RETRIES, con el cual definimos el número máximo de intentos de envío antes de devolver un error y SEND_QUEUE_SIZE con quien definimos el tamaño de la cola de mensajes FIFO.
En caso de tener mensajes dispuestos a ser enviados siempre se enviaran con la mayor celeridad posible, de tal forma que puede que sean enviados de forma seguida en caso de haber más de un mensaje.

Las interficies más importantes y que se interrelacionan con otros componentes como *AODV_PacketForwarder* y *SingleHopManagerM*

```
command result_t Send.send(uint16_t addr, uint8_t length, TOS_MsgPtr msg)
default event result_t Send.sendDone(TOS_MsgPtr msg, result_t success)
```

Lo que hace en realidad es encolar el mensaje en nuestra cola FIFO, y postea una tarea:

```
task void attemptToSend()
```

Esta tarea se encarga realmente del envío de un mensaje, para ello desencola un mensaje de la cola y trata de enviarlo, en caso de no poder enviarlo se llama a si misma hasta que consiga enviar el mensaje o hasta sobrepasar el número máximo de intentos (QUEUE_MAX_RETRIES), en cuyo caso el mensaje será desencolado.

Respecto a la cola FIFO, tenemos los 2 subprogramas responsables de encolar y desencolar los mensajes:

```
void enqueueMessage(QueueEnt *newEnt)
void dequeueMessage()
```

BufferQueueM

Como arriba hemos comentado este componente es utilizado como cola FIFO para guardar los mensajes AODV que queramos enviar y no hayamos podido hacerlo debido a la inexistencia temporal de una ruta. Posee 1 parámetros de nstAODV, QUEUE_SIZE que indica el tamaño máximo de nuestra cola FIFO.

Posee 2 interficies que interaccionan con *AODV_PacketForwarder*:

```
command result_t MyQueue.enQueue(TOS_MsgPtr msg, uint8_t number_tries)
command TOS_MsgPtr MyQueue.getNext(uint8_t *numtries)
```

Con la primera de estas interficies podemos encolar un mensaje después de hacer comprobaciones de capacidad, y con la segunda interficie obtenemos el siguiente mensaje de la cola FIFO y desencolamos a este en la misma interficie de comando.

Igual que en *SimpleQueueM* tenemos dos subprogramas que se encargan de la tarea explicita de encolar un mensaje.

```
   void enqueueMessage(QueueEnt *newEnt)
```

A.6. RandomGen

Únicamente se trata de un generador de números aleatorios, este es usado para generar números aleatorios, y aleatorizar la entrada al medio en componentes como *AODVCore*, la interficie de comando utilizada es la siguiente:

```
async command uint16_t Random.rand()
```

Nos devuelve un número aleatorio.

Anexo B:Parámetros por defecto de AODV

Dentro del RFC se definen una serie de constantes por defecto.

Tabla B.1. Parámetros de configuración AODV

Nombre del Parámetro	**Valor**
ACTIVE_ROUTE_TIMEOUT	3seg.
ALLOWED_HELLO_LOSS	2
BLACKLIST_TIMEOUT	RREQ_RETRIES × NET_TRAVERSAL_TIME
DELETE_PERIOD	k × max (ACTIVE_ROUTE_TIMEOUT, ALLOWED_HELLO_LOSS×HELLO_INTERVAL) (k se recomienda que sea 5)
HELLO_INTERVAL	1seg.
LOCAL_ADD_TTL	2
MAX_REPAIR_TTL	0.3 × NET_DIAMETER
MY_ROUTE_TIMEOUT	2 × ACTIVE_ROUTE_TIMEOUT
NET_DIAMETER	35
NET_TRAVERSAL_TIME	2×NODE_TRAVERSAL_TIME×NET_DIAMETER
NEXT_HOP_WAIT	NODE_TRAVERSAL_TIME + 10
NODE_TRAVERSAL_TIME	40 ms
PATH_DISCOVERY_TIME	2×NET_TRAVERSAL_TIME
RERR_RATELIMIT	10
RING_TRAVERSAL_TIME	2×NODE_TRAVERSAL_TIME × (TTL_VALUE + TIMEOUT_BUFFER)
RREP_WAIT_TIME	3×NODE_TRANSVERSAL_TIME×NET_DIAMETER/2
RREQ_RETRIES	2
RREQ_RATELIMIT	10
TIMEOUT_BUFFER	2
TTL_START	1
TTL_INCREMENT	2
TTL_THRESHOLD	7
TTL_VALUE	Valor del campo TTL en la cabecera IP mientras se hace la búsqueda con el expanding ring search

Anexo C: El entorno de TinyOS

TinyOS es un sistema operativo para sistemas empotrados, pero como tantos necesita de un entorno de desarrollo adecuado para su funcionamiento.
Para cumplir con un amplio elenco de potenciales usuarios se decidió potenciar 2 plataformas para el desarrollo aplicaciones, en ambas impera un entorno basado en interprete de comandos sh, primeramente se desarrolló la versión bajo linux en primer termino fue red hat 9.0, fueron desarrolladas las aplicaciones básicas como el compilador de nesC, así como las librerías necesarias para la utilización de esta.

Poco más tarde se implemento el kit de desarrollo bajo cygwin, que a partir de la base de windows XP y Windows 2000 crea un entorno "linux like" que permite a usuarios menos experimentados en instalaciones bajo entornos UNIX desarrollar aplicaciones para tinyOS de forma más sencilla, aparte también nos instala en caso de no estar en el sistema una maquina virtual de Java, en este caso instala la JVM 1.4.2. de sun.

La pequeña explicación que vamos a realizar de instalación la vamos a realizar bajo este entorno.

C.1. Proceso rápido de instalación

Bien, para empezar hay que ir a la pagina oficial de tinyOS (www.tinyOS.net) y bajarse los binarios correspondientes a la instalación de windows xp, para ello se va a la sección de descarga download, primero bajaremos el paquete correspondiente a la versión 1.0 de tinyOS, para ello picamos en la sección que pone "TinyOS CVS Snapshot Installation Instructions." , ahora habremos llegado a la pagina donde esta la última actualización de tinyOS, en este caso la última versión es la 1.1.14, hay que tener cuidado y bajarse la versión adecuada a la plataforma que queramos instalar.

Accedemos al servidor de descargas correspondiente a windows (http://webs.cs.berkeley.edu/tos/dist-1.1.0/tinyos/windows).

Una vez aquí descargamos los siguientes paquetes:
tinyos-1.1.0-1is.exe
nesc-1.1.2b-1.cygwin.i386.rpm
tinyos-1.1.14Jul2005cvs-1.cygwin.noarch.rpm

Los paquetes serán instalados por ese orden, el primero nos instala toda la plataforma básica de desarrollo (cygwin) con su intérprete de comandos sh, las librerías nesC de tinyOS y el compilador nesC. El entorno de instalación es bastante agradable e intuitivo.

El segundo paquete nos instala la versión más moderna del compilador nesC, su instalación es pre-requisito para poder instalar el parche con la última versión de las librerías tinyOS que es nuestro tercer paquete.

Así para proceder con la instalación de estos paquetes tendremos que arrancar una sesión de cygwin y desempaquetar los 2 RPM's respetando el orden que hemos mencionado:

```
rpm --force -ignoreos -Uvh nesc-1.1.2b-1.cygwin.i386.rpm

rpm --force --ignoreos -Uvh tinyos-1.1.14Jul2005cvs-1.cygwin.noarch.rpm
```

Una vez hayamos instalado los 2 paquetes RPM, nuestro sistema estará listo para trabajar.

Para verificar que la instalación completa ha sido hecha correctamente, utilizaremos un comando

C.2. Descripción del árbol de directorios tinyOS.

Todo el entramado de librerias de TinyOS hay que buscarlo dentro de cygwin, tal y como indicamos en la figura, podemos observar que dentro de cygwin la estructura, es la clásica de un sistema UNIX, vemos que es en la carpeta `opt/tinyos-1.x` donde encontraremos todos los ficheros referidos a tinyOS.

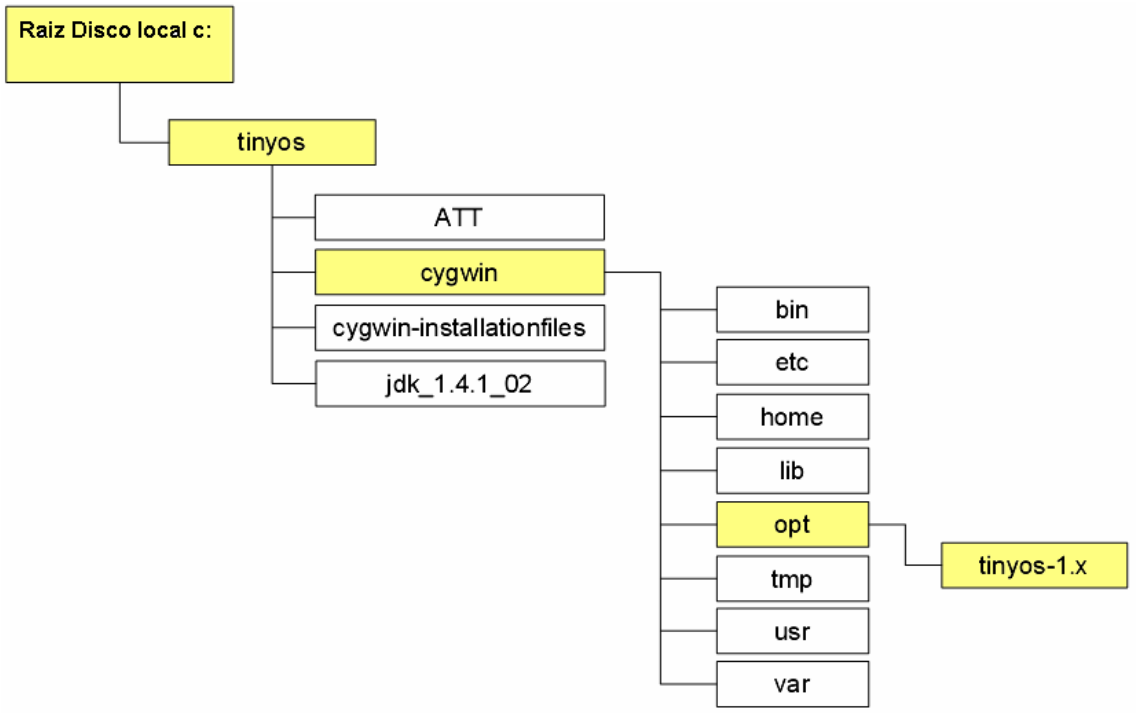


Fig C.1. Estructura de tinyOS desde la raíz.

Dentro de tinyOS la distribución de directorios y sus usos están explicados en las siguientes figuras es la siguiente figuras C.2. y C.3.

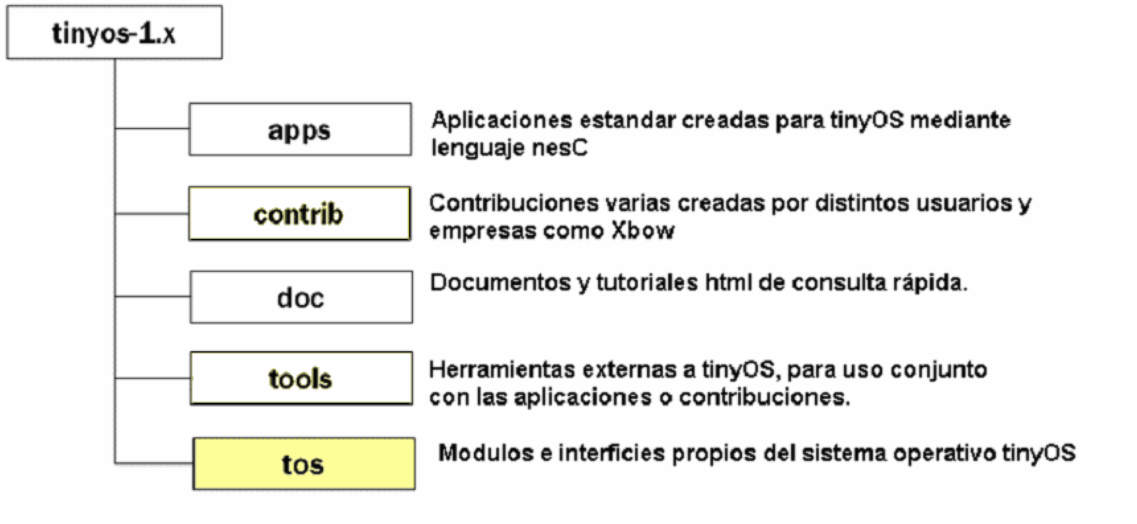


Fig C.2. Estructura principal y usos de los directorios de tinyOS

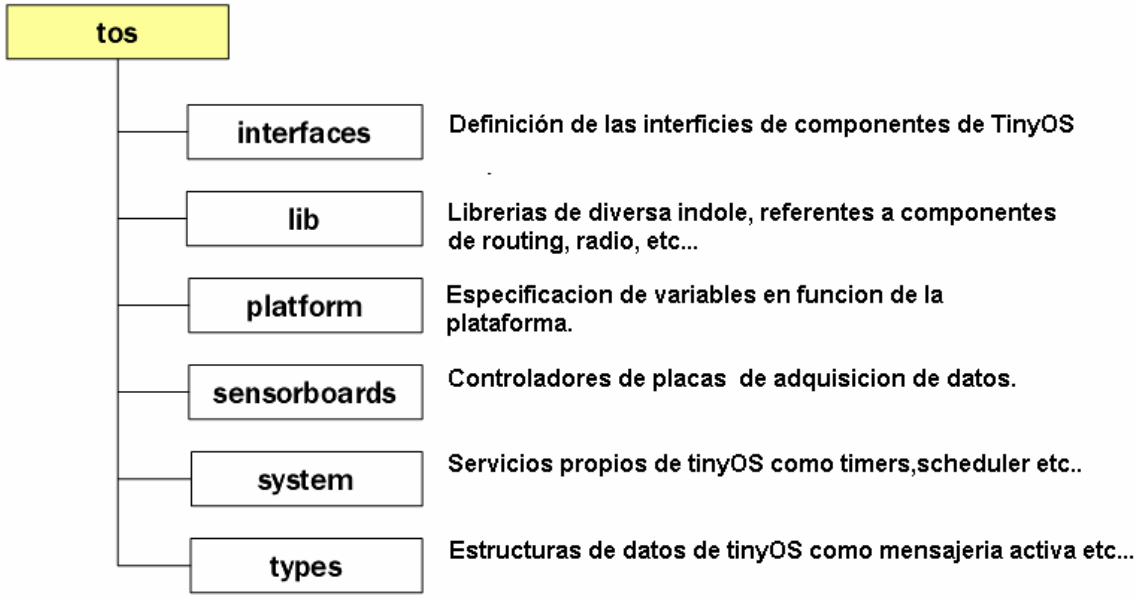


Fig C.3. Estructura y usos de la carpeta tos.

C.3 Ejemplo sencillo de NesC

Pasaremos a describir un programa sencillo en el lenguaje de programación NesC siguiendo el proceso para su funcionamiento y compilación.

El funcionamiento de este programa es sumamente sencillo, únicamente hace que parpadee una vez por segundo uno de los leds del dispositivo sensor.

```
configuration Blink {
}
implementation {
  components Main, BlinkM, SingleTimer, LedsC;
  Main.StdControl -> SingleTimer.StdControl;
  Main.StdControl -> BlinkM.StdControl;
  BlinkM.Timer -> SingleTimer.Timer;
  BlinkM.Leds -> LedsC;
}
```

Fig C.4. Ejemplo de fichero de configuración.

En este fichero de configuración (fig 3.2) podemos ver primeramente declarados los componentes necesarios para su uso dentro de los módulos. En este caso llamamos a cuatro: uno es el propio módulo *BlinkM* donde se desarrolla nuestro programa principal, *Main* es necesario para poder establecer pautas de inicialización, ejecución del código principal del componente y apagado correcto, los dos componentes restantes son para actuar sobre los LEDS (*LedsC*) y sobre temporizadores de tiempo (*SingleTimer*).

Las siguientes cuatro líneas tratan de enlazar las interficies que nos proporcionan estos componentes con el módulo BlinkM, para que así podamos usarlas.

```
module BlinkM {
  provides {
    interface StdControl;
  }
  uses {
    interface Timer;
    interface Leds;
  }
}
implementation {

  command result_t StdControl.init() {
    call Leds.init();
    return SUCCESS;
  }

  command result_t StdControl.start() {
    // Start a repeating timer that fires every 1000ms
    return call Timer.start(TIMER_REPEAT, 1000);
  }

  command result_t StdControl.stop() {
    return call Timer.stop();
  }

  task ejecutar()
  {
       call Leds.redToggle();
  }

  event result_t Timer.fired()
  {
    post ejecutar();
    return SUCCESS;
  }
}
```

Fig C.5. Ejemplo de fichero módulo.

En el fichero módulo, para enlazar con las interficies relacionadas con este fichero, hacemos una breve descripción de las interficies acorde con lo que hemos puesto en el fichero de configuración.

Encontramos 2 clases de interficies:

- Las que están en el claudator *provides* únicamente proporcionan las interficies sin funcionalidad y que nosotros tenemos que rellenar.
- Las que están en el claudator *uses* suelen ser interfaces ya desarrolladas y de las que únicamente sacaremos un uso al llamarlas con el comando call.

Después de describir las interficies, se pasa a la implementación. Cabe destacar que las 3 interficies de comando pertenecen a *stdcontrol* que es la interficie que tenemos que implementar en este caso.

En *StdControl.init* inicializamos todos los componentes que lo requieran, en este caso solo requiere de inicialización expresa el componente Leds (Leds.init). En programas más complejos, este sería el lugar adecuado para inicializar más componentes.

StdControl.start siempre es el lugar en donde se ejecutan los componentes necesarios para que sus eventos actúen, en este caso el componente Timer establece que cada segundo salte un evento de forma indefinida.

StdControl.stop se usa para el caso en que se resetea o se apaga el dispositivo sensor, es ese momento desactiva todas las interficies de aquellos componentes que necesiten pararse. Aquí podemos ver como el componente Timer necesita ser parado porque tiene la interficie *Timer.stop*.

Existe una cuarta interficie de evento perteneciente a *Timer*. Ésta es consecuencia de llamar al componente *Timer* y es de obligada implementación ya que si no nuestro compilador nos indicaría que no se ha implementado el evento que se encarga, cada segundo, de ejecutar el contenido de la interficie. Dentro de la interficie de evento encargamos una tarea (*post ejecutar*) que será ejecutada cuando el sistema disponga de los recursos adecuados, ejecutando la función que alterna el estado actual del led rojo.

Una cosa a tener en cuenta es que podemos realizar subprogramas completos en lenguaje C normal, ya que convivirán sin problemas con nesC.

Una vez tenemos esto, si queremos compilar la aplicación necesitaremos el compilador nesC y un *makefile.* Pondremos un ejemplo sencillo de makefile con este mismo programa:

```
COMPONENT=Blink
XBOWROOT=%T/../contrib/xbow/tos

PFLAGS= -I$(XBOWROOT)/platform/micaz

include ../MakeXbowlocal
include $(TOSROOT)/tools/make/Makerules
```

Fig C.6. Makefile de ejemplo

En la primera línea mencionaremos el nombre del componente principal que queremos compilar, el resto de líneas depende del dispositivo sensor que utilicemos, que en nuestro caso será MICAZ de la casa xbow. Primero definimos el directorio en donde tenemos las librerías de sistema de xbow y a continuación, en la tercera línea, cargamos la configuración de puertos para MICAZ.

Las dos últimas líneas son necesarias para la correcta compilación del programa.

Para saber más cosas referentes a tinyOS consultar los anexos D y L.

Anexo D: El programador ethernet MIB600

Esta sección ha sido creada por la poca información existente a propósito sobre este programador:

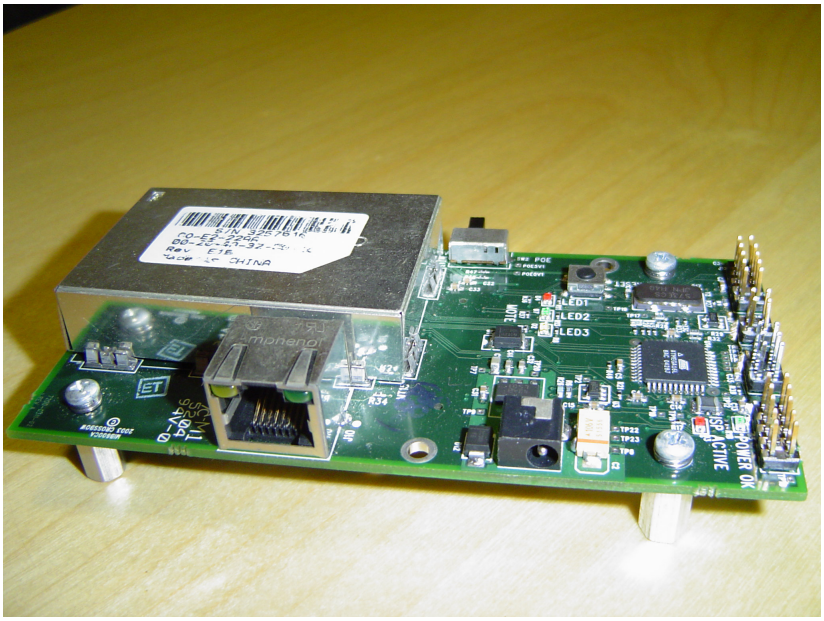


Fig D.1. Programador mib600

Para hacerlo funcionar primero hay que darle una dirección IP, para ello lantronix creo una aplicación llamada DeviceInstaller, el detectara si tiene dispositivos ethernet compatibles y te permitirá la asignación de una dirección IP para poder acceder de forma directa.
Esta aplicación lo podemos encontrar en el CD que viene dentro del kit administrado por xbow.

Una vez hecho esto únicamente queda configurar el programador para su correcto funcionamiento para programar dispositivos sensores MICAZ, con las siguientes figuras D.2. y D.3.

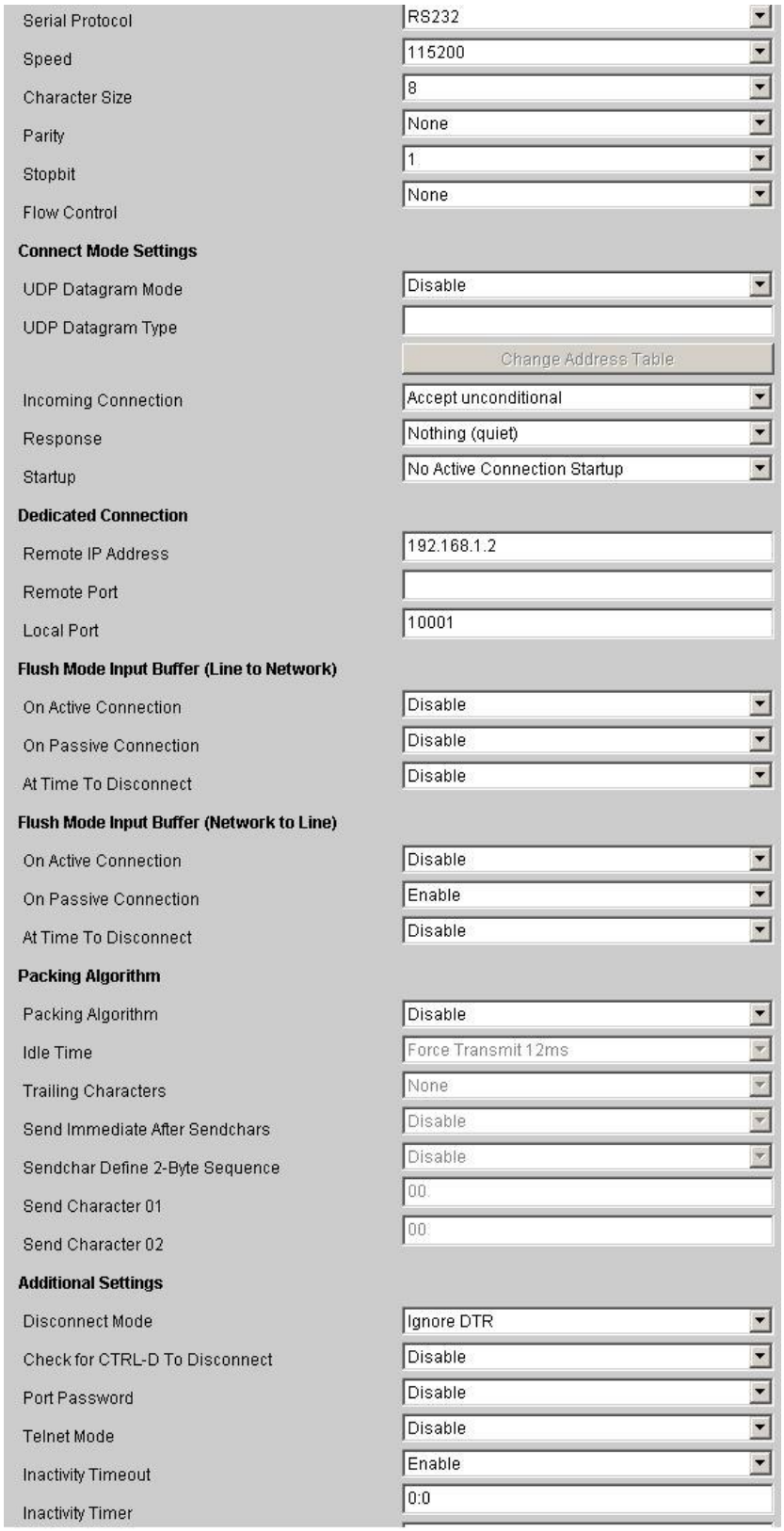


Fig D.2. Configuración del puerto 1

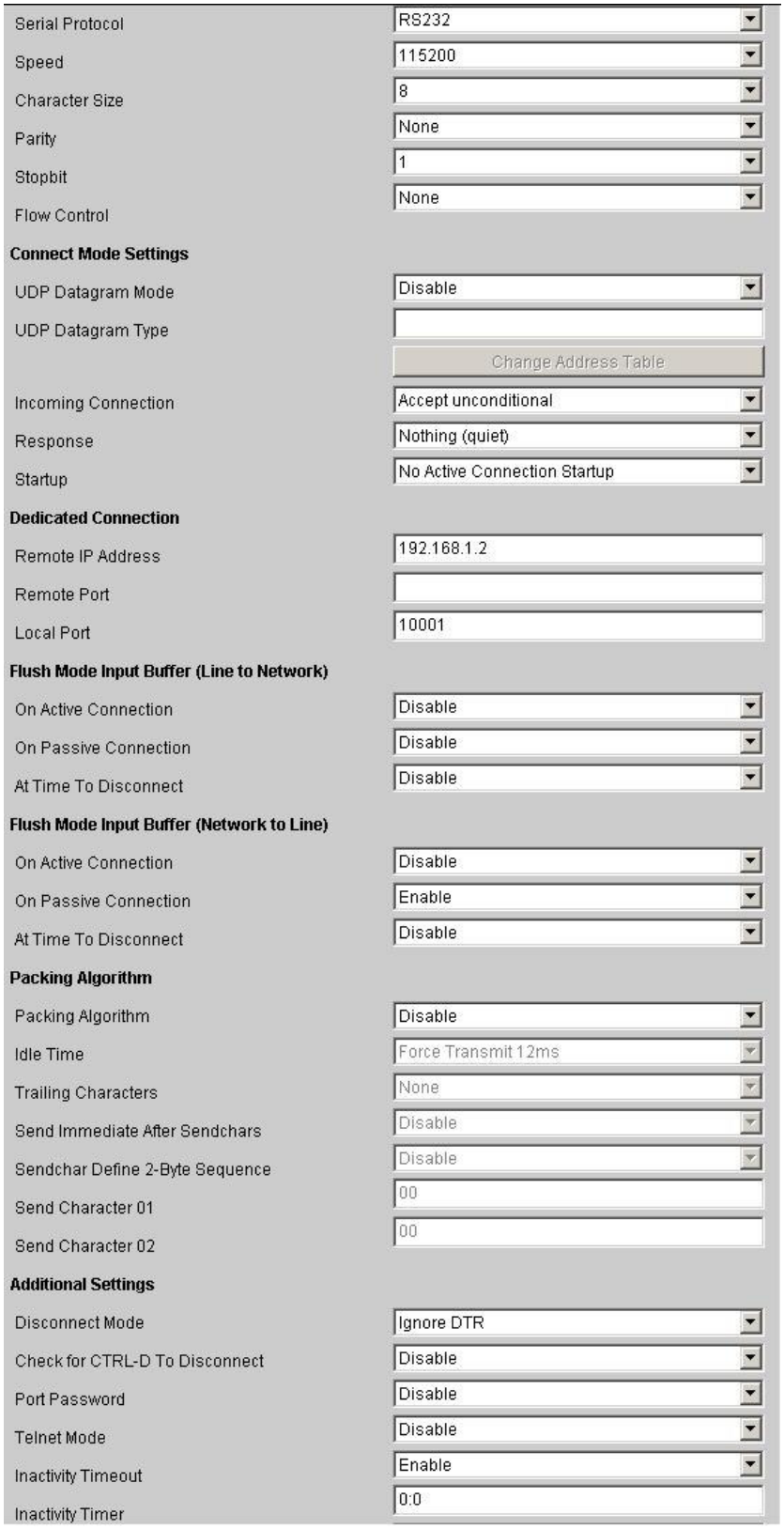


Fig D.3. Configuración del puerto 2

Anexo E: Implementaciones de AODV existentes

En este anexo se enumeran las implementaciones conocidas de AODV hechas hasta el momento. De cada una de las implementaciones enumeran los datos más importantes:

AODV-UCSB
- Desarrollado por la Universidad de Santa Bárbara (California)
- S.O: Free BSD.
- Licencia BSD.
- Trabaja con IPv4.
- Cumple con el Draft 6 de AODV.
- http://moment.cs.ucsb.edu/AODV/aodv.html#Implementations

AODV-UIUC net
- S.O.: Linux.
- Licencia GNU.
- IPv4.
- http://sourceforge.net/projects/aslib/

AODV-UU
- Desarrollado por la universidad de Uppsala (Suecia).
- S.O. : Linux y Linux Embedded.
- Licencia GNU.
- Cumple con el RFC 3561.
- IPv4.
- http://user.it.uu.se/~henrikl/aodv/

AODV-UU IPv6
- Se trata de una extensión para la implementación AODV-UU compatible con IPv6.
- S.O. : Linux.
- Cumple con el draft 1 de AODV-IPv6.
- http://user.it.uu.se/~henrikl/aodv/

AODV para Windows
- Desarrollado por Intel.
- Trabaja bajo Windows XP.
- http://moment.cs.ucsb.edu/AODV/aodv-windows.html

HUT AODV para IPv6
- Parte de la tesis de J. Tuominen.
- IPv6.
- S.O. : Linux
- Licencia BSD
- Cumple con el draft 1 de AODV-IPv6.
- http://www.tcs.hut.fi/~anttit/manet/aodv/

Kernel AODV

- Versión desarrollada por el NIST (Nacional Institute of Standarts and Technology) (EEUU)
- S.O. : Linux.
- IPv4.
- Cumple con el RFC 3561.
- http://w3.antd.nist.gov/wctg/aodv_kernel/

MAODV

- Versión desarrollada por la Universidad de Maryland (EEUU).
- Implementación de AODV Multicast basada en la versión 6 de AODV-UU.
- S.O: Linux.
- Licencia GNU.
- IPv4.
- http://www.isr.umd.edu/CSHCN/research/maodv/MAODV-UMD.html

UoB-JAdhoc

- Versión desarrollada por la Universidad de Bremen (Alemania).
- Implementada en Java.
- S.O.: Linux, Windows (XP y 2000) y Linux Embedded.
- Licencia GPL.
- IPv4.
- Cumple con el RFC 3561.
- http://www.aodv.org/modules.php?op=modload&name=UpDownload&file=index&menu=6

UoBWinAODV

- Versión desarrollada por la Universidad de Bremen (Alemania).
- Implementada en C ++.
- S.O.: Windows XP.
- IPv4.
- Cumple con el RFC 3561.
- http://www.aodv.org/modules.php?op=modload&name=UpDownload&file=index&menu=6

WinAODV

- Desarrollado por el departamento de Informática del Triniti Collage (Dublín).
- S.O: Windows XP y Windows CE.
- Ipv4.
- Cumple con el RFC 3561.
- http://www.dsg.cs.tcd.ie/?category_id=379

Anexo F: Diario del TFC

18/2/05→ Comienzo de 0
Llegada del kit e identificación de todos sus componentes.
-Lectura obligada: "getting started" de xbow. (Funcionamiento programador MIB510.
-Primeras pruebas con programas de ejemplo de *contrib/xbow* con el programador MIB510.
-Documentación sobre lenguaje nesc.
-Documentación genérica sobre AODV.
-Investigamos el funcionamiento del programador Ethernet sin demasiado éxito.

11/3/05→ -Localización de pila genérica tinyAODV.
-Estudio de todo su código y adaptación a la plataforma MICAZ (redireccionamiento de librerías y modos de funcionamiento).
-Surgen muchos problemas debido a que el autor del código no da ningún soporte para su funcionamiento. No se logra que todo funcione correctamente. Compila pero ni siquiera es capaz de confirmar el envío de un paquete (*SendDone*) .
-Error reportado en las listas de correo, igualmente los problemas persisten.

31/3/05→ -El programador Ethernet ya funciona correctamente. (Para programar).
-Explicaré detalladamente como lo hemos conseguido. He hablado con Pere de campus nord y me ha comentado que tiene su propia versión de AODV, le he pedido que en cuanto pueda que me la envíe.

3/4/05→ -Pere Salvatella envía la pila AODV.
Se retocan todas las librerías para su funcionamiento en MICAZ. Consigo que compile.

4/4/05 → Periodo de aprendizaje de uso:
Reserva de espacio (*getbuffer*), asignación de datos y envío mediante *sendttl.* Y luego la posterior confirmación mediante senddone.

5/4/05→ La pila AODV de Pere no acaba de funcionar correctamente, realiza el envió, pero senddone dice que el envió no es correcto, de todas formas el nodo receptor detecta el paquete y reenvía el paquete hacia origen, lo detecta una vez, pero la pila AODV muere, no reacciona al siguiente intento.

7/4/05→ -Evento receive se dispara dos veces y en teoría solo se recibe un paquete.
-Con estas circunstancias me decido a hacer una prueba multihop a dos saltos, parece que el forwarding parece funcionar, para los nodos intermedios utilizo el mismo programa de recepción pero cambio el identificador de red, con lo que me queda un nodo pasivo con pila aodv.

12/4/05→ -Después de reportar los errores anteriores a Pere, me pasa una nueva versión de su pila AODV, volvemos a hacer adaptaciones

para que el código compile, en especial unas dentro del código que están adaptadas al nivel radio de mica2(cc1000) por las de micaz (CC2420).
-Como resultado vemos que nuestra pila AODV ya no se cuelga al hacer más de un envío, manteniendo la cadencia.
-Para realizar las pruebas multisalto , rebajo la potencia de los nodos ya que a potencia por defecto estos tienen un alcance de decenas de metros y las pruebas multisalto serian irrealizables, para ello utilizamos una librería llamada *cc2420const.h*, en ella bajamos la potencia al mínimo consiguiendo que la cobertura de un nodo pase de decenas de metros a centímetros.

13/4/05 → -Dentro de la pila AODV he visto que hay muchos sub-programas que hacen uso de la librería LED para el uso de estos, lo que hace que el funcionamiento del programa no me quede claro. En consecuencia reviso de nuevo todo el código para eliminar estas llamadas y así encenderlos cuando a mi me interese .

18/4/05 → -Una vez el código AODV parece funcionar correctamente, realizo un programa en C#, que su función es capturar los paquetes que envía el nodo, filtrando los que llegan al ultimo nodo para recoger un entero de 32 bit, este entero será el tiempo que tarda en ir y volver un paquete RTT.
Aparte el programa guarda los resultados en un fichero de texto.

21/4/05 → El programador ethernet ya funciona al 100%, sabíamos programar pero no podíamos monitorizar con serial forwarder, el problema era debido a un problema de configuración de la velocidad del puerto serie de los micaz, para ello hemos tenido que entrar en configurador web y cambiar al valor por defecto de micaz (57600bauds), en el serial forwarder había que poner la siguiente palabra: network@192.168.1.2:10002.
-Pere me han pasado un forwarder en JAVA que coge el control del nodo, y al cual los otros nodos pueden enviar información y recuperarla, por una parte no he logrado que funcione, y luego por otra no lo considero interesante, ya que alteraríamos nuestro escenario enviando resultados y haciendo pings al mismo tiempo.
-El programa zping consiste en programas distintos, el primero envía un paquete mediante aodv, momento en el que este nodo coge una referencia de tiempo de reloj (*systime*). El segundo una vez ha recibido el paquete envía otro paquete como respuesta. El primero una vez recibe el paquete toma otra referencia de tiempo (*systime*), resta las mismas, esta diferencia es el RTT, el RTT lo coloca en el siguiente paquete que vaya a enviar de forma temporizada.

23/4/05 → -Veo que puesto esto en la practica, no obtenemos las respuestas deseadas (RTT=0), la librería *systime* resulta no funcionar de forma correcta, el problema radicaba en la forma de llamarlo. El problema esta solucionado y zping comienza a dar resultados correctos.

25/4/05 → Estudio detenido de los paquetes que estamos utilizando (TOSMsg), lo hacemos para poder ubicar el resultado y indicar a

nuestro programa hecho en c# la posición exacta del entero con la información de RTT.
Para ello esnifamos paquetes y los comparamos con la estructura de AM.H.

2/5/05 → -802.15.4 dice que casi todos sus canales dan problemas de coexistencia con 802.11 exceptuando 2 canales 25 y el 26, así que cambio el canal por defecto al número 26. El programa de c# ya parece funcionar y empieza a dar resultados coherentes.

3/5/05 → Comienzo a realizar test de funcionamiento de 2 a7 nodos. Tenemos un nodo que dejar de funcionar. Definitivamente uno de los nodos ha petado. Pasamos a tener 6 nodos para realizar las pruebas. Es observable que las pruebas a más nodos el tiempo rtt aumenta, pero constamos que el primer RTT es siempre de 1.8S independiente de los nodos.
No sabemos en principio la topología que usa IEEE 802.15.4, miramos exhaustivamente todas las librerias de radio sin constatar si tenemos una topología p2p o star.

9/5/05 → -Comenzamos a mirar como consultar la batería, después de consultar otros programas de ejemplo vemos 2 posibles (*ADCC, Voltaje*), de momento no hemos conseguido ningún resultado, todo i que la compilación es correcta y el funcionamiento también el valor que nos devuelve de batería es igual a 0.

13/5/05 → Valores críticos TIME_DISCOVERY, número de repeticiones de envío vemos que no influye demasiado.
Fallo en pila AODV: Constatamos que no es capaz de recuperarse delante de una configuración en rombo, he reportado el error a Pere y me ha dicho que se lo mirara ese mismo fin de semana.

25/5/05→ -Pere me envía la 4 revisión de su AODV.
-El problema del rombo desaparece es capaz de recuperarse en muy poco tiempo.
-Si obligamos a cambiar demasiado el escenario se acaba colgando, pero lo consigo arreglar reiniciando el nodo más lejano.

28/5/05 -Acabo de encontrar la respuesta a la topología de red utilizado por el transceptor radio CC2420, existe un bit en un registro (*MDMCTRL0)* que es modificable que nos permite decir cuando un nodo es PAN coordinator y cuando no.

7/6/06 -Diseño de todos los escenarios para realizar las pruebas oportunas.

10/6/05 -Comienzo las pruebas: Pruebas de ping.
Pruebas de ANCHO DE BANDA.
Pruebas de batería.
Pruebas de GAP.

10/7/05 -Interrupción temporal de las pruebas.

25/7/05 -Reanudación de las pruebas.

7/8/05 -Finalización de todas las pruebas.

Anexo G: Serial forwarder

Serialforwarder es un programa que nos permite recuperar los datos adquiridos por nuestros dispositivos sensores y que son recogidos por un nodo recolector que es el que se comunica con nuestro serial forwarder. Este a su vez se puede comunicar con otros programas mediante sockets, recuperando los datos que los otros programas crean oportunos.

Es un programa realizado en JAVA por lo que será necesario tener instalada la maquina virtual de java (JVM) creada por sun, su ultima versión es la 1.5.0.04, y la podéis encontrar.

Según el programador que usemos (serie o ethernet) tendremos que configurarlo de una forma o de otra.

Con MIB510:
Como este programador es un elemento pasivo, únicamente tendremos que conectarlo al puerto serie COM1 con la configuración mostrada en la captura:

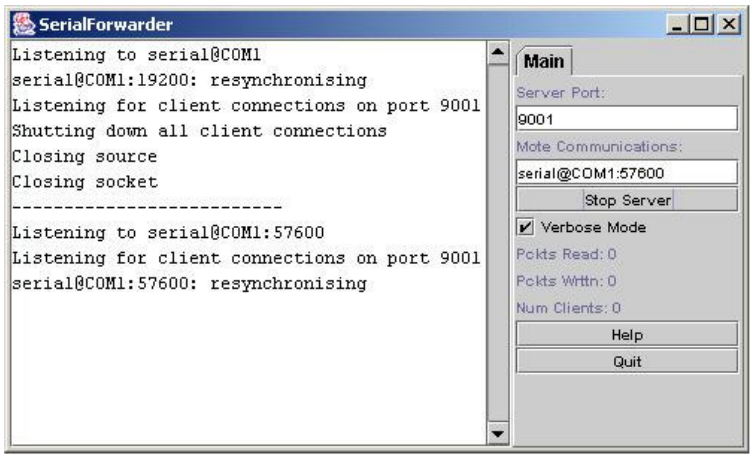


Fig G.1. Configuración para MIB510

Con MIB600:
Con la configuración previamente comentada en el anexo X, tenemos que aportar la dirección IP del programador así como el puerto por el que se comunica, la configuración exacta es la siguiente:

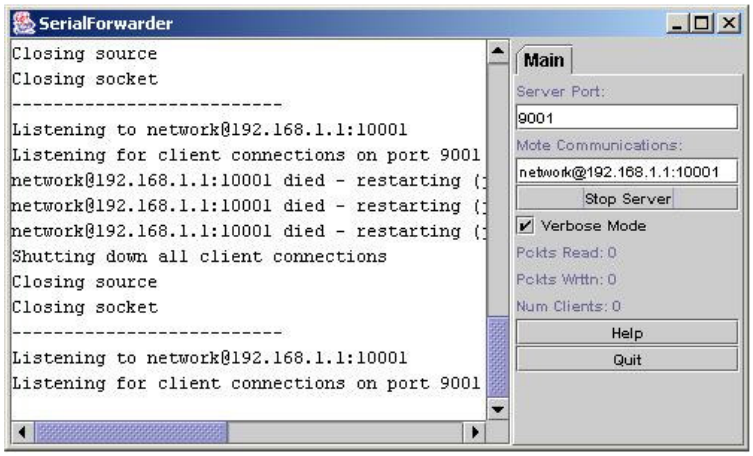


Fig G.2. Configuración MIB600

Anexo H: Código fuente de la implementación nstAODV

En este anexo encontraremos todos los archivos .nc correspondientes a la implementación nstAODV . Existen algunas modificaciones para su correcto funcionamiento con la plataforma MICAZ.

H.1 AODV.H

```
enum {
    ALTA_MOBILITAT    = 0x01,
    BAIXA_MOBILITAT = 0x00,

  INVALID_NODE_ID = 254,
  INVALID_INDEX       = 254,
    MAX_NUM_HOPS              = 254,
};

enum {
  TASK_DONE                  = 0,
  TASK_PENDING               = 1,
  TASK_REPOSTREQ      = 2
};

#ifndef CLOCK_SCALE
#define CLOCK_SCALE 1000L
#endif


typedef struct {
  wsnAddr   dest;                    // destination node of the route
  wsnAddr   nextHop;  // next hop to reach the destination
  uint16_t  destSeq;  // seq# of the lastest reception from the destination.
  uint8_t   numHops;  // number of hops to reach the destination
    uint16_t  lifetimeSeq;
    uint8_t           flag;
} AODV_Route_Table;


typedef struct {
    wsnAddr dest;
    wsnAddr precursor;
} aODV_Precursor_List;


typedef struct {
  wsnAddr   dest;                    // ok
  wsnAddr   src;                     // ok
  wsnAddr   nextHop;  // ok
  uint16_t  rreqID;           // ok
  uint16_t  destSeq;  // ok
  uint8_t   numHops;  // ok
    //uint8_t         flag;   // not used, for caching purposes only
} AODV_Route_Cache;

typedef struct {
  wsnAddr   dest;             // ok
  wsnAddr   src;              // ok
  uint8_t             app;           // ok
  uint8_t   seq;              // ok
} AODV_Data_Cache;
//PROPIETATS MODIFICABLES DE AODV

enum {
    // Interval de temps entre enviament de rreq i possibles recepcio de rreply.
    DISCOVERY_PERIOD                        = 100, //AODV_PacketForwarder.nc

    // Maxim nombre d'intents per enviar msg de dades quan la ruta falla. Amb
descobriment
    // de ruta a cada intent.
    MAX_INTENTS_GLOBAL                      = 4,          //AODV_PacketForwarder.nc
```

```
    // Maxim nombre d'intents sobre una destinació abans de retornar el msg al
AODV_PacketForwarder.
    SEND_QUEUE_MAX_RETRIES    = 3,           //SimpleQueueM.nc

    // Aleatorietat en l'enviament de missatges. Cada vegada que es fa una peticio
d'enviament es
    // retrassen uns 250 mseg el 75% de peticions.
    AODV_MAX_RAND                                   = 3,          //AODV_Core.nc regula
el número de missatges delayed

    // Maxim nombre de salts que pot tenir una ruta per considerar-se valida. Maxima
distancia a la
    // que arriba un msg BCAST.
    AODV_MAX_METRIC                         = 10,  //AODV_Core.nc maxim nombre de salts
en una ruta.

//Altres parametres de menys importancia en quant a resultats temporals.
    SEND_QUEUE_SIZE                         = 10,          //SimpleQueueM.nc
    QUEUE_SIZE                                                     = 10,
    //BufferQueueM.nc
    AODVR_NUM_TRIES                         = 2,           //AODV_Core.nc
    AODV_RTABLE_SIZE                        = 5,           //AODV_Core.nc
    AODV_RQCACHE_SIZE               = 5,           //AODV_Core.nc
    AODV_DATACACHE_SIZE             = 5,           //AODV_Core.nc
    DEFAULT_TTL                                            = 4,

};
```

H.2 AODV_Msg.h

```
enum {
  AM_ID_AODV = 9,                   // AODV data packet
  AM_ID_AODV_RREQ = 13,                         // AODV Rreq
  AM_ID_AODV_RREPLY = 14,                   // AODV Rreply
  AM_ID_AODV_RERR = 15,                         // AODV Rerr
  AM_ID_FROM_UART = 126,                    // Identifica missatges que venen de la UART

};

typedef struct SHop_Msg{
  uint8_t src; // indica el node que envia el missatge.
  uint8_t seq; // identifica el número de missatge total enviat pel node.
  uint8_t data[1]; // start of payload; size is not known at compile time
} SHop_Msg;

typedef SHop_Msg *SHop_MsgPtr;

typedef struct {
  uint8_t src;        //indica el node que ha originat el missatge
  uint8_t dest;              //indica la destinació final del missatge
  uint8_t app;        //per identificar l'aplicació
  uint8_t length;
} MHop_Header;

typedef struct AODV_Msg{
  MHop_Header mhop;
  uint8_t seq;         // identifica el número de missatge AODV.
  uint8_t ttl;        // indica número de salts.
    uint8_t flag;
  uint8_t data[1]; //start of payload; size is not known at compile time
} AODV_Msg;

typedef AODV_Msg *AODV_MsgPtr;

typedef struct AODV_Rreq_Msg{
  uint8_t dest;              // the destination desired for the RREQ
  uint8_t src;        // identifies the RREQ with the rreqID
  uint16_t rreqID;    // seq# identifies the RREQ with the src
  uint16_t srcSeq;  // seq# used for storing entries back to the source
  uint16_t destSeq;   // seq# last received from the destination by the source
  uint8_t metric[1];  // the number of hops from the Originator IP address
    uint8_t flag;
  uint8_t data[1];
} AODV_Rreq_Msg;
```

```
typedef AODV_Rreq_Msg* AODV_Rreq_MsgPtr;

typedef struct AODV_Rreply_Msg{
  uint8_t dest;              // the destination of the route suplied
  uint8_t src;        // the source of the route suplied
  uint16_t destSeq;   // seq# associated with the route
  uint8_t metric[1];  // The number of hops from the src to the dest
    uint8_t flag;
  uint8_t data[1];  // Utilitzat data[0] per indicar reply de servei.
} AODV_Rreply_Msg;

typedef AODV_Rreply_Msg* AODV_Rreply_MsgPtr;

typedef struct AODV_Rerr_Msg{
  uint8_t dest;                      // unreachable destination node
  uint16_t destSeq;          // seq# in the route table entry for the dest node in table
entry
    uint8_t flag;
  uint8_t data[1];
} AODV_Rerr_Msg;

typedef AODV_Rerr_Msg* AODV_Rerr_MsgPtr;

enum {
  SHOP_HEADER_LEN = offsetof(SHop_Msg, data),
  AODV_HEADER_LEN = offsetof(AODV_Msg, data),
  AODV_RREQ_HEADER_LEN = offsetof(AODV_Rreq_Msg, data),
  AODV_RREPLY_HEADER_LEN = offsetof(AODV_Rreply_Msg, data),
  AODV_RERR_HEADER_LEN = offsetof(AODV_Rerr_Msg, data),
};
```

H.3 AODV.nc

```
includes AODV_Msg;
includes AODV;

configuration AODV {
    provides {
        interface StdControl as Control;
        interface StdControl as TimerControl;

        interface Receive[uint8_t app];
        interface Intercept[uint8_t app];
        interface SendMHopMsg[uint8_t app];

        interface SingleHopMsg; // access to single hop packet decoding
        interface MultiHopMsg;        // access to multihop packet decoding
        interface AODVMsg;    // access to AODV packet decoding

        interface Payload2 as AODVPayload;

    }
}

implementation {
    components   SingleHopManager,   AODV_Core,   AODV_PacketForwarder,      SimpleQueueM,
BufferQueueM, AODV_Tables, RandomGen, TimerC, LedsC;

    //inicialitzacio de tots els moduls per ordre.
    TimerControl      = TimerC.StdControl;
    Control                  = BufferQueueM;
    Control                  = SimpleQueueM;
    Control                  = AODV_Tables;
    Control                  = SingleHopManager;
    Control                  = AODV_Core;
    Control                  = AODV_PacketForwarder;

    SendMHopMsg       = AODV_PacketForwarder.SendMHopMsg;
    Receive                  = AODV_PacketForwarder.Receive;
    Intercept         = AODV_PacketForwarder.Intercept;

    MultiHopMsg       = AODV_PacketForwarder.MultiHopMsg;
    AODVMsg                  = AODV_PacketForwarder.AODVMsg;
    SingleHopMsg      = SingleHopManager.SingleHopMsg;
```

```
    AODVPayload              = AODV_PacketForwarder.AODVPayload;

    //maneig de rutes
    AODV_PacketForwarder.ReactiveRouter      -> AODV_Core.ReactiveRouter;
    AODV_PacketForwarder.DiscoveryTimer      -> TimerC.Timer[unique("Timer")];

    //singlehop management
    AODV_PacketForwarder.SingleHopMsg        -> SingleHopManager;
    AODV_PacketForwarder.SingleHopReceive    ->
SingleHopManager.PromiscuousReceiveMsg[AM_ID_AODV];
    AODV_PacketForwarder.SingleHopPayload    -> SingleHopManager.Payload2;
    AODV_PacketForwarder.SingleHopSend       -> SimpleQueueM.Send;
    SimpleQueueM.SubSend      -> SingleHopManager.SendMsg[AM_ID_AODV];

    AODV_PacketForwarder.FindRouteQueue      -> BufferQueueM.MyQueue;

    // Base Station
#if BS
    AODV_PacketForwarder.UARTReceiveMsg      ->
SingleHopManager.ReceiveMsg[AM_ID_FROM_UART];
#endif
        //debugging
    AODV_Core.Leds     -> LedsC;
    AODV_PacketForwarder.Leds -> LedsC;
    SimpleQueueM.Leds -> LedsC;
    //AODV msg

    AODV_Core.SendRreq        -> SingleHopManager.SendMsg[AM_ID_AODV_RREQ];
    AODV_Core.ReceiveRreq-> SingleHopManager.ReceiveMsg[AM_ID_AODV_RREQ];
    AODV_Core.SendRreply      -> SingleHopManager.SendMsg[AM_ID_AODV_RREPLY];
    AODV_Core.ReceiveRreply   -> SingleHopManager.ReceiveMsg[AM_ID_AODV_RREPLY];
    AODV_Core.SendRerr        -> SingleHopManager.SendMsg[AM_ID_AODV_RERR];
    AODV_Core.ReceiveRerr
-> SingleHopManager.ReceiveMsg[AM_ID_AODV_RERR];

    AODV_Core.Payload2-> SingleHopManager.Payload2;
    AODV_Core.Timer    -> TimerC.Timer[unique("Timer")];
    AODV_Core.Random   -> RandomGen.Random;
    AODV_Core.SingleHopMsg -> SingleHopManager;

    AODV_Core.AODV_Route -> AODV_Tables.AODV_Route;
    AODV_Core.AODV_Cache -> AODV_Tables.AODV_Cache;
    AODV_PacketForwarder.AODV_Cache  -> AODV_Tables.AODV_Cache;
    AODV_Tables.SequenceNumber       -> SingleHopManager;
    SingleHopManager.getAODVseqNum   -> AODV_Core.getAODVseqNum;

    SimpleQueueM.Random       -> RandomGen.Random;
}
```

H.4 AODV_Core.nc

```
module AODV_Core {
        provides {
                interface StdControl as Control;
                interface ReactiveRouter;
                command uint16_t getAODVseqNum();
        }
        uses {
                interface SendMsg            as SendRreq;
                interface ReceiveMsg  as ReceiveRreq;
                interface SendMsg            as SendRreply;
                interface ReceiveMsg  as ReceiveRreply;
                interface SendMsg            as SendRerr;
                interface ReceiveMsg  as ReceiveRerr;

                interface Payload2;
                interface Random;
                interface Timer;
                interface SingleHopMsg;

                interface AODV_Route;
                interface AODV_Cache;

                interface Leds;
```

```
        }
}

implementation {

        TOS_Msg msgBuf1, msgBuf2, msgBuf3;
        TOS_MsgPtr rreqMsg;
        TOS_MsgPtr rReplyMsg;
        TOS_MsgPtr rErrMsg;

        uint8_t rreqTaskPending;
        uint8_t rreplyTaskPending;
        uint8_t rerrTaskPending;

        uint8_t rreqRandomize;        // baixem de 16 a 8 bits, no cal números tan grans

        uint8_t rreqNumTries;
        uint8_t rreplyNumTries;
        uint8_t rerrNumTries;

        bool sendPending;

        wsnAddr rReplyDest;

        uint16_t mySeq;               //destination sequence number for LOCAL NODE
        uint16_t rreqID;

        uint8_t ownFlag;

    void dbgPacket(TOS_MsgPtr data) {
    uint8_t it;
        dbg(DBG_ROUTE, "SRV-C, Debugging Packet :\n");
        dbg(DBG_ROUTE, "      MSG_DBG:");
    for(it = 0; it < (29); it++)
              dbg_clear(DBG_ROUTE, "%02hhx ", ((uint8_t *)data)[it]);
    dbg(DBG_ROUTE, "\n");
  }



    command result_t Control.init() {
        dbg(DBG_BOOT, "AODV_Core inicialitza\n");
        rreqMsg        = &msgBuf1;
        rReplyMsg = &msgBuf2;
        rErrMsg        = &msgBuf3;

        rreqTaskPending       = TASK_DONE;
        rreplyTaskPending = TASK_DONE;
        rerrTaskPending       = TASK_DONE;

        sendPending = FALSE;

        rreqRandomize  = 0;
        rreqNumTries          = 0;
        rreplyNumTries = 0;
        rerrNumTries          = 0;
        mySeq                                       = 0;
        rreqID                                = 0;

        rReplyDest  = INVALID_NODE_ID;

        call Random.init();
        call Leds.init();
        return SUCCESS;
    }

    command result_t Control.start() {
        dbg(DBG_ROUTE, "AODV_Core: Setting ScheduleTimer a %d\n", (CLOCK_SCALE/4));
        return call Timer.start(TIMER_REPEAT, (CLOCK_SCALE/4));
    }

    command result_t Control.stop() {
        dbg(DBG_ROUTE, "aturant service core timer\n");
        return call Timer.stop();
    }

    task void sendRreq() {
```

```
    // Bloquejem enviament de missatges
    if (!sendPending){
            sendPending = TRUE;
            if (call SendRreq.send(TOS_BCAST_ADDR,AODV_RREQ_HEADER_LEN,
            rreqMsg)==SUCCESS) {
                    dbg(DBG_ROUTE, "AODV_Core: s'ha enviat msg Rreq
                    satisfactoriament.\n");
                    // Crida correcta a send.
                    return;
            }else{
                    // Error al cridar send
                    dbg(DBG_ROUTE, "AODV_Core: error en la comanda
                    SendRreq.send(...).\n");
            }
    }else{
            // Error al sendPending
            dbg(DBG_ROUTE, "AODV_Core: error en la tasca sendRreq(), sendPending
            bloquejat.\n");
    }

    // En cas d'error:
    dbg(DBG_ROUTE, "AODV_core: allibera sendPending\n");
    sendPending = FALSE;

    // Comprovem si es el maxim nombre d'intents
    if (rreqNumTries > AODVR_NUM_TRIES){
            // Descartar el packet
            dbg(DBG_ROUTE, "AODV_Core: error, maxim nombre d'intents per al msg
            rreq.\n");
            rreqTaskPending = TASK_DONE;
    }else{
            // Tornar a intentar
            dbg(DBG_ROUTE, "AODV_Core: tornem a postejar la tasca sendRreq().\n");
            rreqTaskPending = TASK_REPOSTREQ;
    }
}



task void sendRerr(){
    // Bloquejem enviament de missatges
    if (!sendPending){
            sendPending = TRUE;
            if (call SendRerr.send(TOS_BCAST_ADDR, AODV_RERR_HEADER_LEN,
            rErrMsg)==SUCCESS) {
                    // Crida correcta a send. Un routeError no el posem en cache
                    dbg(DBG_ROUTE, "AODV_Core: s'ha enviat msg Rerr
                    satisfactoriament.\n");
                    return;
            }else{
                    // Error al cridar send
                    dbg(DBG_ROUTE, "AODV_Core: error en la comanda
                    SendRerr.send(...).\n");
            }
    }else{
            // Error al sendPending
            dbg(DBG_ROUTE, "AODV_Core: error en la tasca sendRerr(), sendPending
            bloquejat.\n");
    }

    // En cas d'error:
    sendPending = FALSE;

    rerrNumTries++;

    // Comprovem si es el maxim nombre d'intents
    if (rerrNumTries > AODVR_NUM_TRIES){
            // Descartar el packet
            dbg(DBG_ROUTE, "AODV_Core: error, maxim nombre d'intents per al msg
            rerr.\n");
            rerrTaskPending = TASK_DONE;
    }else{
            // Tornar a intentar
            dbg(DBG_ROUTE, "AODV_Core: tornem a postejar la tasca sendRerr().\n");
            rerrTaskPending = TASK_REPOSTREQ;
    }
}
```

```
    task void sendRreply(){
        //Comprovant variables missatge
        if (!sendPending){
                sendPending = TRUE;
                if (call SendRreply.send(rReplyDest, AODV_RREPLY_HEADER_LEN,
                rReplyMsg)==SUCCESS) {
                        // Crida correcta a send.
                        dbg(DBG_ROUTE, "AODV_Core: s'ha enviat msg Rreply
                        satisfactoriament.\n");
                        return;
                }else{
                        // Error al cridar send
                        dbg(DBG_ROUTE, "AODV_Core: error en la comanda
                        SendRreply.send(...).\n");
                }
        }else{
                // Error al sendPending
                dbg(DBG_ROUTE, "AODV_Core: error en la tasca sendRreply(), sendPending
                bloquejat.\n");
        }

        // En cas d'error:
        sendPending = FALSE;

        // Comprovem si es el maxim nombre d'intents
        if (rreplyNumTries > AODVR_NUM_TRIES){
                // Descartar el packet
                dbg(DBG_ROUTE, "AODV_Core: error, maxim nombre d'intents per al msg
                rreply.\n");
                rreplyTaskPending = TASK_DONE;
        }else{
                // Tornar a intentar
                dbg(DBG_ROUTE, "AODV_Core: tornem a postejar la tasca sendRreply().\n");
                rreplyTaskPending = TASK_REPOSTREQ;
        }
    }

    command wsnAddr ReactiveRouter.getNextHop(wsnAddr dest){
#if BS
        if (dest == TOS_UART_ADDR){
                dbg(DBG_ROUTE, "AODV_Core: getNextHop retorna TOS_UART_ADDR com a
                local.\n");
                return (TOS_UART_ADDR);
        }
#endif

        if (dest == TOS_LOCAL_ADDRESS){
                dbg(DBG_ROUTE, "AODV_Core: estem buscant ruta al LOCAL_NODE.\n");
                return TOS_LOCAL_ADDRESS;
        }

        if (dest == (uint8_t)TOS_BCAST_ADDR){
                dbg(DBG_ROUTE, "AODV_Core: getNextHop retorna TOS_BCAST_ADDR\n");
                return (uint8_t)TOS_BCAST_ADDR;
        }

        return call AODV_Route.getNextHop(dest);
  }

    command result_t ReactiveRouter.generateRoute(wsnAddr dest, uint8_t flag){
        uint8_t length;
        uint8_t seqNum;
        AODV_Rreq_MsgPtr msg = call Payload2.linkPayload(rreqMsg, &length);

        // Incrementem número de sequencia
        mySeq++;
        rreqID++;

        //guardem estat actual flag
        ownFlag = flag;

        dbg(DBG_ROUTE, "AODV_Core: incrementem num de sequencia a valor %d.\n", mySeq);

        if(rreqTaskPending == TASK_DONE){
                rreqTaskPending = TASK_PENDING;
```

```
                //Preparar tasca i rreqmsg
                dbg(DBG_ROUTE, "AODV_Core: generateRoute bloqueja rreqTask i inicia
                parametres.\n");
                rreqNumTries = 0;

                seqNum = call AODV_Route.getRouteSeqNum(call
                AODV_Route.getRouteIndex(dest,TRUE));
                if (seqNum == INVALID_INDEX){
                        seqNum = 0;
                }else{
                        dbg(DBG_ROUTE, "AODV_C, ruta en taula tenia la seq a
                        %d\n",seqNum);
                }

                msg->dest             = dest;
                msg->src              = TOS_LOCAL_ADDRESS;
                msg->rreqID           = rreqID;
                msg->srcSeq           = mySeq;
                msg->destSeq          = seqNum;
                msg->metric[0]        = 0;
                msg->flag                           = flag;

                //Postejar tasca
                dbg(DBG_ROUTE, "AODV_Core: generateRoute posteja rreqTask.\n");

                //cal incloure a la cache
                call AODV_Cache.updateCache(msg->dest,msg->src, TOS_LOCAL_ADDRESS, msg-
                >rreqID, msg->destSeq, msg->metric[0], msg->flag);

        post sendRreq();
        return SUCCESS;
        }else {
                // Error tasca pendent.
                dbg(DBG_ROUTE, "AODV_Core: error, RreqTask pendent no permet modificar
                parametres.\n");
        return FAIL;
        }
  }

  command result_t ReactiveRouter.SendRouteErr(wsnAddr dest){
        uint8_t length;
        AODV_Rerr_MsgPtr msg = call Payload2.linkPayload(rErrMsg, &length);

        call AODV_Cache.deleteCache();
        dbg(DBG_ROUTE, "AODV_Core: borrem la cache de missatges en bcast, i hi introduim
el msg rerr.\n");

        // Comprovem que tenim ruta afectada pel node
        if (call AODV_Route.getRouteIndex(dest, FALSE) == INVALID_INDEX){
                dbg(DBG_ROUTE, "AODV_Core: no hi ha rutes afectades per aquest node.\n");
                call Leds.redToggle();
                return FAIL;
        }

        // prosseguim si no hi ha tasca pendent
        if(rerrTaskPending == TASK_DONE){
                rerrTaskPending = TASK_PENDING;

                // Create RerrMsg.
                dbg(DBG_ROUTE, "AODV_Core: sendRouteErr bloqueja rerrTask i inicia
                parametres.\n");
                msg->dest = dest;
                msg->destSeq   = call AODV_Route.getRouteSeqNum(call
                AODV_Route.getRouteIndex(dest, TRUE));

                // Delete routeTable for node_id
                call AODV_Route.RemoveRoute(dest);
                dbg(DBG_ROUTE, "AODV_Core: borrem les rutes afectades.\n");

                // Posting  sendErr
                dbg(DBG_ROUTE, "AODV_Core: enviem missatge Rerr per dest %d.\n", msg-
                >dest);
                post sendRerr();

                return SUCCESS;
    }else{
                //route pending
```

```
                dbg(DBG_ROUTE, "AODV_Core: error, RerrTask pendent no permet modificar
                parametres.\n");
                return FAIL;
        }

    }

    command result_t ReactiveRouter.invalidateRoute(wsnAddr dest){
        dbg(DBG_ROUTE, "AODV_Core: invalidem ruta a destinació de %d.\n", dest);
        return call AODV_Route.invalidateRoute(dest);
    }

    event result_t SendRreq.sendDone(TOS_MsgPtr sentBuffer, bool success) {
        // Comprovem errors
        if (sendPending==FALSE){
                // Error sendPending hauria de ser TRUE
                dbg(DBG_ROUTE, "AODV_Core: Rreq.sendDone detecta sendpending no
                bloquejat.\n");
        }

        // Alliberem missatge
        dbg(DBG_ROUTE, "AODV_Core: alliberem sendPending.\n");
        sendPending = FALSE;

        // Comprovem enviament correcte.
        if (success){
                //enviament correcte
                dbg(DBG_ROUTE, "AODV_Core: enviament realitzat correctament per Rreq.
                Alliberem rreqTaskPending\n");
                rreqTaskPending = TASK_DONE;
        }else{
                // Comprovem si es el maxim nombre d'intents
                if (rreqNumTries > AODVR_NUM_TRIES){
                        // Descartar el packet
                        dbg(DBG_ROUTE, "AODV_Core: error, maxim nombre d'intents per al
                        msg rreq.\n");
                        rreqTaskPending = TASK_DONE;
                }else{
                        // Tornar a intentar
                        dbg(DBG_ROUTE, "AODV_Core: tornem a postejar la tasca
                        sendRreq().\n");
                        rreqTaskPending = TASK_REPOSTREQ;
                }
        }

        return SUCCESS;
    }


    event result_t SendRerr.sendDone(TOS_MsgPtr sentBuffer, bool success) {
        // Comprovem errors
        if (sendPending==FALSE){
                // Error sendPending hauria de ser TRUE
                dbg(DBG_ROUTE, "AODV_Core: Rerr.sendDone detecta sendpending no
                bloquejat.\n");
        }

        // Alliberem missatge
        sendPending = FALSE;
        dbg(DBG_ROUTE, "AODV_Core: alliberem sendPending.\n");

        // Comprovem enviament correcte.
        if (success){
                //enviament correcte
                dbg(DBG_ROUTE, "AODV_Core: enviament realitzat correctament per
                Rerr.\n");
                rerrTaskPending = TASK_DONE;
        }else{
                // Comprovem si es el maxim nombre d'intents
                if (rerrNumTries > AODVR_NUM_TRIES){
                        // Descartar el packet
                        dbg(DBG_ROUTE, "AODV_Core: error, maxim nombre d'intents per al
                        msg rerr.\n");
                        rerrTaskPending = TASK_DONE;
                }else{
                        // Tornar a intentar
                        dbg(DBG_ROUTE, "AODV_Core: tornem a postejar la tasca
```

```
                    sendRerr().\n");
                    rerrTaskPending = TASK_REPOSTREQ;
            }
    }

    return SUCCESS;
}

event TOS_MsgPtr ReceiveRreq.receive(TOS_MsgPtr receivedMsg) {
    //link rreq msg
    uint8_t length;
    AODV_Rreq_MsgPtr rreq_msg = call Payload2.linkPayload(receivedMsg, &length);
    AODV_Rreq_MsgPtr rreqToSend = call Payload2.linkPayload(rreqMsg, &length);
    AODV_Rreply_MsgPtr rreply_msg = call Payload2.linkPayload(rReplyMsg, &length);

    // Comprovem metrica
    if(rreq_msg->metric[0] > AODV_MAX_METRIC){
            // Assolits el maxim nombre d'intents
            dbg(DBG_ROUTE, "AODV_Core: error, maxim nombre de salts per al msg
            rreq.\n");
            return receivedMsg;
    }else{
            dbg(DBG_ROUTE, "AODV_Core: fem update del headerAODV (camp TTL).\n");
            rreq_msg->metric[0]++;
    }

    //comprobem la cache, per no repetir msg.
    if(call AODV_Cache.checkCache(rreq_msg->src, rreq_msg->rreqID, rreq_msg-
    >metric[0]) == FAIL){
            // Missatge ja rebut anteriorment
            dbg(DBG_ROUTE, "AODV_Core: descartant missatge(node:%d, RREQ, rreqID:%d),
            ja rebut anteriorment.\n", rreq_msg->src, rreq_msg->rreqID);
            return receivedMsg;
    }

    call AODV_Cache.updateCache(rreq_msg->dest,rreq_msg->src, call
    SingleHopMsg.getSrcAddress(receivedMsg), rreq_msg->rreqID, rreq_msg->destSeq,
    rreq_msg->metric[0], rreq_msg->flag);
    dbg(DBG_ROUTE, "AODV_Core: cache actualitzada.\n");


    //comprobem destinació del missatge
#if BS
    if((rreq_msg->dest == TOS_UART_ADDR) || (rreq_msg->dest == TOS_LOCAL_ADDRESS)){
            dbg(DBG_ROUTE, "AODV_Core: missatge amb destinació node LOCAL o
            UART.\n");
#else
    if(rreq_msg->dest == TOS_LOCAL_ADDRESS){
            dbg(DBG_ROUTE, "AODV_Core: missatge amb destinació node LOCAL.\n");
#endif
            // En destinació local

            // Fake inicial
            if (rreq_msg->destSeq == 0 && mySeq == 0){
                    dbg(DBG_ROUTE, "AODV_Core: incrementem el número de sequencia
                    aodv\n");
                    mySeq++;
            }

            // Aixo hauriem de poder ometre-ho
            if(rreq_msg->destSeq > mySeq ) {
                    // Update de seq number.
                    mySeq = rreq_msg->destSeq;
                    dbg(DBG_ROUTE, "AODV_Core: fem update del myseq a %d.\n",mySeq);
    }else if ((rreq_msg->destSeq < mySeq) && (rreq_msg->destSeq!=0)){
            // Missatge vell. Tenim número de sequencia guardat de valor més alt.
                    dbg(DBG_ROUTE, "AODV_Core: error, descartat missatge perque te
                    número de sequencia antic.\n");

                    // Indicador visual
                    return receivedMsg;
            }

            //generem el paquet Rreply
    if(rreplyTaskPending == TASK_DONE){
                    rreplyTaskPending = TASK_PENDING;
                    rreplyNumTries = 0;
```

```
                dbg(DBG_ROUTE, "AODV_Core: blooquejem rreplyTask i inicialitzem
                variables\n");
                rReplyDest = call SingleHopMsg.getSrcAddress(receivedMsg);

                rreply_msg->dest                    = rreq_msg->dest;
                rreply_msg->src                     = rreq_msg->src;
                rreply_msg->destSeq   = mySeq;
                rreply_msg->metric[0] = 0;
                rreply_msg->flag                    = ownFlag;

                // Update route info
                call AODV_Route.updateRouteInfo(rreq_msg->src, rReplyDest,
                rreq_msg->destSeq, rreq_msg->metric[0],mySeq, rreq_msg->flag);
                dbg(DBG_ROUTE, "AODV_Core: taula de rutes actualitzada\n");

                // aleatoritzant entrada al medi
                rreqRandomize = (uint8_t)((call Random.rand() & 0xff) %
                AODV_MAX_RAND);
                if(rreqRandomize){
                        dbg(DBG_ROUTE, "AODV_Core: entrada aleatoritzada al medi,
                        repost rreplyTask\n");
                        rreplyTaskPending = TASK_REPOSTREQ;
                        return receivedMsg;
                }

                //Posting task sendRreply
                dbg(DBG_ROUTE, "AODV_Core: posting sendRreply()\n");
                post sendRreply();
        }else{
                // Error tasca pendent.
                dbg(DBG_ROUTE, "AODV_Core: error, RreplyTask pendent no permet
                modificar parametres.\n");
                return receivedMsg;
        }
    }else{
        // El missatge te destinació no local. Cal reenviar-lo.
        dbg(DBG_ROUTE, "AODV_Core: missatge per reenviar.\n");
        if(rreq_msg->src == TOS_LOCAL_ADDRESS) {
                // Error hauria d'haver estat filtrat per la cache
                dbg(DBG_ROUTE, "AODV_Core: missatge hauria de ser filtrat per la
                cache\n");
                return receivedMsg;
        }

    if(rreqTaskPending == TASK_DONE){
                rreqTaskPending = TASK_PENDING;
                rreqNumTries = 0;

                rreqToSend->dest                    = rreq_msg->dest;
                rreqToSend->src                     = rreq_msg->src;
                rreqToSend->rreqID           = rreq_msg->rreqID;
                rreqToSend->srcSeq           = rreq_msg->srcSeq;
                rreqToSend->destSeq   = rreq_msg->destSeq;
                rreqToSend->metric[0] = rreq_msg->metric[0];
                rreqToSend->flag                    = rreq_msg->flag | ownFlag;

                dbg(DBG_ROUTE, "AODV_Core: bloquejem rreqTask i inicialitzem
                variables\n");
                // aleatoritzant entrada al medi
                rreqRandomize = (uint8_t)((call Random.rand() & 0xff) %
                AODV_MAX_RAND);

                #ifndef PLATFORM_PC
                if(rreqRandomize){
                        dbg(DBG_ROUTE, "AODV_Core: entrada aleatoritzada al medi,
                        repost rreqTask\n");
                        rreqTaskPending = TASK_REPOSTREQ;
                        return receivedMsg;
                }
                #endif

                // Posting task
                dbg(DBG_ROUTE, "AODV_Core: posting sendRreq.\n");
                post sendRreq();

        }else{
                // Error tasca pendent
```

```
                        dbg(DBG_ROUTE, "AODV_Core: error, RreqTask pendent no permet
                        modificar parametres.\n");
                }
        #if BS
        }
        #else
        }
        #endif

        return receivedMsg;
  }

    event result_t SendRreply.sendDone(TOS_MsgPtr sentBuffer, bool success) {
        // Comprovem errors
        if (sendPending==FALSE){
                // Error sendPending hauria de ser TRUE
                dbg(DBG_ROUTE, "AODV_Core: Rreply.sendDone detecta sendpending no
                bloquejat.\n");
        }

        // Alliberem missatge
        sendPending = FALSE;
        dbg(DBG_ROUTE, "AODV_Core: alliberem sendPending.\n");

        // Comprovem enviament correcte.
        if (success){
                //enviament correcte
                dbg(DBG_ROUTE, "AODV_Core: enviament realitzat correctament per
                Rreply.\n");
                rreplyTaskPending = TASK_DONE;
        }else{
                // Comprovem si es el maxim nombre d'intents
                if (rreplyNumTries > AODVR_NUM_TRIES){
                        // Descartar el packet
                        dbg(DBG_ROUTE, "AODV_Core: error, maxim nombre d'intents per al
msg rreply.\n");
                        rreplyTaskPending = TASK_DONE;
                }else{
                        // Tornar a intentar
                        dbg(DBG_ROUTE, "AODV_Core: tornem a postejar la tasca
                        sendRreply().\n");
                        rreplyTaskPending = TASK_REPOSTREQ;
                }
        }

        return SUCCESS;
    }

  event TOS_MsgPtr ReceiveRreply.receive(TOS_MsgPtr receivedMsg) {
        uint8_t length;
        AODV_Rreply_MsgPtr msg = call Payload2.linkPayload(receivedMsg, &length);

        // Update routeTable info
        call                    AODV_Route.updateRouteInfo(msg->dest,                   call
SingleHopMsg.getSrcAddress(receivedMsg), msg->destSeq, msg->metric[0],mySeq, msg->flag);
        dbg(DBG_ROUTE, "AODV_Core: fent update de RouteInfo. Llistant Taula\n");

#if BS
        if(msg->src == TOS_LOCAL_ADDRESS || msg->src == TOS_UART_ADDR){
                dbg(DBG_ROUTE, "AODV_Core: missatge procedent del node LOCAL o UART.\n");
#else
        if(msg->src == TOS_LOCAL_ADDRESS){
                dbg(DBG_ROUTE, "AODV_Core: missatge procedent del node LOCAL.\n");
#endif
                //Ja hem arribat a la destinació
                dbg(DBG_ROUTE, "AODV_C, ruta creada a la destinació %d\n", msg->dest);
        }else{
                if (rreplyTaskPending == TASK_DONE){
                        rreplyTaskPending = TASK_PENDING;
                        dbg(DBG_ROUTE, "AODV_Core: bloquejem rreplyTask i inicialitzem
                        variables\n");
                        //Busquem destinació de reply
                        rReplyDest = call AODV_Route.getReverseRoute(msg->src);
                        if (rReplyDest == INVALID_NODE_ID){
                                // Error al buscar ruta inversa pel reply
                                dbg(DBG_ROUTE, "AODV_Core: no trobem ruta inversa pel
                                rreply.\n");
```

```
                        rreplyTaskPending = TASK_DONE;
                    }else{
                        //metric erronia
                        call AODV_Route.updateRouteInfo(msg->src, rReplyDest, msg-
                        >destSeq, call AODV_Route.getReverseHops(msg->src), mySeq,
                        msg->flag);

                        rreplyNumTries = 0;

                        //Posting task
                        dbg(DBG_ROUTE, "AODV_Core: posting sendRreply.\n");

                        msg->flag = msg->flag | ownFlag;
                        //POSSIBLE_BUG
                        //rReplyMsg = receivedMsg;
                        memcpy(rReplyMsg, receivedMsg, sizeof(TOS_Msg));
                        /*FI DE POSSIBLE BUG ? */

                        post sendRreply();
                    }
                }else{
                    //error taskPending
                    dbg(DBG_ROUTE, "AODV_Core: error, RreplyTask pendent no permet
                    modificar parametres.\n");
                }
        #if BS
        }
        #else
        }
        #endif
        return receivedMsg;
    }

event TOS_MsgPtr ReceiveRerr.receive(TOS_MsgPtr receivedMsg) {
    // Link msg
        uint8_t length;
        AODV_Rerr_MsgPtr msg = call Payload2.linkPayload(receivedMsg, &length);

        // Aixo serveix per tornar a acceptar els nous missatges de rreq
        call AODV_Cache.deleteCache();
        dbg(DBG_ROUTE, "AODV_Core: borrem la cache de msg en bcast.\n");


        // Comprovem que tenim ruta afectada pel node
        if (call AODV_Route.getRouteIndex(msg->dest, TRUE) == INVALID_INDEX){
                // no hi han rutes passant pel node indicat
                dbg(DBG_ROUTE, "AODV_Core: error, no hi han rutes al node indicat.\n");
                return receivedMsg;
        }

        // Comprovem que tenim ruta afectada pel node
        if (call AODV_Route.getRouteIndex(msg->dest, FALSE) == INVALID_INDEX){
                // no hi han rutes passant pel node indicat
                dbg(DBG_ROUTE, "AODV_Core: error, no hi han rutes al node indicat2.\n");
                return receivedMsg;
        }

        // prosseguim si no hi ha tasca pendent
        if(rerrTaskPending == TASK_DONE){
                rerrTaskPending = TASK_PENDING;
                dbg(DBG_ROUTE, "AODV_Core: bloquejem rerrTask i inicialitzem
                variables\n");

                // Delete routeTable for node_id
                call AODV_Route.RemoveRoute(msg->dest);
                dbg(DBG_ROUTE, "AODV_Core: borrem rutes afectades pel node %d.\n", msg-
                >dest);

                // Posting  sendErr
                dbg(DBG_ROUTE, "AODV_Core: posting sendRerr.\n");


                //POSSIBLE_BUG
                //rErrMsg = receivedMsg;
                memcpy(rErrMsg, receivedMsg, sizeof(TOS_Msg));
                /*FI DE POSSIBLE BUG ? */
```

```
                post sendRerr();

                return receivedMsg;
    }else{
                //error taskPending
                        dbg(DBG_ROUTE, "AODV_Core: error, RerrTask pendent no permet
                        modificar parametres.\n");
                        return receivedMsg;
        }

  }

  event result_t Timer.fired() {
        //Definim prioritats !!! podriem situar aqui tambe un aleatoritzador.
    if(rreplyTaskPending == TASK_REPOSTREQ){
                dbg(DBG_ROUTE, "AODV_Core: Timer posts sendRreply().\n");
                rreplyTaskPending = TASK_PENDING;
                rreplyNumTries++;
                post sendRreply();
    }

    if(rerrTaskPending == TASK_REPOSTREQ){
                dbg(DBG_ROUTE, "AODV_Core: Timer posts sendRerr().\n");
                rerrTaskPending = TASK_PENDING;
                rerrNumTries++;
                post sendRerr();
    }

    if(rreqTaskPending == TASK_REPOSTREQ){
                dbg(DBG_ROUTE, "AODV_Core: Timer posts sendRreq().\n");
                rreqTaskPending = TASK_PENDING;
                rreqNumTries++;
                post sendRreq();
    }
        return SUCCESS;
  }

    command uint16_t getAODVseqNum(){
        return mySeq;
    }
}
```

H.5 AODV_PacketForwarder.nc

```
module AODV_PacketForwarder {
  provides {
      interface StdControl as Control;

      interface SendMHopMsg[uint8_t app];
      interface Receive[uint8_t app];
      interface Intercept[uint8_t app];
      interface Intercept as PromiscuousIntercept[uint8_t app];

        interface AODVMsg;
      interface MultiHopMsg;

        interface Payload2    as AODVPayload;
  }
  uses {
      interface SendMsg             as SingleHopSend;

      interface ReceiveMsg    as SingleHopReceive;          //from radio to AODV
                interface ReceiveMsg  as UARTReceiveMsg;    //from UART to AODV

      interface Payload2            as SingleHopPayload;
                interface MyQueue     as FindRouteQueue;

                interface Timer       as DiscoveryTimer;    //timer per esperar respostes

      interface SingleHopMsg;
      interface ReactiveRouter;
        interface AODV_Cache;
        interface Leds;
  }
}

implementation {
```

```
//////////////////////VARIABLES ////////////////////////////////////////////////////
    // Variables que emmagatzemen informació del msg en proces.
  TOS_Msg             msg_buf;
  TOS_MsgPtr   send_msg;
    uint8_t           sendIntent;           // # d'intent actual per send_msg.

    // Variables que controlen el funcionament global del programa.
    uint8_t aodv_seqnum;                    // número de sequencia AODV.
    bool route_repairing;     // Quan ens trobem en reparació es posen
                              tots els missatges en cua.
    bool sendPending;         // Quan hi ha enviament pendent es
                              posen tots els missatges en cua.

////////////////////////////////FUNCIONS/////////////////////////////////////////////
#ifdef PLATFORM_PC
    void dbgPacket(AODV_MsgPtr data) {
    uint8_t it;
        dbg(DBG_ROUTE, "SRV-C, Debugging Packet :\n");
        dbg(DBG_ROUTE, "      MSG_DBG:");
    for(it = 0; it < (24); it++)
              dbg_clear(DBG_ROUTE, "%02hhx ", ((uint8_t *)data)[it]);
    dbg(DBG_ROUTE, "\n");
  }
#endif

    //
    // Retorna un punter al missatge AODV de dins del missatge TOS_Msg
    //
  inline AODV_MsgPtr getAODVPtr(TOS_MsgPtr msg) {
    AODV_MsgPtr aodv_ptr;
        uint8_t length;
        aodv_ptr = call SingleHopPayload.linkPayload(msg, &length);
    return aodv_ptr;
  }

    //
    // Inicialitza les variables del modul.
    //
    void initVar(){
        atomic{
                send_msg                                = &msg_buf;
                route_repairing = FALSE;
                sendPending           = FALSE;
                sendIntent                      = 0;
                aodv_seqnum           = 0;
        }
    }

////////////////////////////AODVPAYLOAD/////////////////////////////////////////////
    command void * AODVPayload.linkPayload(TOS_MsgPtr msg, uint8_t *len) {
        AODV_MsgPtr aodvMsg;
        uint8_t length;
        aodvMsg = call SingleHopPayload.linkPayload(msg, &length);
        *len = length - AODV_HEADER_LEN;
        return aodvMsg->data;
    }

    ////////////////////////////// STDCONTROL
/////////////////////////////////////////////////////////////

  command result_t Control.init() {
        dbg(DBG_BOOT, "AODV_PacketForwarder inicialitza.\n");
        initVar();

        call Leds.init();
    return SUCCESS;
  }

  command result_t Control.start() {
        dbg(DBG_ROUTE, "AODVPF_NODE INICIAT\n");
        return SUCCESS;
  }

  command result_t Control.stop() {
        return SUCCESS;
  }
```

```
//////////////////////////MULTIHOPMSG///i///AODVMSG////////////////////////////////
    //
    //  Acces als diferents camps del missatge AODV.
    //
    // AODVHEADER: < [(src) (dest) (app) (length)] (seq) (ttl) (flag) >
  command wsnAddr MultiHopMsg.getSource(TOS_MsgPtr msg) {
    return getAODVPtr(msg)->mhop.src;
  }

  command wsnAddr MultiHopMsg.getDest(TOS_MsgPtr msg) {
    return getAODVPtr(msg)->mhop.dest;
  }

  command uint8_t MultiHopMsg.getApp(TOS_MsgPtr msg) {
    return getAODVPtr(msg)->mhop.app;
  }

  command uint8_t MultiHopMsg.getLength(TOS_MsgPtr msg) {
    return getAODVPtr(msg)->mhop.length;
  }

  command uint8_t AODVMsg.getSequenceNum(TOS_MsgPtr msg) {
    return getAODVPtr(msg)->seq;
  }

  command uint8_t AODVMsg.getTTL(TOS_MsgPtr msg) {
    return getAODVPtr(msg)->ttl;
  }

    command uint8_t AODVMsg.getFlag(TOS_MsgPtr msg){
        return getAODVPtr(msg)->flag;
    }

  command uint8_t AODVMsg.getNext(TOS_MsgPtr msg) {
    return call SingleHopMsg.getDestAddress(msg);
  }

//////////////////////////COMUNICACIONS/////////////////////////////////////////////////////
  default event result_t Intercept.intercept[uint8_t app](TOS_MsgPtr m, void *payload,
uint16_t len) {
    dbg(DBG_ROUTE, "PF, default intercept event\n");
        return SUCCESS;
  }

  default event result_t PromiscuousIntercept.intercept[uint8_t app](TOS_MsgPtr m, void
*payload, uint16_t len) {
    return SUCCESS;
  }

  default event TOS_MsgPtr Receive.receive[uint8_t app](TOS_MsgPtr m, void *payload,
uint16_t len) {
    return NULL;
  }

  command void * SendMHopMsg.getBuffer[uint8_t app](TOS_MsgPtr msg, uint16_t *len) {
    AODV_MsgPtr aodv_ptr;
        uint8_t len2;
    aodv_ptr = call SingleHopPayload.linkPayload(msg, &len2);
    *len = (uint16_t)(len2 - AODV_HEADER_LEN);
    return aodv_ptr->data;
  }


//////////////////////////////////TASKS/////////////////////////////////////////////////////
//
// Task per enviar el missatge. Busca ruta i envia, en cas d'error el posa a la cua de
//msg i s'inicia el routediscovery. Si s'ha assolit el nombre maxim d'intents (l'intent
//s'augmenta al posar a la cua) es descarta el paquet i s'envia un signal sendDone. Si
//el node es troba en route_repairing es desvia el trafic cap a la cua de msg.

    // Previous declaration of seguentMsg();
    void seguentMsg();

    task void enviarMsg(){
        AODV_MsgPtr aodv_msg  = getAODVPtr(send_msg);
        uint8_t appDest                              = aodv_msg->mhop.app;
```

```
        uint8_t msgDest                         = aodv_msg->mhop.dest;
        uint8_t nextHop;

        call Leds.greenToggle();

        dbg(DBG_ROUTE, "AODV_PF: task enviarMsg()\n");
        // Si node Base Station, i el node es destina a TOS_LOCAL, s'envia a la UART.
        #if BS
        if ((msgDest == (uint8_t)TOS_LOCAL_ADDRESS) ||(msgDest ==
(uint8_t)TOS_UART_ADDR)){
                        if (call SingleHopSend.send(TOS_UART_ADDR, (uint8_t)aodv_msg-
    >mhop.length + AODV_HEADER_LEN, send_msg) == SUCCESS){
                        // enviat satisfactoriament a la UART
                        dbg(DBG_ROUTE, "AODV_PF: missatge enviat a la UART.\n");
                        return;
                }else{
                        // error
                        dbg(DBG_ERROR, "AODV_PF: error enviant missatge a la UART.\n");
                }
        }
        #endif

        nextHop =  call ReactiveRouter.getNextHop(msgDest);

        if (nextHop != INVALID_NODE_ID){
                dbg(DBG_ROUTE,"AODV_PF: trobem ruta via node %d\n",nextHop);
                if (call SingleHopSend.send(nextHop, (uint8_t)aodv_msg->mhop.length +
                AODV_HEADER_LEN, send_msg) == SUCCESS){
                        dbg(DBG_ROUTE, "AODV_PF: enviant missatge a la destinació(%d) via
                        node(%d).\n",msgDest,nextHop);
                        aodv_seqnum++;
                        return;
                }else{
                        dbg(DBG_ERROR, "AODV_PF: error al enviar missatge amb
                        singlehopsend\n");
                        sendPending=FALSE;
                }
        }

        //call Leds.yellowToggle();

        dbg(DBG_ROUTE,"AODV_PF: no s'ha trobat ruta al node indicat. intent=%d\n",
        sendIntent);
        // Comprovem que no hagi assolit el maxim nombre d'intents.
        if (sendIntent >= MAX_INTENTS_GLOBAL){
                dbg(DBG_ERROR,"AODV_PF: hem assolit el maxim nombre d'intents\n");
#if BS
                if (aodv_msg->mhop.src == (uint8_t)TOS_LOCAL_ADDRESS || aodv_msg-
                >mhop.src == (uint8_t)TOS_UART_ADDR){
#else
                if (aodv_msg->mhop.src == (uint8_t)TOS_LOCAL_ADDRESS){
#endif
                        // aIXO EM SEMBLA Q SOBRA.
                        if ((msgDest != (uint8_t)TOS_BCAST_ADDR) && (nextHop !=
                        INVALID_NODE_ID)){
                                // enviem el routeError message
                                //call Leds.redToggle();
                                dbg(DBG_ERROR,"AODV_PF: enviant msg d'error a la destinació
                                %d.\n", nextHop);
                                call ReactiveRouter.SendRouteErr(nextHop);
                        }

                        if (msgDest != (uint8_t)TOS_BCAST_ADDR){
                                // enviem el routeError message
                                //call Leds.redToggle();
                                dbg(DBG_ERROR,"AODV_PF: enviant msg d'error a la destinació
                                %d.\n", msgDest);
                                call ReactiveRouter.SendRouteErr(msgDest);
                        }

                        dbg(DBG_ROUTE,"AODV_PF: signaling SendDone pel missatge amb maxim
                        num intents.\n");
                        if (signal SendMHopMsg.sendDone[appDest](send_msg, FAIL)!=
                        SUCCESS){
                                // enviat sendDone incorrectament
                                dbg(DBG_ERROR,"AODV_PF: enviament incorrecte del
                                sendDone.ERROR IMPORTANT\n");
```

```
                }else{
                        // enviat sendDone correctament
                        dbg(DBG_ERROR,"AODV_PF: enviant msg d'error a la destinació
                        %d.\n", msgDest);
                }
        }else{
                if (msgDest != (uint8_t)TOS_BCAST_ADDR){
                        // enviem el routeError message
                        dbg(DBG_ERROR,"AODV_PF: enviant msg d'error a la destinació
                        %d.\n", msgDest);
                        call ReactiveRouter.SendRouteErr(msgDest);
                }
        }
    }else{
        // Incrementem intents i posem en cua.
        dbg(DBG_ROUTE,"AODV_PF. incrementem intents\n");
        sendIntent++;

        dbg(DBG_ROUTE,"AODV_PF: encuem msg\n");
        if(call FindRouteQueue.enQueue(send_msg, sendIntent)!=SUCCESS){
                dbg(DBG_ROUTE,"AODV_PF: signal per missatge DESCARTAT.\n");
                signal SendMHopMsg.sendDone[appDest](send_msg, FAIL);
        }else{
                // Iniciem el descobriment de rutes
                dbg(DBG_ROUTE,"AODV_PF: Iniciem route discovery\n");
                route_repairing = TRUE;
                dbg(DBG_ROUTE, "AODV_PF, generant ruta al node %d\n",msgDest);
                call ReactiveRouter.generateRoute(msgDest, aodv_msg->flag);
                call DiscoveryTimer.start(TIMER_ONE_SHOT,
                DISCOVERY_PERIOD*(sendIntent+1));
        }
    }

    // Alliberem les variables del missatge
    dbg(DBG_ROUTE,"AODV_PF: Alliberem missatge. (sendPending)\n");
    send_msg = NULL;
    sendIntent = 0;
    sendPending = FALSE;

    // postejem la tasca de crida al seguent msg
    dbg(DBG_ROUTE,"AODV_PF: cridem seguentMsg().\n");
    seguentMsg();
}

task void buidarCua(){
    dbg(DBG_ROUTE,"AODV_PF: iniciem task buidarcua\n");

    // Bloquejem missatge send_msg.
    if (!sendPending){
        dbg(DBG_ROUTE,"AODV_PF: bloquejem el missatge (sendPending).\n");
        sendPending=TRUE;
    }else{
        dbg(DBG_ROUTE,"AODV_PF: sendpending es troba bloquejat\n");
        return;
    }

    // Agafem seguent missatge en cua i num d'intents.
    send_msg = call FindRouteQueue.getNext(&sendIntent);
    dbg(DBG_ROUTE,"AODV_PF:  agafem seguent msg en cua.\n");

    if (send_msg == NULL){
        dbg(DBG_ROUTE,"AODV_PF: no hi ha més msg en cua ??. (alliberem
        sendPending)\n");
        sendPending = FALSE;
        return;
    }

    // Fem tasca d'enviar missatge.
    dbg(DBG_ROUTE,"AODV_PF: postejem enviarMsg();.\n");
    if (!route_repairing){
        post enviarMsg();
    }
}

void seguentMsg(){

    if (route_repairing){
```

```
                dbg(DBG_ROUTE,"AODV_PF, estem en route repairing, esperem\n");
                return;
        }

        // Postejem tasca per quan acabi la tasca actual.
        dbg(DBG_ROUTE,"AODV_PF: postejem task buidarcua\n");
        post buidarCua();
    }

////////////////////////////COMUNICACIONS/////////////////////////////////////////
//
//  Enviar missatge a node 'dest' creant ruta fins al node indicat. (ttl control if
//ttl!= 0xff)
    //
    command result_t SendMHopMsg.sendTTL[uint8_t app](uint16_t dest, uint8_t len,
    TOS_MsgPtr msg, uint8_t flagfield){
        AODV_MsgPtr aodv_msg = getAODVPtr(msg);

        dbg(DBG_ROUTE, "AODV-Z dins SEndTTL\n");

        // Creació del header AODV_Msg
        dbg(DBG_ROUTE,"AODV_PF: creem header AODV\n");
    aodv_msg->mhop.src                      = (wsnAddr) TOS_LOCAL_ADDRESS;
    aodv_msg->mhop.length     = (uint8_t) len;
    aodv_msg->mhop.dest             = (wsnAddr) dest;
    aodv_msg->mhop.app                      = app;

    aodv_msg->seq                                   = aodv_seqnum;
        aodv_msg->ttl                               = DEFAULT_TTL;
        aodv_msg->flag                              = flagfield;
        dbg(DBG_ROUTE,"AODV_PF: header creat\n");

        // Comprovar si es BroadCast
        if (aodv_msg->mhop.dest ==  (uint8_t)TOS_BCAST_ADDR){
                dbg(DBG_ROUTE,"AODV_PF: posem msg en cache de bcastmsg\n");
                call AODV_Cache.updateCacheData(dest, aodv_msg->mhop.src, aodv_msg-
                >mhop.app, aodv_msg->seq, aodv_msg->flag);
        }

        // Bloquejem enviament
        if ((!sendPending) && (!route_repairing)){
                dbg(DBG_ROUTE,"AODV_PF: bloquejem variable msg\n");
                sendIntent            = 0;
                sendPending    = TRUE;
                send_msg                      = msg;
                dbg(DBG_ROUTE, "AODV_PF: posting enviarMsg();\n");
                post enviarMsg();
        }
        else{
                dbg(DBG_ROUTE,"AODV_PF: encuem msg perque variables msg estan
                ocupades\n");
                if (sendPending)
                        dbg(DBG_ERROR, "AODV_PF: sendPending culpable\n");
                if (route_repairing)
                        dbg(DBG_ERROR, "AODV_PF: route_repairing culpable\n");
                call FindRouteQueue.enQueue(msg,0);
        }

        return SUCCESS;
    }

    //
    //  Enviar missatge a node 'dest' creant ruta fins al node indicat. (ttl control if
        ttl!= 0xff)
    //
  command result_t SendMHopMsg.forwardTTL[uint8_t app](TOS_MsgPtr msg){
        AODV_MsgPtr aodv_msg = getAODVPtr(msg);

        // Comprovar si es BroadCast
        if (aodv_msg->mhop.dest ==  (uint8_t)TOS_BCAST_ADDR){
                dbg(DBG_ROUTE,"AODV_PF: posem msg en cache de bcastmsg.\n");
                call AODV_Cache.updateCacheData(aodv_msg->mhop.dest, aodv_msg->mhop.src,
                aodv_msg->mhop.app, aodv_msg->seq, aodv_msg->flag);
        }

        // Bloquejem enviament
        if ((!sendPending) && (!route_repairing)){
```

```
                dbg(DBG_ROUTE,"AODV_PF: bloquejem variable msg (sendPending)\n");
                sendIntent              = 0;
                sendPending    = TRUE;
                send_msg                        = msg;
                dbg(DBG_ROUTE, "AODV_PF: posting enviarMsg();\n");
                post enviarMsg();
        }else{
                dbg(DBG_ROUTE,"AODV_PF: encuem msg perque variables msg estan
                ocupades\n");
                call FindRouteQueue.enQueue(msg,0);
        }
        return SUCCESS;
    }


    //
    // Missatge a singlehopmanager enviat
    //
    event result_t SingleHopSend.sendDone(TOS_MsgPtr sentBuffer, result_t success) {
        uint8_t appDest                         = getAODVPtr(sentBuffer)->mhop.app;
        dbg(DBG_ROUTE,"AODV_PF: rebem signal sendDone.\n");

        sendPending = FALSE;
        dbg(DBG_ROUTE, "AODV_PF: Alliberant missatge (sendPending)\n");

        // Comprovar si es missatge propi o es forwarded.
#if BS
        if (getAODVPtr(sentBuffer)->mhop.src != TOS_LOCAL_ADDRESS &&
getAODVPtr(sentBuffer)->mhop.src != TOS_UART_ADDR){
#else
        if (getAODVPtr(sentBuffer)->mhop.src != TOS_LOCAL_ADDRESS){
#endif
                if (((success == FAIL) || !(sentBuffer->ack)) && (sentBuffer->addr !=
                TOS_BCAST_ADDR)){
                        // Invalidem ruta
                        dbg(DBG_ROUTE,"AODV_PF: invalidem ruta perque ha fallat\n");
                        call ReactiveRouter.invalidateRoute((wsnAddr)sentBuffer->addr);

                        // Posem msg en cua
                        dbg(DBG_ROUTE,"AODV_PF: posem missatge en cua EN %d\n",
                        sendIntent);
                        if (call FindRouteQueue.enQueue(sentBuffer, sendIntent) !=
                        SUCCESS){
                                dbg(DBG_ROUTE,"AODV_PF: fem signal pq ha fallat
                                l'enqueueing\n");

                                call ReactiveRouter.SendRouteErr((wsnAddr)sentBuffer-
                                >addr);
                        }
                        // Cal invalidar la ruta si falla.
                        dbg(DBG_ROUTE,"AODV_PF: no es missatge local. no cal passar el
                        signal. Alliberem missatge (sendPending)\n");
                }
                post buidarCua();
                return SUCCESS;
        }

        // Comprovar enviament correcte.
        if (((success == FAIL) || !(sentBuffer->ack)) && (sentBuffer->addr !=
        TOS_BCAST_ADDR)){
                // Invalidem ruta
                dbg(DBG_ROUTE,"AODV_PF: invalidem ruta perque ha fallat\n");
                call ReactiveRouter.invalidateRoute((wsnAddr)sentBuffer->addr);

                // Posem msg en cua
                dbg(DBG_ROUTE,"AODV_PF: posem missatge en cua EN %d\n", sendIntent);
                if (call FindRouteQueue.enQueue(sentBuffer, sendIntent) != SUCCESS){
                        dbg(DBG_ROUTE,"AODV_PF: fem signal pq ha fallat l'enqueueing\n");
                        //call ReactiveRouter.SendRouteErr(getAODVPtr(sentBuffer)-
                        >mhop.dest);
                        call ReactiveRouter.SendRouteErr((wsnAddr)sentBuffer->addr);
                        if (signal SendMHopMsg.sendDone[appDest](sentBuffer, FAIL) !=
                        SUCCESS){
                                // no s'ha enviat el sendDone.
                                dbg(DBG_ERROR,"AODV_PF: Error al enviar sendDone. Alliberem
                                missatge (sendPending)\n");
                                post buidarCua();
```

```
                        return FAIL;
                }
        }

    }else   if (sentBuffer->addr == TOS_BCAST_ADDR){
            dbg(DBG_ROUTE,"AODV_PF: destinacio bcast pel sendDone\n");
            if (signal SendMHopMsg.sendDone[appDest](sentBuffer, success) !=
            SUCCESS){
                // no s'ha enviat el sendDone.
                dbg(DBG_ERROR,"AODV_PF: Error al enviar sendDone.\n");
                return FAIL;
            }

    }else{
            // Caldria moure aixo doncs si falla, ja no s'accepta més.
            dbg(DBG_ROUTE,"AODV_PF: incrementem el num de sequencia AODV\n");
            aodv_seqnum++;

            if (signal SendMHopMsg.sendDone[appDest](sentBuffer, success) != SUCCESS)
{
                // no s'ha enviat el sendDone.
                dbg(DBG_ERROR,"AODV_PF: Error al enviar sendDone.\n");
                return FAIL;
            }
    }

    // Postejem el seguent missatge
    dbg(DBG_ROUTE,"AODV_PF: Postejem buidarCua\n");
    aodv_seqnum++;
    post buidarCua();
    return SUCCESS;
  }

  //
  //  Signal per indicar a capes superiors(app) que s'ha enviat el missatge (SendTTL)
  //
 default event result_t SendMHopMsg.sendDone[uint8_t app](TOS_MsgPtr sentBuffer,
   result_t success) {
   return SUCCESS;
 }

  // Recepció de missatges provinents de la UART  (AODV_WSN)
  //
  //
  event TOS_MsgPtr UARTReceiveMsg.receive(TOS_MsgPtr msg){
  TOS_MsgPtr retmsg         = msg;
     AODV_MsgPtr aodv_msg  = getAODVPtr(msg);
     uint8_t appDest                      = aodv_msg->mhop.app;

     // modifiquem els parametres q no modifica la UART. Evita errors.
     msg->crc = 1;
     msg->ack = 1;
     // Incrementem num sequencia
     aodv_seqnum++;

     // Comprovem destinació
  if ((msg->addr == (wsnAddr) TOS_LOCAL_ADDRESS) || (msg->addr == (wsnAddr)
TOS_BCAST_ADDR)) {
            // fem signal a aplicació.
            retmsg = signal Receive.receive[appDest](msg, aodv_msg->data, aodv_msg-
            >mhop.length);
  }

     //L'aplicació ha de retornar NULL si vol que es reenvii el missatge.
     if (retmsg != NULL){
            dbg(DBG_ROUTE,"AODV_PF: creant missatge ... . si necessari (hauria de
            construir-se a l'aplicacio java).\n");


            //call Leds.greenToggle();

            // Bloquejem enviament
            if ((!sendPending) && (!route_repairing)){
                dbg(DBG_ROUTE,"AODV_PF: bloquejem variable missatge (sendPending)
                1 .\n");
                sendIntent            = 0;
                sendPending    = TRUE;
```

```
                        //POSSIBLE_BUG
                        send_msg                        = msg;
                        //memcpy(send_msg, msg, sizeof(TOS_Msg));

                        dbg(DBG_ROUTE, "AODV_PF: posting enviarMsg();\n");
                        post enviarMsg();
                }else{
                        dbg(DBG_ROUTE,"PF, encuem msg pq esta ocupat\n");
                        call FindRouteQueue.enQueue(msg,0);
                }

        }
        return retmsg;
    }

    //
    // Recepció de missatges (el missatge retornat sera enviat a AODV_C si es diferent
de NULL).
    //
    event TOS_MsgPtr SingleHopReceive.receive(TOS_MsgPtr msg){
    TOS_MsgPtr retmsg         = msg;
    AODV_MsgPtr aodv_msg      = getAODVPtr(msg);
        uint8_t appDest                         = aodv_msg->mhop.app;
        uint8_t nodeDest                        = aodv_msg->mhop.dest;

        // update del camp ttl
        dbg(DBG_ROUTE,"PF, fem update del camp TTL\n");
        aodv_msg->ttl--;

        // is bcast ?
        if (aodv_msg->mhop.dest == (uint8_t)TOS_BCAST_ADDR){
                dbg(DBG_ROUTE,"PF, cal comprovar la cache (msg bcast)\n");
                // missatge en bcast cal comprovar cache.
                if(call AODV_Cache.checkCacheData(aodv_msg->mhop.dest, aodv_msg-
                >mhop.src, aodv_msg->mhop.app, aodv_msg->seq) == FAIL){
                        dbg(DBG_ROUTE,"PF, trobat msg a cache\n");
                        return NULL;
                }else{
                        dbg(DBG_ROUTE,"PF, incloem msg a cache\n");
                        call AODV_Cache.updateCacheData(aodv_msg->mhop.dest, aodv_msg-
                        >mhop.src, aodv_msg->mhop.app, aodv_msg->seq, aodv_msg->flag);
                }
        }

#if BS
        // Per recepcionar els missatges de la UART
        if((msg->addr == TOS_UART_ADDR) ||  (nodeDest == (uint8_t)TOS_UART_ADDR) ||
        (nodeDest == (uint8_t)TOS_BCAST_ADDR)) {
                dbg(DBG_ROUTE,"PF, signal al receive de la UART\n");
                return signal Receive.receive[appDest](msg, aodv_msg->data,
                (uint16_t)aodv_msg->mhop.length);
        }
#endif

        // Si el missatge te com a destinació SingleHop el node local
    if ((msg->addr == TOS_LOCAL_ADDRESS) || (msg->addr == TOS_BCAST_ADDR)) {
                dbg(DBG_ROUTE,"PF, destinacio singleHop local\n");
                // Fem signal al promiscuous intercept (el node te destinacio al node
                LOCAL pero no es la destinació final)
      if (signal Intercept.intercept[appDest](msg, aodv_msg->data, (uint16_t)aodv_msg-
        >mhop.length) == SUCCESS) {
                        // Si el missatge te com a destinació MultiHop el node local
                        if(nodeDest == (wsnAddr) TOS_LOCAL_ADDRESS || nodeDest ==(wsnAddr)
                        TOS_BCAST_ADDR) {
                                dbg(DBG_ROUTE,"PF, destinació multihop local\n");
                                retmsg = signal Receive.receive[appDest](msg, aodv_msg-
                                >data, (uint16_t)aodv_msg->mhop.length);

                                if ((retmsg == NULL) || (nodeDest != (wsnAddr)
                                TOS_BCAST_ADDR)){
                                        // missatge utilitzat
                                        return NULL;
                                }else{
                                        // Seguim enviant el BCAST msg
                                }
         }else{
```

```
                        //
                        dbg(DBG_ROUTE,"AODV_PF: cal enviar missatge a dest %d\n",
                        nodeDest);
                }

                // Comprovar que no hagi arribat al maxim de salts TTL
                if (!(aodv_msg->ttl>0)){
                        dbg(DBG_ROUTE,"PF, maxim TTL\n");
                        return NULL;
                }

                // Intentem enviar el missatge a la destinació multihop

                // Bloquejem enviament
                if ((!sendPending) && (!route_repairing)){
                        dbg(DBG_ROUTE,"AODV_PF: bloquejem variable missatge
                        (sendPending) 2.\n");

                        sendIntent            = 0;
                        sendPending    = TRUE;
                        send_msg                      = msg;

                        dbg(DBG_ROUTE, "AODV_PF: pot haver-hi problemes amb
                        send_msg = msg\n");

                        dbg(DBG_ROUTE, "AODV_PF: posting enviarMsg();\n");
                        post enviarMsg();
                }else{
                        dbg(DBG_ROUTE,"PF, encuem msg pq esta ocupat\n");
                        call FindRouteQueue.enQueue(msg,0);
                }
            }
    } else {
            // Fem signal al promiscuous intercept (el node no te destinacio al node
            LOCAL pero s'ha interceptat)
            signal PromiscuousIntercept.intercept[appDest](msg, aodv_msg->data,
(uint16_t)aodv_msg->mhop.length);
    }
        return NULL;
    }

////////////////////////////////// TIMERS//////////////////////////////////////////
    //
    // Timer d'espera de respostes. Implementat per extreure de cua, sense donar
        prioritat al node previ.

    event result_t DiscoveryTimer.fired() {
        dbg(DBG_ROUTE,"PF, discovery timer fired. Alliberem route_repairing.\n");
        route_repairing = FALSE;
        seguentMsg();
        dbg(DBG_ROUTE, "Timer for discovery fired\n");
        return SUCCESS;
    }

}
```

H.6. AODV_Tables.nc

```
module AODV_Tables {
        provides {
                interface StdControl;
                interface AODV_Route;
                interface AODV_Cache;
        }
        uses {
                interface SequenceNumber;     //Actualitza el num de
                                              sequencia a cada nova recepció del node.
        }
}

implementation {
    int i;
    AODV_Route_Table   routeTable[AODV_RTABLE_SIZE]; // Per emmagatzemar rutes creades
    AODV_Route_Cache   rreqCache[AODV_RQCACHE_SIZE]; // Per missatges de control AODV
                                                     enviats en BCAST
    AODV_Data_Cache    aodvCache[AODV_DATACACHE_SIZE];// Per missatges de dades enviats
                                                     en BCAST
```

```
    command result_t StdControl.init() {
// inicialització de la taula de rutes (de moment no s'utilitza el lifetimeseq i flag)
        for(i = 0; i< AODV_RTABLE_SIZE; i++){
                routeTable[i].dest                  = INVALID_NODE_ID;
                routeTable[i].nextHop               = INVALID_NODE_ID;
                routeTable[i].destSeq               = 0;
                routeTable[i].numHops               = 0;
                routeTable[i].lifetimeSeq    = 0;   //Per identificar la ruta més vella.
                routeTable[i].flag                          = 0;
        }
        // inicialització de la cache de Rreq (podriem prescindir de numHops)
        for(i = 0; i< AODV_RQCACHE_SIZE; i++){
                rreqCache[i].dest     = INVALID_NODE_ID;
                rreqCache[i].src      = INVALID_NODE_ID;
                rreqCache[i].nextHop  = INVALID_NODE_ID;
                rreqCache[i].destSeq  = 0;
                rreqCache[i].numHops  = 0;
        }
        //Inicialització de la cache de dades (tots requerits)
        for(i = 0; i< AODV_RQCACHE_SIZE; i++){
                aodvCache[i].dest     = INVALID_NODE_ID;
                aodvCache[i].src      = INVALID_NODE_ID;
                aodvCache[i].app                    = 0;
                aodvCache[i].seq                    = 0;
        }
        return SUCCESS;
    }

    command result_t StdControl.start() {
        return SUCCESS;
    }

    command result_t StdControl.stop() {
        return SUCCESS;
    }

#ifdef PLATFORM_PC
    //
    //  Fa un print de la taula de rutes
    //
    void printTaula(){
        dbg(DBG_USR3, "TAULA DE RUTES AL NODE %d\n", TOS_LOCAL_ADDRESS);
        for(i=0; i< AODV_RTABLE_SIZE; i++){
                dbg(DBG_USR3, "ROUTE at Pos(%d): dest=%d, viaNode=%d, destSeq=%d,
numHops=%d\n", i, routeTable[i].dest,routeTable[i].nextHop, routeTable[i].destSeq,
routeTable[i].numHops);
        }
    }

    //
    //  Fa un print de la taula de cache
    //
    void printCache(){
        dbg(DBG_USR3, "TAULA DE CACHE AL NODE %d\n", TOS_LOCAL_ADDRESS);
        for(i=0; i< AODV_RQCACHE_SIZE; i++){
                dbg(DBG_USR3, "POS(%d)::dest=%d, from node=%d, via node %d destSeq=%d,
numHops=%d\n", i, rreqCache[i].dest,rreqCache[i].src, rreqCache[i].nextHop,
rreqCache[i].destSeq, rreqCache[i].numHops);
        }
    }
#endif

    //
    // Sobreescriu la ruta de la posicio 'indx' desplacant totes les rutes un posició.
    //
    result_t deleteRTableEntry(int indx){
        dbg(DBG_USR3, "AODV_T: Eliminada ruta(%d), amb destinació %d i nexthop %d.\n", i,
        routeTable[i].dest, routeTable[i].nextHop);

        for(i = 0; i< AODV_RQCACHE_SIZE; i++){
                if(rreqCache[i].dest == routeTable[i].dest){
                        rreqCache[i].dest     = INVALID_NODE_ID;
                        rreqCache[i].nextHop  = INVALID_NODE_ID;
                        rreqCache[i].destSeq  = 0;
                        rreqCache[i].numHops  = 0;
                }
```

```
        }

        for(i = indx; i< AODV_RTABLE_SIZE-1; i++) {
                if(routeTable[i+1].dest != INVALID_NODE_ID){
                        routeTable[i] = routeTable[i+1];
                }else{
                        routeTable[i].dest                  = INVALID_NODE_ID;
                        routeTable[i].nextHop               = INVALID_NODE_ID;
                        routeTable[i].destSeq               = 0;
                        routeTable[i].numHops               = 0;
                        routeTable[i].lifetimeSeq    = 0;
                        routeTable[i].flag                          = 0;
                        break;
                }
        }
        return SUCCESS;
    }

    //
    //  Elimina les rutes amb nexthop a 'nextnode'. Retorna el num de rutes eliminades.
    //
    command uint8_t AODV_Route.RemoveRoute(wsnAddr nextnode) {
        uint8_t n_eliminades=0;
        for(i = 0; i<AODV_RTABLE_SIZE; i++){
                // Rutes que tinguin seguent salt a 'nextnode'
                if(routeTable[i].nextHop == nextnode || (routeTable[i].dest ==
                nextnode)){
                        dbg(DBG_USR2, "AOVD_T, borrant posicio %d\n", i);
                        deleteRTableEntry(i);
                        n_eliminades++;
                }
        }
        dbg(DBG_USR3, "S'han eliminat %d rutes passant pel node %d.\n",n_eliminades,
        nextnode);
        return n_eliminades;
    }

    //
    //  Invalida les rutes amb nexthop a 'node'
    //
    command result_t AODV_Route.invalidateRoute(wsnAddr node){
        uint8_t n_invalidada=0;
        for(i = 0; i<AODV_RTABLE_SIZE; i++){
                if(routeTable[i].nextHop == node){
                        dbg(DBG_USR3, "S'han invalidat ruta(%d) amb destinacio %d i
                        nexthop %d.\n",i, routeTable[i].dest, node);
                        routeTable[i].destSeq=INVALID_INDEX;
                        routeTable[i].numHops               = INVALID_INDEX;
                        routeTable[i].lifetimeSeq    = 0;
                        routeTable[i].flag                          = 0;
                        n_invalidada++;
                }
        }
        dbg(DBG_USR3, "S'han invalidat %d rutes.\n",n_invalidada);
#ifdef PLATFORM_PC
        printTaula();
#endif
        return SUCCESS;
    }

    //
    //  Retorna l'index de la ruta a la destinació 'dest'
    //
    uint8_t getRTableIndex(wsnAddr dest){
        for(i=0; i< AODV_RTABLE_SIZE; i++){
                if(routeTable[i].dest == dest){
                        return i;
                }
        }
        return INVALID_INDEX;
    }

    //
    //  Retorna el num de rutes totals en la taula
    //
    uint8_t getRTableLen(){
        uint8_t length=0;
```

```
        for(i=0; i< AODV_RTABLE_SIZE; i++){
                if(routeTable[i].dest != INVALID_NODE_ID){
                        length++;
                }
        }
        return length;
    }

    //
    //  Indica l'index de la ruta més vella. (la que te destSeq més baix).
    //
    uint8_t getOlderRoute(){
        uint8_t ruta = 0;
        for(i=1; i< AODV_RTABLE_SIZE; i++){
                if(routeTable[i].flag == ALTA_MOBILITAT)
                        ruta = i;
                if(routeTable[i].lifetimeSeq < routeTable[ruta].lifetimeSeq)
                        ruta = i;
                if(ruta != 0)
                        return ruta;
        }
        return ruta;
    }


    //
    //  Passats uns parametres, inclou / actualitza la ruta a la taula.
    //
    command result_t AODV_Route.updateRouteInfo(wsnAddr dest, wsnAddr nextHop, uint16_t
    destSeq, uint8_t numHops, uint16_t mySeq, uint8_t flag) {
        //dbg(DBG_USR2, "UPDATING ROUTE INFO\n");
#ifdef PLATFORM_PC
        printTaula();
#endif

        if (flag == ALTA_MOBILITAT){
                dbg(DBG_USR3, "node d'alta mobilitat %d, %d\n", flag, (flag &
                ALTA_MOBILITAT));
        }else{
                dbg(DBG_USR3, "node de baixa mobilitat %d, %d\n", flag, (flag &
                ALTA_MOBILITAT));
        }

        i = getRTableIndex(dest);
        if (i==INVALID_INDEX){
                //Nova ruta
                //dbg(DBG_USR2, "Ruta previa no trobada, nou via node\n");
                if (getRTableLen() > AODV_RTABLE_SIZE){
                        // no tenim lloc a la taula --> Eliminem la ruta menys utilitzada
                        i = getOlderRoute();
                }else{
                        // tenim lloc a la taula
                        i=getRTableLen();
                }
        }else{
                if ((routeTable[i].flag & ALTA_MOBILITAT) == ALTA_MOBILITAT){
                        //dbg(DBG_USR2, "AODV_T, ruta sera actualitzada sempre doncs es
                        d'alta mobilitat\n");

                }else if ((routeTable[i].destSeq == INVALID_INDEX) &&
                        (routeTable[i].numHops == INVALID_INDEX)){
                        //dbg(DBG_USR2, "AODV_T, recuperant una ruta invalidada amb el
                        mateix valor\n");
                }else  if ((routeTable[i].destSeq >= destSeq) ||
                ((routeTable[i].destSeq == destSeq) && (numHops >
                routeTable[i].numHops))){
                        //tot i aixo fem un update del camp lifetimeSeq
                        routeTable[i].lifetimeSeq = mySeq;
                        return SUCCESS;
                }else{

                        if (routeTable[i].numHops == 0xFF ){
                                routeTable[i].numHops= numHops;
                        }
                }
        }
```

```
        //Afegim la ruta
        routeTable[i].dest                  = dest;
        routeTable[i].destSeq = destSeq;
        routeTable[i].nextHop = nextHop;
        routeTable[i].numHops = numHops;
        routeTable[i].lifetimeSeq = mySeq;
        routeTable[i].flag                  = flag;
#ifdef PLATFORM_PC
        printTaula();
#endif
        return SUCCESS;
    }

    //
    //  Retorna el node via per arribar a destinació.
    //
    command wsnAddr AODV_Route.getNextHop(wsnAddr dest){
#ifdef PLATFORM_PC
        printTaula();
#endif
        for(i =0; i <AODV_RTABLE_SIZE; i++){
         if((routeTable[i].dest == dest)&& (routeTable[i].numHops != INVALID_NODE_ID)){
                        dbg(DBG_USR3, "AODV_T: Ruta(%d): va a %d via node %d\n", i, dest,
routeTable[i].nextHop);
                        return routeTable[i].nextHop;
         }
        }
        dbg(DBG_USR3, "AODV_T:No hi ha ruta al node %d\n", dest);
        return INVALID_NODE_ID;
    }

    //  Retorna el destSeq per a la ruta a la posicio 'ind' de la taula

    command uint8_t AODV_Route.getRouteSeqNum(uint8_t ind){
        return routeTable[i].destSeq;
    }

    command uint8_t AODV_Route.getRouteIndex(wsnAddr node, bool dest){
#ifdef PLATFORM_PC
        printTaula();
#endif
        if (dest){
                return getRTableIndex(dest);
        }else{
                for(i=0; i< AODV_RTABLE_SIZE; i++){
                        //dbg(DBG_USR2, "%d nexthop = %d\n",i,routeTable[i].nextHop);
                        if(routeTable[i].nextHop == node || routeTable[i].dest == node){
                                return i;
                        }
                }
                return INVALID_INDEX;
        }
    }
//Actualitzem el num de sequencia de la ruta, pq hem rebut dades noves per aquesta ruta.

    event void SequenceNumber.updateSeqNum(wsnAddr addr, uint8_t seqNum, uint16_t
    mySeq){
        //dbg(DBG_USR2, "AODV, fent update del #seq de la ruta al node %d\n",addr);
        i = getRTableIndex(addr);
        routeTable[i].lifetimeSeq = mySeq;
        if (routeTable[i].destSeq < seqNum){
                routeTable[i].destSeq=seqNum;
        }
    }

        //      Construeix una ruta inversa al node 'nodeSrc' revisant la cache.

    command uint8_t AODV_Route.getReverseRoute(wsnAddr nodeSrc){
#ifdef PLATFORM_PC
        printCache();
#endif
        for(i=0; i< AODV_RQCACHE_SIZE; i++){
                if(rreqCache[i].dest == INVALID_NODE_ID){
                        dbg(DBG_USR3, "AODV_T: No hi han msg a la cache.\n");
                        return INVALID_NODE_ID;
                }
                if(rreqCache[i].src == nodeSrc){
```

```
                    dbg(DBG_USR3, "AODV_T: Trobada ruta inversa al node %d via node
                    %d.\n", nodeSrc, rreqCache[i].nextHop);
                    return rreqCache[i].nextHop;
            }
        }
        dbg(DBG_USR3, "AODV_T: No trobem entrada per src %d\n", nodeSrc);
        return INVALID_NODE_ID;
    }

    command uint8_t AODV_Route.getReverseHops(wsnAddr nodeSrc){
        for(i=0; i< AODV_RQCACHE_SIZE; i++){
            if(rreqCache[i].dest == INVALID_NODE_ID){
                    dbg(DBG_USR3, "AODV_T: No hi han msg a la cache.\n");
                    return INVALID_NODE_ID;
            }
            if(rreqCache[i].src == nodeSrc){
                    dbg(DBG_USR3, "AODV_T: Trobada ruta inversa al node %d via node
                    %d.\n", nodeSrc, rreqCache[i].nextHop);
                    return rreqCache[i].numHops;
            }
        }
        dbg(DBG_USR3, "AODV_T: No trobem entrada per src %d\n", nodeSrc);
        return INVALID_NODE_ID;
    }

    //  Busca a la cache per veure si el missatge rebut es el mateix.
    command result_t AODV_Cache.checkCache(wsnAddr src, uint16_t rreqID, uint8_t
numHops) {
#ifdef PLATFORM_PC
        printCache();
#endif
        //
        for(i=0; i< AODV_RQCACHE_SIZE; i++){
            if(rreqCache[i].src == src){
                    if (rreqCache[i].rreqID > rreqID){
                            dbg(DBG_ROUTE, "entrada %d, te rreqID=%d, msg actual=
                            %d\n", i, rreqCache[i].rreqID, rreqID);
                            return FAIL;
                    }
                    if (rreqCache[i].rreqID == rreqID && rreqCache[i].numHops <
                    numHops){
                            dbg(DBG_ROUTE, "entrada %d, te rreqID=%d=msg actual, pero
                            menys salts %d\n", i, rreqCache[i].rreqID, numHops);
                            return FAIL;
                    }
            }
        }
        return SUCCESS;
    }

        //      Inclou missatge rebut a la cache.

    command result_t AODV_Cache.updateCache(wsnAddr dest, wsnAddr src, wsnAddr nextHop,
    uint16_t rreqID, uint16_t destSeq, uint8_t numHops, uint8_t flag) {
        int endIndex                    = -1;
        int replaceIndex        = -1;
        #ifdef PLATFORM_PC
        printCache();
        #endif
        for(i=0; i< AODV_RQCACHE_SIZE; i++){
            if(rreqCache[i].dest == INVALID_NODE_ID){
                    endIndex = i;
                    break;
            }
            if(rreqCache[i].src == src){
                    replaceIndex = i;
                    break;
            }
        }
        if(replaceIndex != -1){
            for(i=replaceIndex; i< AODV_RQCACHE_SIZE-1; i++){
                    if(rreqCache[i+1].dest != INVALID_NODE_ID){
                            rreqCache[i] = rreqCache[i+1];
                    }else{
                            break;
                    }
            }
```

```
        }else{
                if(endIndex == -1){
                        for(i=0; i< AODV_RQCACHE_SIZE-1; i++){
                                rreqCache[i] = rreqCache[i+1];
                        }
                }
        }
        rreqCache[i].dest            = dest;
        rreqCache[i].src                     = src;
        rreqCache[i].nextHop  = nextHop;
        rreqCache[i].rreqID   = rreqID;
        rreqCache[i].destSeq  = destSeq;
        rreqCache[i].numHops  = numHops;

        dbg(DBG_USR3, "AODV_T: CANVIS EN LA CACHE: Ruta(%d)= dest %d, src %d, nextHop %d,
        rreqID %d, ...\n",
        i,dest,src,nextHop, rreqID);
        #ifdef PLATFORM_PC
        printCache();
        #endif
        return SUCCESS;

    }


    //  Busca a la cache per veure si el missatge AODV_data rebut es el mateix, quan
    s'envia en flooding
    command result_t AODV_Cache.checkCacheData(wsnAddr dest, wsnAddr src, uint8_t app,
uint8_t aodvSeq) {
        for(i=0; i< AODV_DATACACHE_SIZE; i++){
                if((aodvCache[i].dest == dest) && (aodvCache[i].src == src) &&
(aodvCache[i].app == app) && (aodvCache[i].seq == aodvSeq)){
                        dbg(DBG_USR3, "AT, trobat a la cache\n");
                        return FAIL;
                }
        }
        return SUCCESS;
    }

    //  Inclou missatge rebut a la cache de AODV_data

    command result_t AODV_Cache.updateCacheData(wsnAddr dest, wsnAddr src, uint8_t app,
    uint8_t aodvSeq, uint8_t flag) {
        int endIndex                 = -1;
        int replaceIndex      = -1;
        for(i=0; i< AODV_DATACACHE_SIZE; i++){
                if(aodvCache[i].dest == INVALID_NODE_ID){
                        endIndex = i;
                        break;
                }
                if(aodvCache[i].src == src){
                        replaceIndex = i;
                        break;
                }
        }
        if(replaceIndex != -1){
                for(i=replaceIndex; i< AODV_DATACACHE_SIZE-1; i++){
                        if(aodvCache[i+1].dest != INVALID_NODE_ID){
                                aodvCache[i] = aodvCache[i+1];
                        }else{
                                break;
                        }
                }
        }else{
                if(endIndex == -1){
                        for(i=0; i< AODV_DATACACHE_SIZE-1; i++){
                                aodvCache[i] = aodvCache[i+1];
                        }
                }
        }
        aodvCache[i].dest     = dest;
        aodvCache[i].src      = src;
        aodvCache[i].app                     = app;
        aodvCache[i].seq                     = aodvSeq;
        return SUCCESS;

    }
```

```
    command result_t AODV_Cache.deleteCache(){
        // inicialització de la cache de Rreq
        for(i = 0; i< AODV_RQCACHE_SIZE; i++){
                rreqCache[i].dest     = INVALID_NODE_ID;
                rreqCache[i].nextHop  = INVALID_NODE_ID;
                rreqCache[i].destSeq  = 0;
                rreqCache[i].numHops  = 0;
        }
        return SUCCESS;
    }
}
```

H.7. SingleHopManager.nc

```
configuration SingleHopManager
{
   provides {
      interface StdControl as Control;
      interface SendMsg[uint8_t id];
      interface ReceiveMsg[uint8_t id];     //interface dirigit a AODV_Core [id=0x0d,
                                            0x0e, 0x0f,]
      interface ReceiveMsg as PromiscuousReceiveMsg[uint8_t id];
 //interface dirigit a AODV_PacketForwarder [id=0x09, 0x99, 0x7E(UART),]
      interface Payload2;
      interface SequenceNumber;
      interface SingleHopMsg;

   }
     uses {
               command uint16_t getAODVseqNum();
     }
}

implementation {
  components SingleHopManagerM, LedsC,

#if BS
    PromiscuousCommUART as Comm;
#else
    PromiscuousCommNoUART as Comm;
#endif

#ifndef PLATFORM_PC
  components CC2420RadioC;
#endif

    Payload2   =  SingleHopManagerM.Payload2;
  PromiscuousReceiveMsg =  SingleHopManagerM.PromiscuousReceiveMsg;
  SequenceNumber =  SingleHopManagerM.SequenceNumber;
  SingleHopMsg  =  SingleHopManagerM.SingleHopMsg;
  Control =  SingleHopManagerM.Control;
  SendMsg =  SingleHopManagerM.SendMsg;
  ReceiveMsg =  SingleHopManagerM.ReceiveMsg;

  SingleHopManagerM.RadioControl                    -> Comm;
  SingleHopManagerM.RadioCommControl        -> Comm;
  SingleHopManagerM.RadioSend                               -> Comm;
  SingleHopManagerM.RadioReceive                    -> Comm.ReceiveMsg;
#if BS
    SingleHopManagerM.UARTReceive                           -> Comm.UARTReceiveMsg;
#endif
    SingleHopManagerM.getAODVseqNum         =       getAODVseqNum;
  SingleHopManagerM.Leds                                                -> LedsC;

#ifndef PLATFORM_PC
    SingleHopManagerM.MacControl                            -> CC2420RadioC;
#endif

}
```

H8. SingleHopManagerM.nc

```
module SingleHopManagerM
```

```
{
    provides {
    interface StdControl as Control;
    interface SendMsg[uint8_t id];
    interface ReceiveMsg      as PromiscuousReceiveMsg[uint8_t id];
    interface ReceiveMsg[uint8_t id];
       interface Payload2;
    interface SingleHopMsg;
    interface SequenceNumber;

   }
  uses {
       command uint16_t getAODVseqNum();
    interface StdControl      as RadioControl;
    interface CommControl as RadioCommControl;
    interface SendMsg         as RadioSend[uint8_t id];
    interface ReceiveMsg      as RadioReceive[uint8_t id];
       interface Payload2            as SubPayload2;
#if BS
       interface ReceiveMsg  as UARTReceive[uint8_t id];
#endif
       interface Leds;
#ifndef PLATFORM_PC
       interface MacControl;
#endif
  }
}
implementation {
    uint8_t seq;
    uint8_t seqP;
    uint8_t index;

    command result_t Control.init() {
       dbg(DBG_BOOT, "SingleHopManager s'inicialitza\n");
       seq = 0;
       seqP = 0;
       call Leds.init();
       call RadioControl.init();
       call RadioCommControl.setCRCCheck(TRUE);
       call RadioCommControl.setPromiscuous(TRUE);
#ifndef PLATFORM_PC
       call MacControl.enableAck();
#endif
       index=0;
       dbg(DBG_ROUTE,  "SHOP,  Acknowledgments,  control  CRC    i  mode  Promiscuous
activats.\n");
       return SUCCESS;
    }

    command result_t Control.start() {
       return call RadioControl.start();
    }

    command result_t Control.stop() {
       dbg(DBG_USR2, "STOP del modul de RADIO\n");
       return call RadioControl.stop();
    }

    command void * Payload2.linkPayload(TOS_MsgPtr msg, uint8_t *len) {
       SHop_MsgPtr sHopMsg;
       uint8_t length;
       sHopMsg = call SubPayload2.linkPayload(msg, &length);
       *len = length - SHOP_HEADER_LEN;
       return sHopMsg->data;
    }

    default command void * SubPayload2.linkPayload(TOS_MsgPtr msg, uint8_t *len) {
       *len = DATA_LENGTH;
       return msg->data;
    }


    //  Permet l'enviament de missatges en mode salt directe. (cal indicar adreca
    //destinació, longitud
    //  de les dades i el punter al missatge. S'envia amb indicador de port 'id'.

    command result_t SendMsg.send[uint8_t id](uint16_t addr, uint8_t length, TOS_MsgPtr
```

```
    msg) {
        SHop_MsgPtr sHopMsg = (SHop_MsgPtr) msg->data;
        sHopMsg->seq = seq;
        sHopMsg->src = (wsnAddr) TOS_LOCAL_ADDRESS;

        dbg(DBG_ROUTE, "SHOP_M: incrementant número de sequenciaShop a %d\n", seq);

                        if (addr == 0xff)
                addr= TOS_BCAST_ADDR;

        dbg(DBG_ROUTE,"SHOP_M: comanda SendMsg.send(...). enviant missatge #%d [tipus=%d]
        al node %d.\n", seq, id, addr);
        return call RadioSend.send[id](addr, length + SHOP_HEADER_LEN, msg);
    }

        //      Event de missatge enviat per radio.

    event result_t RadioSend.sendDone[uint8_t id](TOS_MsgPtr msg, result_t success) {
        result_t ret = success;

        if((msg->addr == TOS_BCAST_ADDR ||msg->addr == TOS_LOCAL_ADDRESS) && id==0x88){
                call Leds.redOff();
                call Leds.greenOff();
                call Leds.yellowOff();
                return SUCCESS;
        }

        seq++;
        if ((msg->addr == TOS_BCAST_ADDR)&&(success==SUCCESS)){
                msg->ack=1;
                dbg(DBG_ROUTE, "SHop, BCAST_MSG: el camp ack val: %d\n",msg->ack);
                return signal SendMsg.sendDone[id](msg, SUCCESS);
        }

        //Control d'ACK       i CRC [1: ok, 0: !ok]
        dbg(DBG_ROUTE, "SHop, UNICAST_MSG: el camp ack val: %d\n",msg->ack);
        if ((msg->ack == 1) && (success == SUCCESS)){
                dbg(DBG_ROUTE, "SHOP_M: radiosend.senddone dona enviament correcte.\n");
                ret = signal SendMsg.sendDone[id](msg, success);
        }else{
                dbg(DBG_ERROR, "SHOP_M: RadioSend detecta: ERROR en el camp ACK o
                general.");
                ret = signal SendMsg.sendDone[id](msg, FAIL);
        }
        return SUCCESS;
  }

    //  Prescindible. Unicament per assegurar funcionament si el metode no esta
    //  implementat.

    default event result_t SendMsg.sendDone[uint8_t id](TOS_MsgPtr msg, result_t
    success) {
        return SUCCESS;
    }

    //  Recepció de missatges de la UART.
    //  Atenció: Missatge UART ve unicament provist de dades a partir de AODV_Msg->data,
        es passa
    //  directament el missatge a l'aplicació AODV_PacketForwarder, que l'envia com si
        fos salt.
#if BS
    event TOS_MsgPtr UARTReceive.receive[uint8_t id](TOS_MsgPtr msg) {
        SHop_MsgPtr sHopMsg = (SHop_MsgPtr) msg->data;
        dbg(DBG_ROUTE,  "SHOP_M:  rebut  packet  procedent  de  la  UART.  Fem  signal  a
promiscuousreceive\n");

        if((msg->addr == TOS_BCAST_ADDR ||msg->addr == TOS_LOCAL_ADDRESS) && id==0x88){
                call Leds.redOn();
                call Leds.yellowOn();
                call Leds.greenOn();

                call RadioControl.stop();
                call RadioControl.init();
                call RadioCommControl.setCRCCheck(FALSE);
                call RadioCommControl.setPromiscuous(TRUE);
#ifndef PLATFORM_PC
                call MacControl.enableAck();
```

```
#endif
                call RadioControl.start();

                seqP = sHopMsg->seq;
                sHopMsg->src = (wsnAddr) TOS_LOCAL_ADDRESS;

                call RadioSend.send[0x88](TOS_BCAST_ADDR, 0x0A, msg);
                return msg;
        }
        return signal ReceiveMsg.receive[AM_ID_FROM_UART](msg);
    }
#endif

    //  Recepció de missatges de la Radio. Controlem el camp CRC.

    event TOS_MsgPtr RadioReceive.receive[uint8_t id](TOS_MsgPtr msg) {
        TOS_MsgPtr ret;
        SHop_MsgPtr sHopMsg = (SHop_MsgPtr) msg->data;


        if((msg->addr == TOS_BCAST_ADDR ||msg->addr == TOS_LOCAL_ADDRESS) && id==0x88) {

                if (seqP == sHopMsg->seq){
                        call Leds.greenToggle();
                        return msg;
                }

                call Leds.redOn();
                call Leds.yellowOn();
                call Leds.greenOn();

                call RadioControl.stop();
                call RadioControl.init();
                call RadioCommControl.setCRCCheck(FALSE);
                call RadioCommControl.setPromiscuous(TRUE);
#ifndef PLATFORM_PC
                call MacControl.enableAck();
#endif
                call RadioControl.start();

                sHopMsg->src = (wsnAddr) TOS_LOCAL_ADDRESS;
                seqP = sHopMsg->seq;

                call RadioSend.send[0x88](TOS_BCAST_ADDR, 0x0A, msg);
                return msg;
        }

        if (msg->crc == 0) {
                // En aquest cas assumim que el packet es erroni de MicaHighSpeedRadio.
                dbg(DBG_ERROR, "SHOP_M: rebut packet de la Radio amb CRC erroni\n");
                ret=msg;
        } else {
                // El CRC es correcte, interpretem el packet.
                dbg(DBG_ROUTE,"SHOP_M: rebut msg correctament de tipus %d [seq#%d del
                node %d].\n", id, sHopMsg->seq, sHopMsg->src);

                dbg(DBG_ROUTE,"SHOP_M: updating sequencenumber\n");
                signal SequenceNumber.updateSeqNum(sHopMsg->src, sHopMsg->seq, call
                getAODVseqNum());

                // Per passar missatge a capes superiors, es prova primer amb l'interface
                //de sortida cap al AODV_PF, si aquest retorna NULL vol dir que el packet
                //s'utilitza a la capa AODV_PF i no cal ser enviat a  la capa DE
                //AODV_Core. Si no el requereix es retorna el mateix packet. (veure
                //funció default de event     PromiscuousReceiveMsg)

                dbg(DBG_ROUTE,"SHOP_M: signaling PromiscuousReceiveMsg\n");
                ret = signal PromiscuousReceiveMsg.receive[id](msg);
                if (ret != NULL) {
                        if ((ret->addr == TOS_LOCAL_ADDRESS) || (ret->addr ==
                        TOS_BCAST_ADDR)) {
                                dbg(DBG_ROUTE,"SHOP_M: Missatge no utilitzat per
                                AODV_PacketForwarder, el passem a AODV_Core\n");
                                ret = signal ReceiveMsg.receive[ret->type](ret);
                        }else{
                                dbg(DBG_ROUTE,"SHOP_M: Missatge no utilitzat
                                descartat.\n");
```

```
                }
            }else{
                dbg(DBG_ROUTE, "SHOP_M: Missatge utilitzat per
                AODV_PacketForwarder.\n");
            }
        }

        return msg;
    }

    // Event d'enviament al AODV_PacketForwarder
  default event TOS_MsgPtr PromiscuousReceiveMsg.receive[uint8_t id](TOS_MsgPtr msg) {
    return msg;
  }

    // Event d'enviament al AODV_Core
  default event TOS_MsgPtr ReceiveMsg.receive[uint8_t id](TOS_MsgPtr msg) {
    return msg;
  }

    // Event d'actualització del num de sequencia
  default event void SequenceNumber.updateSeqNum(wsnAddr addr, uint8_t seqNum, uint16_t
    mySeq) {
  }

      command wsnAddr SingleHopMsg.getSrcAddress(TOS_MsgPtr msg) {
    SHop_MsgPtr sHopMsg = (SHop_MsgPtr) msg->data;
    return sHopMsg->src;
  }

    command wsnAddr SingleHopMsg.getDestAddress(TOS_MsgPtr msg) {
        return (wsnAddr) msg->addr;
    }

  command wsnAddr SingleHopMsg.getSeqNum(TOS_MsgPtr msg) {
    SHop_MsgPtr sHopMsg = (SHop_MsgPtr) msg->data;
    return sHopMsg->seq;
  }


}
```

H.9. SimpleQueueM.nc

```
module SimpleQueueM {
   provides {
      interface StdControl    as Control;
      interface SendMsg               as Send;
   }
   uses {
      interface StdControl    as SubControl;
      interface SendMsg               as SubSend;
               interface Random;
               interface Leds;
   }
}

implementation {
  typedef struct QueueEnt {
    TOS_Msg msg;
       uint16_t dest;                       //tamany destinació depen de capes inferiors
       uint8_t length;
       uint8_t sendAttempts;
    struct QueueEnt *next;
  } QueueEnt;

  QueueEnt entryBuffers[SEND_QUEUE_SIZE];
  QueueEnt * emptyList;
  QueueEnt * queueHead;

  bool sendPending;

  command result_t Control.init() {
    int i;
    emptyList = NULL;
    for (i=0; i<SEND_QUEUE_SIZE; i++) {
      entryBuffers[i].next = emptyList;
               entryBuffers[i].sendAttempts = 0;
```

```
            entryBuffers[i].dest = 0;
            entryBuffers[i].length=0;
    emptyList = &(entryBuffers[i]);
  }

     call Leds.init();
     call Random.init();

  sendPending = FALSE;
  queueHead = NULL;
     return SUCCESS;
}

command result_t Control.start() {
     //call Leds.redToggle();
  return SUCCESS;
}

command result_t Control.stop() {
  return SUCCESS;
  }

// return a slot from the empty list
QueueEnt * getNewEntry() {
  QueueEnt * ret = emptyList;
  if (ret != NULL) {
    emptyList = ret->next;
     }
  return ret;
}

 // add a new message to the end of the list
void enqueueMessage(QueueEnt *newEnt) {
  newEnt->next = NULL;
  if (queueHead == NULL) {
    queueHead = newEnt;
  } else {
    QueueEnt * ent;
    for (ent = queueHead; ent->next != NULL; ent = ent->next);
    ent->next = newEnt;
  }
}

void dequeueMessage() {
  QueueEnt *ent = queueHead;
  queueHead = queueHead->next;

  ent->next = emptyList;  // enqueue onto emtpy list
  emptyList = ent;

  ent->sendAttempts = 0;
     ent->dest = 0;
     ent->length=0;
}

task void attemptToSend() {

     if (queueHead == NULL || sendPending == TRUE) {
            dbg(DBG_USR3, "SQM, attemptToSend falla perque %s.\n",(sendPending) ?
            "sendPending=TRUE" : "No hi han msg a enviar");
    return;
  }
  if (queueHead->sendAttempts > SEND_QUEUE_MAX_RETRIES) {
            // caldria comprobar que quan fem el dequeue del msg, no afecta a les
            //dades guardades
            dbg(DBG_USR3, "SQM, se fa signal d'error perque s'han esgotat els intents
            a destinació %d.\n", queueHead->dest);
            signal Send.sendDone(&queueHead->msg, FAIL);
    dequeueMessage();
            return;
  }
     dbg(DBG_USR3, "SQM, Realitzant intent %d d'enviar msg al node %d.\n",queueHead-
     >sendAttempts, queueHead->dest);
     sendPending = TRUE;

     // Retrassem algunes peticions
            if((call Random.rand() & 0xff) % (AODV_MAX_RAND/2)){
```

```
                    sendPending = FALSE;
                    dbg(DBG_ROUTE, "SQM fa un delay del missatge\n");
                    post attemptToSend();
                    return;
                }


    if (call SubSend.send(queueHead->dest,queueHead->length,&queueHead->msg) == FAIL) {
      sendPending = FALSE;
      queueHead->sendAttempts++;
      dbg(DBG_USR3, "SQM, SubSend(FAIL), queden %d intents.\n", queueHead-
       >sendAttempts);
              post attemptToSend();
      }
  }

  command result_t Send.send(uint16_t addr, uint8_t length, TOS_MsgPtr msg) {
    QueueEnt * newEntry = getNewEntry();

       //call Leds.yellowToggle();

    if (newEntry == NULL) {  // queue is full
              dbg(DBG_USR3, "SQM, Cua plena, missatge descartat.\n");
      return FAIL;
    }

       memcpy(&newEntry->msg, msg, sizeof(TOS_Msg));
       newEntry->sendAttempts = 0;
       newEntry->dest =addr;
       newEntry->length = length;

    // enqueue new entry to the last entry of the list
    enqueueMessage(newEntry);
       dbg(DBG_USR3, "SQM, posat en cua missatge amb destinació %d.\n", addr);
       // Intenta enviar msg, si falla més de MAX_RETRIES s'envia a la cua de
       BuffereQueue
       post attemptToSend();
       return SUCCESS;
  }

  event result_t SubSend.sendDone(TOS_MsgPtr msg, result_t success) {
       if (sendPending == TRUE){
      sendPending = FALSE;
      dbg(DBG_USR3, "SQM, Enviat msg amb destinacio %d.\n", queueHead->dest);
      if ((success == SUCCESS) && (msg->ack)) {
                    signal Send.sendDone(msg, success);
        dequeueMessage();
        dbg(DBG_USR3, "SQM, Missatge extret de la cua.\n");
      } else {
                    queueHead->sendAttempts++;
        dbg(DBG_USR3, "SQM, SubSend failed, s'han realitzat %d intents.\n", queueHead-
        >sendAttempts);
      }
    }else{
              dbg(DBG_USR3, "SQM, No es el misssatge esperat.\n");
              signal Send.sendDone(msg, success);
       }

       if (queueHead != NULL) {
              dbg(DBG_USR3,"SQM, posting attemptToSend.\n");
              post attemptToSend(); //trigger to send pending elements
    }
    return SUCCESS;
  }

  default event result_t Send.sendDone(TOS_MsgPtr msg, result_t success) {
    return SUCCESS;
  }
}
```

H.10. BufferQueueM.nc

```
module BufferQueueM {
   provides {
      interface StdControl as QueueControl;
      interface MyQueue;
   }
```

```
}

implementation {

  TOS_Msg msg_out;
  TOS_MsgPtr msg_outPtr;

  typedef struct QueueEnt {
    TOS_Msg msg;// guarda el msg enter
    uint8_t tries;
    struct QueueEnt *next;// punter al seguent element de la cua
  } QueueEnt;

  QueueEnt entryBuffers[QUEUE_SIZE];
  QueueEnt * emptyList;// punter a la primera posició lliure de la cua
  QueueEnt * queueHead;// punter al missatge inicial de la cua

  command result_t QueueControl.init() {
    uint8_t i;
    msg_outPtr = &msg_out;

    emptyList = NULL;
    for (i=0; i<QUEUE_SIZE; i++) {
      entryBuffers[i].next                                    = emptyList;           // next
= anterior msg (i--)
                                                               entryBuffers[i].tries = 0;
      emptyList = &(entryBuffers[i]);
    // emptyList apunta a ultima posició de cua(la primera.
    }
    queueHead = NULL;
                                                                                      //
inicialitzem queueHEAD a NULL
                                                            return SUCCESS;
  }

  command result_t QueueControl.start() {
    return SUCCESS;
  }

  command result_t QueueControl.stop() {
    return SUCCESS;
    }

  // return a slot from the empty list
  QueueEnt * getNewEntry() {
                                                            // retorna la posicio indicada
per el camp next de l'ultim missatge (emptylist)
    QueueEnt * ret = emptyList;
    if (ret != NULL) {
      emptyList = ret->next;
                                                            }
    return ret;

  }

  uint8_t queueLen() {
    QueueEnt *ent = queueHead;
    uint8_t len=0;

    while (ent != NULL) {
      ent = ent->next;
      len++;
    }
    return len;
  }

    // add a new message to the end of the list
  void enqueueMessage(QueueEnt *newEnt) {
    QueueEnt * ent;
    newEnt->next = NULL;
    if (queueHead == NULL) {
    // no hi ha msg, per tant assignem el nou msg a la capcalera de la cua.
                                                             dbg(DBG_USR3, "BQM, encuat
com a primer missatge a la cua.\n");
      queueHead = newEnt;
    } else {
    // si hi han msg en cua, cal actualitzar el valor de next, en el penultim missatge a
```

```
    la nova entrada.
      dbg(DBG_USR3, "BQM, encuat com a missatge num %d.\n", queueLen());
      for (ent = queueHead; ent->next != NULL; ent = ent->next){
                                                              ent->next = newEnt;
                                                          }
    }
  }


    command result_t MyQueue.enQueue(TOS_MsgPtr msg, uint8_t number_tries){
    QueueEnt * newEntry;

    dbg(DBG_USR3, "BQM, intents %d.\n", number_tries);

    newEntry = getNewEntry();

    if (newEntry == NULL) {
    // no hi han posicions lliures o ve hi ha un error.
    dbg(DBG_USR3, "BQM, es descarta packet perque la cua esta plena.\n");
      return FAIL;
    }

    // copiem les dades i actualitzem cua
    memcpy(&newEntry->msg, msg, sizeof(TOS_Msg));
    newEntry->tries = number_tries;
    enqueueMessage(newEntry);

    dbg(DBG_USR3, "BQM, colocat satisfactoriament el packet a la cua en posició %d, cua
    len = %d.\n",  newEntry, queueLen());
    return SUCCESS;
    }

    command TOS_MsgPtr MyQueue.getNext(uint8_t *numtries){

    QueueEnt * ent;

    if (queueHead == NULL) {
    //indica que no hi ha msg en cua
    dbg(DBG_USR3, "BQM, no hi han més missatges en cua (len = %d).\n", queueLen());
      return NULL;
    }
    //guardem posicio de memoria de queueHead a posicio de memoria msg_outPtr
    memcpy(msg_outPtr, &queueHead->msg, sizeof(TOS_Msg));
    dbg(DBG_USR3, "BQM, intents de sortida %d\n", queueHead->tries);
    *numtries = queueHead->tries;

    // assignem queueHead al seguent msg
    ent = queueHead;
    queueHead = queueHead->next;

    // preparem la posició esborrada, assignant camp next, i colocant posicio al camp
    emtylist.
    ent->next = emptyList;
    emptyList = ent;
    dbg(DBG_USR3, "BQM, agafem missatge de la cua.\n");

    return msg_outPtr;
    }
}
```

Anexo I: Código escenarios nesC

Con este código hemos desarrollado todas los escenarios antes descritos en la memoria pero haciendo variaciones a pie de código para adaptarlos en particular este pertenece a un escenario de envío bidireccional (envío de paquetes de ida y vuelta) con cadencia de envío de 10 paquetes por segundo.

Consideramos oportuno hacer una breve descripción de los componentes que forman partes de nuestros programas:

LedsC: Componente que se ocupa de accionar los leds de nuestros dispositivos sensores.
SysTimeC: Componente una vez inicializado, controla un reloj de tiempo local, el tiempo es generado en ticks, cada tick son 1,09us, tiene una resolucion máxima de 32 bits, limite que cuando supera vuelve a empezar de 0.
VoltageC: Este componente nos da acceso a un conversor analogico/digital que nos devuelve en mV el estado de las baterias.
TimerC:Temporizador propio de tinyOS.
AODC: Este componente es tratado en el proyecto y en diversos anexos.

I.1 Primerizo.nc

```
includes WSNMsg;
includes AODV;


configuration Primerizo {
}

implementation {
   components /*WSN_Main,*/Main, LedsC, SysTimeC,
              PrimerizoM, AODV, TimerC,VoltageC;

    Main.StdControl -> PrimerizoM;
    PrimerizoM.Leds -> LedsC;
    PrimerizoM.Timer -> TimerC.Timer[unique("Timer")];
    PrimerizoM.Receive -> AODV.Receive[APP_ID_WSN];
    PrimerizoM.AODV -> AODV.Control;
    PrimerizoM.SysTime -> SysTimeC;
    PrimerizoM.SendMHopMsg -> AODV.SendMHopMsg[APP_ID_WSN];
    PrimerizoM.MultiHopMsg -> AODV;
    PrimerizoM.SingleHopMsg->AODV;
    PrimerizoM.BattControl -> VoltageC;
    PrimerizoM.ADCBATT -> VoltageC;

}
```

I.2 PrimerizoM.nc

```
includes WSNMsg;

module PrimerizoM {
   provides {
      interface StdControl as Control;
   }
   uses {
          interface StdControl as AODV;
          interface Timer;
          interface Leds;
          interface SendMHopMsg;
```

```
          interface Receive;
          interface SysTime ;
          interface MultiHopMsg;
          interface SingleHopMsg;
          interface ADC as ADCBATT;
          interface StdControl as BattControl;
   }
}

implementation {

  typedef struct {
    uint32_t tiempo;
    uint16_t bateria;
    uint32_t tabsolut;
    uint32_t seq;
    uint8_t relleno[90]; // La tasa maxima de relleno es 90.
    } datos;

    uint32_t tiempo1;
    uint32_t tiempo2;
    uint32_t tiempo;
    uint16_t bateria;
    TOS_Msg msgbuf;
    TOS_MsgPtr msgptr;
    uint32_t myseq;
    int i;

   command result_t Control.init() {
      call Leds.init();
      call AODV.init();
      call BattControl.init();

      msgptr = &msgbuf;
      i=0;
      myseq  =0;
      return SUCCESS;
   }

   command result_t Control.start() {
      call AODV.start();
      call Timer.start(TIMER_REPEAT,100);
      call Leds.greenOff();
      call Leds.yellowOff();
      call Leds.redOff();

      return SUCCESS;
   }

   command result_t Control.stop() {
      call Timer.stop();
      call AODV.stop();
      call BattControl.stop();

      return SUCCESS;
    }


     task void sender()
    {
      datos *ptrmsg;
    //uint16_t *ptrmsg;
      uint16_t len;
     atomic {
               myseq++;
               ptrmsg =  call SendMHopMsg.getBuffer(msgptr, &len);
               ptrmsg->bateria = bateria;
               ptrmsg->seq = myseq;
               ptrmsg->tiempo = myseq;//tiempo;
               atomic tiempo1 = call SysTime.getTime32();
               ptrmsg->tabsolut = tiempo1;
               call SendMHopMsg.sendTTL(0x02,sizeof(datos), msgptr,0); //ojo!!!
                                                              }
    }

    event result_t Timer.fired()
    {
```

```
         call Leds.redToggle();
         post sender();
         return SUCCESS;
    }

    event result_t SendMHopMsg.sendDone(TOS_MsgPtr sent, result_t success)
    {
       if (success==FAIL) call Leds.redToggle();            //envío incorrecto
       if (success==SUCCESS)  call Leds.greenToggle(); //envío correcto

      return SUCCESS;
    }


    event TOS_MsgPtr Receive.receive(TOS_MsgPtr m,void *eso,uint16_t longitud)
    {
        atomic{
               tiempo2 = call SysTime.getTime32();

               tiempo = tiempo2 - tiempo1;
              call BattControl.start();
              call ADCBATT.getData();
              call Leds.yellowToggle();
              return m;
             }

async event result_t ADCBATT.dataReady(uint16_t data) {
    atomic{
            bateria = data;
            call BattControl.stop();
            call BattControl.init();
                                                              }
        return SUCCESS;
    }

}
```

I.3 Segundo.nc

```
includes WSNMsg;
includes AODV;


configuration Segundo {

}

implementation
{
  components Main, SegundoM, TimerC, LedsC,SysTimeC, AODV ;
  //components TimerC as tempo;

  Main.StdControl -> SegundoM;
  Main.StdControl -> AODV.TimerControl;
  Main.StdControl -> TimerC;
  SegundoM.Leds -> LedsC;
  SegundoM.Receive -> AODV.Receive[APP_ID_WSN];
  SegundoM.AODV -> AODV.Control;
  SegundoM.SendMHopMsg -> AODV.SendMHopMsg[APP_ID_WSN];
  SegundoM.SysTime -> SysTimeC;
  SegundoM.MultiHopMsg -> AODV;
  SegundoM.SingleHopMsg-> AODV;
  SegundoM.Timer ->  TimerC.Timer[unique("Timer")];
  SegundoM.ConfigTimer -> TimerC.Timer[unique("Timer")];
}
```

I.4 SegundoM.nc

```
module SegundoM {
  provides {
    interface StdControl;
  }
  uses {
    interface Timer;
```

```
    interface Receive;
    interface SendMHopMsg;
    //interface Send;
    interface StdControl as AODV;
    interface Leds;
    interface MultiHopMsg;
    interface SingleHopMsg;
    interface Timer as ConfigTimer;
    interface SysTime ;

  }
}

implementation {
     typedef struct {
    uint32_t tiempo;
    uint16_t bateria;
    uint32_t tabsolut;
    uint32_t seq;
    //uint8_t relleno[10]; //El tamaño máximo del relleno es de 90
    } datos;
    uint32_t tiempo1;
    uint32_t tiempo2;
    uint32_t tiempo;
    uint16_t bateria;
    uint32_t myseq;
    TOS_Msg msgbuf;
    TOS_MsgPtr msgptr;
    int i;



void reiniciar(){
    call Leds.set(0x07);
    call AODV.stop();
    call AODV.init();
    }
    command result_t StdControl.init()
    {
       call Leds.init();
       call AODV.init();
       msgptr = &msgbuf;
       i= 0;
       myseq  =0;
       bateria=0;
       tiempo=0;
       return SUCCESS;
    }

    command result_t StdControl.start()
    {
       call AODV.start();
       call Leds.greenOff();
       call Leds.yellowOff();
       call Leds.redOff();
       return SUCCESS;
    }

    command result_t StdControl.stop()
    {
       call AODV.stop();
       return SUCCESS;
    }
    task void sender()
    {
       datos *ptrmsg;
       uint16_t len;
        atomic {

                       ptrmsg =  call SendMHopMsg.getBuffer(msgptr, &len);
                       ptrmsg->bateria = bateria;
                       myseq++;
                       ptrmsg->tiempo = myseq;
                       ptrmsg->tabsolut = tiempo1;
                       ptrmsg->seq = myseq;
                       call SendMHopMsg.sendTTL(0x0001,sizeof(datos), msgptr,0); //ojo!!!
        }
```

```
        }
event result_t Timer.fired()
    {
        //call Leds.set(0x00);
        call Timer.stop();
        return SUCCESS;
    }


    event result_t SendMHopMsg.sendDone(TOS_MsgPtr sent, result_t success)
    {
        call Leds.greenToggle();

        //if (success == SUCCESS)  call Leds.set(0x07); //envío correcto todos los leds
        //else call Leds.set(0x04);                 //envío incorrecto amarillo
        return SUCCESS;
    }

  event TOS_MsgPtr Receive.receive(TOS_MsgPtr msg, void  *payload, uint16_t payloadLen)
      {

              atomic tiempo1 = call SysTime.getTime32();
              call Leds.yellowToggle();
              post sender();
              return msg;

      }


}
```

Anexo J: Codigo C# de interficie del PC.

Para poder recuperar los datos de los dispositivos, aparte del serialforwarder (anexo G) hemos necesitado crear una aplicación que nos recuperase los datos y los tratase para así obtener los distintos parámetros que nos devuelven los dispositivos sensores y así poder evaluar nuestra red de sensores.

Con el entorno Microsoft visual studio y el lenguaje de programación C# hemos creado ese programa.

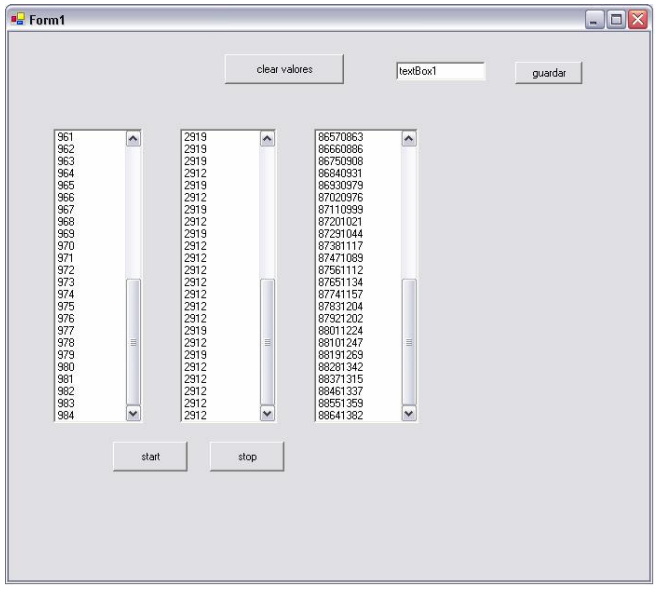


Fig J.1. Aspecto de la interficie de PC.

Su modo de funcionamiento es bastante sencillo hemos arrancado el serial forwarder, arrancamos este programa que abre un socket TCP hacia el puerto establecido por serialforwarder una vez lo arrancamos y picamos al botón start el nos ira filtrando los paquetes que tengan información e ira clasificando la información en las 3 ventanas que vemos en la figura, en el caso concreto de la figura el primer cuadro indica el número de secuencia, en el segundo el valor en mV del voltaje que dan las pilas del dispositivo sensor, y la tercera es un valor de tiempo absoluto expresado en ticks.

Aparte este programa nos permite guardar en un fichero de texto todos los resultados obtenidos, con lo cual es fácil ponerlos en una hoja de cálculo para realizar las estadísticas pertinentes.

J.1 Codigo utilizado

```
using System;
using System.Drawing;
//using System.Collections.Generic;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Threading;
using System.Net;
using System.Net.Sockets;
```

```
using System.IO;


namespace WindowsApplication3
{
	/// <summary>
	/// Summary description for Form1.
	/// </summary>
	public class Form1 : System.Windows.Forms.Form
	{

		Thread t;
		uint C1, C2, C3;

		Socket C;
		static IPAddress local = Dns.Resolve(Dns.GetHostName()).AddressList[0];
		static byte[] buffer = new byte[40];
		IPEndPoint localEndPoint = new IPEndPoint(local, 9001);
		static byte[] init = new byte[2];
		static uint I,J,K,M,i;
		//System.DateTime t1;
		private System.Windows.Forms.Button button1;
		private System.Windows.Forms.ListBox valores1;
		private System.Windows.Forms.TextBox textBox1;
		private System.Windows.Forms.Button guardar;
		private System.Windows.Forms.Button button2;
		private System.Windows.Forms.Button button3;
		private System.Windows.Forms.ListBox valores2;
		private System.Windows.Forms.ListBox valores3;
		/// <summary>
		/// Required designer variable.
		/// </summary>
		private System.ComponentModel.Container components = null;

		public Form1()
		{
			//
			// Required for Windows Form Designer support
			//
			InitializeComponent();

			//
			// TODO: Add any constructor code after InitializeComponent call
			//
		}

		/// <summary>
		/// Clean up any resources being used.
		/// </summary>
		protected override void Dispose( bool disposing )
		{
			if( disposing )
			{
				if (components != null)
				{
					components.Dispose();
				}
			}
			base.Dispose( disposing );
		}

		#region Windows Form Designer generated code
		/// <summary>
		/// Required method for Designer support - do not modify
		/// the contents of this method with the code editor.
		/// </summary>
		private void InitializeComponent()
		{
			this.valores1 = new System.Windows.Forms.ListBox();
			this.button1 = new System.Windows.Forms.Button();
			this.textBox1 = new System.Windows.Forms.TextBox();
			this.guardar = new System.Windows.Forms.Button();
			this.button2 = new System.Windows.Forms.Button();
			this.button3 = new System.Windows.Forms.Button();
			this.valores2 = new System.Windows.Forms.ListBox();
			this.valores3 = new System.Windows.Forms.ListBox();
			this.SuspendLayout();
```

```
//
// valores1
//
this.valores1.Location = new System.Drawing.Point(48, 104);
this.valores1.Name = "valores1";
this.valores1.Size = new System.Drawing.Size(96, 316);
this.valores1.TabIndex = 0;
//
// button1
//
this.button1.Location = new System.Drawing.Point(112, 440);
this.button1.Name = "button1";
this.button1.Size = new System.Drawing.Size(80, 32);
this.button1.TabIndex = 1;
this.button1.Text = "start";
this.button1.Click += new System.EventHandler(this.button1_Click);
//
// textBox1
//
this.textBox1.Location = new System.Drawing.Point(416, 32);
this.textBox1.Name = "textBox1";
this.textBox1.Size = new System.Drawing.Size(96, 20);
this.textBox1.TabIndex = 2;
this.textBox1.Text = "textBox1";
//
// guardar
//
this.guardar.Location = new System.Drawing.Point(544, 32);
this.guardar.Name = "guardar";
this.guardar.Size = new System.Drawing.Size(72, 24);
this.guardar.TabIndex = 3;
this.guardar.Text = "guardar";
this.guardar.Click += new System.EventHandler(this.guardar_Click);
//
// button2
//
this.button2.Location = new System.Drawing.Point(216, 440);
this.button2.Name = "button2";
this.button2.Size = new System.Drawing.Size(80, 32);
this.button2.TabIndex = 4;
this.button2.Text = "stop";
this.button2.Click += new System.EventHandler(this.button2_Click);
//
// button3
//
this.button3.Location = new System.Drawing.Point(232, 24);
this.button3.Name = "button3";
this.button3.Size = new System.Drawing.Size(128, 32);
this.button3.TabIndex = 5;
this.button3.Text = "clear valores";
this.button3.Click += new System.EventHandler(this.button3_Click);
//
// valores2
//
this.valores2.Location = new System.Drawing.Point(184, 104);
this.valores2.Name = "valores2";
this.valores2.Size = new System.Drawing.Size(104, 316);
this.valores2.TabIndex = 6;
//
// valores3
//
this.valores3.Location = new System.Drawing.Point(328, 104);
this.valores3.Name = "valores3";
this.valores3.Size = new System.Drawing.Size(112, 316);
this.valores3.TabIndex = 7;
//
// Form1
//
this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
this.ClientSize = new System.Drawing.Size(688, 589);
this.Controls.Add(this.valores3);
this.Controls.Add(this.valores2);
this.Controls.Add(this.button3);
this.Controls.Add(this.button2);
this.Controls.Add(this.guardar);
this.Controls.Add(this.textBox1);
this.Controls.Add(this.button1);
```

```
                        this.Controls.Add(this.valores1);
                        this.Name = "Form1";
                        this.Text = "Form1";
                        this.Load += new System.EventHandler(this.Form1_Load);
                        this.ResumeLayout(false);

                }
                #endregion

                /// <summary>
                /// The main entry point for the application.
                /// </summary>
                [STAThread]
                static void Main()
                {
                        Application.Run(new Form1());
                }

                private void Form1_Load(object sender, System.EventArgs e)
                {

                }
                public void captura()
                {
                        C = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
                        //valores1.Items.Clear();

                        /* valores2.Items.Clear();
                        valores3.Items.Clear(); */
                        C1 = C2 = C3 = 0;
                        //uint media = 0;
                        //  uint minimo = 100000;
                        // max = 7000;
                        try
                        {
                                C.Connect(localEndPoint);

                                C.Receive(init);
                                foreach (byte a in init) //Salta por pantalla el codigo de
inicio
                                {
                                        Console.Write(a);
                                }
                                Console.Write("\n");
                                C.Send(init);                //Le devolvemos el mismo codigo
de inicio
                                while ((C1 < 1000000))     //10 medidas?
                                {

                                        for (i=0;i<40;i++) buffer[i]=0;
                                        C.Receive(buffer);

                                        foreach (byte a in buffer)
                                        {
                                                Console.Write(a + "-");
                                        }
                                        Console.Write("\n");

                                        switch (buffer[2])
                                        {
                                                case 0x09:
                                                        Console.Write("AODV_DATA ");
                                                        break;

                                                case 0x0f:
                                                        Console.Write("AODV_RERR ");
                                                        break;
                                                case 0x0e:
                                                        Console.Write("AODV_RRPLY ");
                                                        break;
                                                case 0x0d:
                                                        Console.Write("AODV_RREQ ");
                                                        break;
                                        }
```

```
        Console.WriteLine(System.DateTime.Now.ToString()+":"+System.DateTime.Now.Millisec
ond.ToString() );
                                    if (buffer[0] == 1)  //primer byte a 2
                                {    //   ##############################3
        I = System.BitConverter.ToUInt32(buffer, 14); //A  partir del byte  de buffer
        J = System.BitConverter.ToUInt16(buffer, 18);
        K = System.BitConverter.ToUInt32(buffer, 20);  //Lo he de determinar. todavia!!
        M =  System.BitConverter.ToUInt32(buffer, 24);


        Console.WriteLine(" tiempo= " + I);//debug e consola
        Console.WriteLine(" voltaje(mV)=" + J);
        Console.WriteLine(" Tiempo absoluto =" + K);

        Console.WriteLine(" Número de secuencia = " + M);

                                                valores1.Items.Add(I);
                                                valores2.Items.Add(J);
                                                valores3.Items.Add(K);
                                                C1++;
                                        }
                                }
                                C.Close();
                        }
                        catch
                        {
                                Console.WriteLine("error de conexión");
                                t.Abort();
                        }
                }

                private void button1_Click(object sender, System.EventArgs e)
                {
                        t = new Thread(new ThreadStart(captura));
                        t.Start();
                }

                private void guardar_Click(object sender, System.EventArgs e)
                {
                        guarda();
                }
                private void guarda()
                {
                        FileStream Ficherin = new FileStream (textBox1.Text+".txt",
                        FileMode.Create,FileAccess.Write,FileShare.ReadWrite );
                        StreamWriter F = new StreamWriter(Ficherin);
                        //F.WriteLine(valores1.Text);
                        F.Write( "TTL:");
                        foreach (uint i in valores1.Items)
                        {
                                F.Write(i+"\n");
                        }
                        F.Write( "Valores de tensión (mV):");
                        foreach (uint i in valores2.Items)
                        {
                                F.Write(i+"\n");
                        }
                        F.Flush();
                        F.Close();
                }

                private void button2_Click(object sender, System.EventArgs e)
                {
                        t.Interrupt();
                        t.Abort();
                        C.Close();
                }

                private void button3_Click(object sender, System.EventArgs e)
                {
                        valores1.Items.Clear();
                        valores2.Items.Clear();
                        valores3.Items.Clear();
                }
        }
}
```

Anexo K: Librería de constantes referentes a chip de comunicaciones CC2420

Para poder acceder a estos parámetros previamente debemos estudiar el datasheet [3] que nos habla de la parte radio. Posteriormente, buscamos en las librerías radio algún fichero relevante hasta encontrar el fichero llamado cc2420const.h en donde se definen los parámetros por defecto. Dentro nos encontramos con 2 clases de constantes, las que podemos modificar de forma directa y los que podemos modificar de una forma más indirecta, haciendo un pequeño análisis de las más relevantes.

De forma directa:

CC2420_DEF_RFPOWER: Nos permite variar la potencia de emisión de nuestra parte radio en 5 niveles distintos que oscilan entre 0dBm y -25dBm

CC2420_DEF_CHANNEL: Nos permite definir el canal por el que vamos a transmitir. Su valor va entre el canal 11 y el canal 26 es utilizado en combinación a CC2420_DEF_PRESET. Así el canal final se define de la siguiente forma:

Frecuencia(Mhz)=CC2420_DEF_PRESET + (CC2420_DEF_CHANNEL-11) $\times 5$

CC2420_ACK_DELAY: Tiempo en μs que puede tardar en confirmarse un paquete antes de declararlo inválido.

De forma indirecta:

También existen otra clase de variables que únicamente nos indican la posición en memoria y la posición del bit que modifica la configuración, éstos son una serie de registros que se manejan como tales.

Haremos una pequeña descripción de los que consideramos más importantes y su valor por defecto, para el resto será necesario consultar el datasheet que proporciona chipcon:

MDMCTRL1 y MDMCTRL0: son dos registros con diversas variables de configuración que nos permiten modificar los campos que se envían dentro de los paquetes a nivel de enlace, tales como el tamaño de preámbulo, tiempos entre paquetes, así como aspectos de compatibilidad y modulación. También nos permite determinar que nodo se comportará como coordinador PAN.

TXCTRL: es considerado importante ya que nos informa, entre otras cosas, de la potencia de transmisión que vamos a utilizar, de todas formas este parámetro hemos tenido ocasión de configurarlo de forma directa.

FSCTRL entre otros usos también nos permite configurar la frecuencia a la que transmitimos, en la forma directa esto se hace con las constantes CC2420_DEF_PRESET y CC2420_DEF_CHANNEL.

El resto de registros no tienen funciones que sean consideradas relevantes en nuestro proyecto, una extensión de las que hemos comentado y sus valores por defecto están puestos en el anexo K y el datasheet del fabricante [3] .

Como bien decimos en la memoria existe un fichero de configuración y posicionamiento de registros llamado *CC2420Const.h* .

Para consultar la configuración que se utiliza por defecto se ha de consultar el fichero *CC2420ControlM.nc* que aquí es expuesto, como se podrá observar se utilizan mascaras para definir el valor de cada registro.

Existen más ficheros referentes al chipset radio CC2420 pero no entraremos en ellos dada su extensión.

K.1. CC2420const.h

```
#ifndef _CC2420CONST_H
#define _CC2420CONST_H

// times for the CC2420 in microseconds
enum {
  CC2420_TIME_BIT = 4,
  CC2420_TIME_BYTE = CC2420_TIME_BIT << 3,
  CC2420_TIME_SYMBOL = 16
};

#ifndef CC2420_DEF_RFPOWER
#define CC2420_DEF_RFPOWER          0X01
#endif

#ifndef CC2420_DEF_CHANNEL
#define CC2420_DEF_CHANNEL        26  //channel select
#endif

enum {
  CC2420_MIN_CHANNEL =              11,
  CC2420_MAX_CHANNEL =              26
};

#define CC2420_DEF_PRESET           2405  //freq select

#define CC2420_DEF_FCF_LO          0x08
#define CC2420_DEF_FCF_HI          0x01  // without ACK
#define CC2420_DEF_FCF_HI_ACK      0x21  // with ACK
#define CC2420_DEF_FCF_TYPE_BEACON 0x00
#define CC2420_DEF_FCF_TYPE_DATA   0x01
#define CC2420_DEF_FCF_TYPE_ACK    0x02
#define CC2420_DEF_FCF_BIT_ACK     5
#define CC2420_DEF_BACKOFF         500
#define CC2420_SYMBOL_TIME         16 // 2^4
// 20 symbols make up a backoff period
// 10 jiffy's make up a backoff period
// due to timer overhead, 30.5us is close enough to 32us per 2 symbols
#define CC2420_SYMBOL_UNIT         10

// delay when waiting for the ack
#ifndef CC2420_ACK_DELAY
#define CC2420_ACK_DELAY           75
#endif

#ifndef CC2420_XOSC_TIMEOUT
#define CC2420_XOSC_TIMEOUT 200     //times to chk if CC2420 crystal is on
#endif

#define CC2420_SNOP            0x00
#define CC2420_SXOSCON         0x01
#define CC2420_STXCAL          0x02
```

```
#define CC2420_SRXON           0x03
#define CC2420_STXON           0x04
#define CC2420_STXONCCA        0x05
#define CC2420_SRFOFF          0x06
#define CC2420_SXOSCOFF        0x07
#define CC2420_SFLUSHRX        0x08
#define CC2420_SFLUSHTX        0x09
#define CC2420_SACK            0x0A
#define CC2420_SACKPEND        0x0B
#define CC2420_SRXDEC          0x0C
#define CC2420_STXENC          0x0D
#define CC2420_SAES            0x0E
#define CC2420_MAIN            0x10
#define CC2420_MDMCTRL0        0x11
#define CC2420_MDMCTRL1        0x12
#define CC2420_RSSI            0x13
#define CC2420_SYNCWORD        0x14
#define CC2420_TXCTRL          0x15
#define CC2420_RXCTRL0         0x16
#define CC2420_RXCTRL1         0x17
#define CC2420_FSCTRL          0x18
#define CC2420_SECCTRL0        0x19
#define CC2420_SECCTRL1        0x1A
#define CC2420_BATTMON         0x1B
#define CC2420_IOCFG0          0x1C
#define CC2420_IOCFG1          0x1D
#define CC2420_MANFIDL         0x1E
#define CC2420_MANFIDH         0x1F
#define CC2420_FSMTC           0x20
#define CC2420_MANAND          0x21
#define CC2420_MANOR           0x22
#define CC2420_AGCCTRL         0x23
#define CC2420_AGCTST0         0x24
#define CC2420_AGCTST1         0x25
#define CC2420_AGCTST2         0x26
#define CC2420_FSTST0          0x27
#define CC2420_FSTST1          0x28
#define CC2420_FSTST2          0x29
#define CC2420_FSTST3          0x2A
#define CC2420_RXBPFTST        0x2B
#define CC2420_FSMSTATE        0x2C
#define CC2420_ADCTST          0x2D
#define CC2420_DACTST          0x2E
#define CC2420_TOPTST          0x2F
#define CC2420_RESERVED        0x30
#define CC2420_TXFIFO          0x3E
#define CC2420_RXFIFO          0x3F

#define CC2420_RAM_SHORTADR    0x16A
#define CC2420_RAM_PANID       0x168
#define CC2420_RAM_IEEEADR     0x160
#define CC2420_RAM_CBCSTATE    0x150
#define CC2420_RAM_TXNONCE     0x140
#define CC2420_RAM_KEY1        0x130
#define CC2420_RAM_SABUF       0x120
#define CC2420_RAM_RXNONCE     0x110
#define CC2420_RAM_KEY0        0x100
#define CC2420_RAM_RXFIFO      0x080
#define CC2420_RAM_TXFIFO      0x000

// MDMCTRL0 Register Bit Positions
#define CC2420_MDMCTRL0_FRAME        13  // 0 : reject reserved frame types, 1 = accept
#define CC2420_MDMCTRL0_PANCRD       12  // 0 : not a PAN coordinator
#define CC2420_MDMCTRL0_ADRDECODE    11  // 1 : enable address decode
#define CC2420_MDMCTRL0_CCAHIST      8   // 3 bits (8,9,10) : CCA hysteris in db
#define CC2420_MDMCTRL0_CCAMODE      6   // 2 bits (6,7)    : CCA trigger modes
#define CC2420_MDMCTRL0_AUTOCRC      5   // 1 : generate/chk CRC
#define CC2420_MDMCTRL0_AUTOACK      4   // 1 : Ack valid packets
#define CC2420_MDMCTRL0_PREAMBL      0   // 4 bits (0..3): Preamble length

// MDMCTRL1 Register Bit Positions
#define CC2420_MDMCTRL1_CORRTHRESH   6   // 5 bits (6..10) : correlator threshold
#define CC2420_MDMCTRL1_DEMOD_MODE   5   // 0: lock freq after preamble match, 1:
continous udpate
#define CC2420_MDMCTRL1_MODU_MODE    4   // 0: IEEE 802.15.4
#define CC2420_MDMCTRL1_TX_MODE      2   // 2 bits (2,3) : 0: use buffered TXFIFO
#define CC2420_MDMCTRL1_RX_MODE      0   // 2 bits (0,1) : 0: use buffered RXFIFO
```

```
// RSSI Register Bit Positions
#define CC2420_RSSI_CCA_THRESH       8   // 8 bits (8..15) : 2's compl CCA threshold

// TXCTRL Register Bit Positions
#define CC2420_TXCTRL_BUFCUR         14  // 2 bits (14,15) : Tx mixer buffer bias
current
#define CC2420_TXCTRL_TURNARND       13  // wait time after STXON before xmit
#define CC2420_TXCTRL_VAR            11  // 2 bits (11,12) : Varactor array settings
#define CC2420_TXCTRL_XMITCUR        9   // 2 bits (9,10)  : Xmit mixer currents
#define CC2420_TXCTRL_PACUR          6   // 3 bits (6..8)  : PA current
#define CC2420_TXCTRL_PADIFF         5   // 1: Diff PA, 0: Single ended PA
#define CC2420_TXCTRL_PAPWR          0   // 5 bits (0..4): Output PA level

// Mask for the CC2420_TXCTRL_PAPWR register for RF power
#define CC2420_TXCTRL_PAPWR_MASK (0x1F << CC2420_TXCTRL_PAPWR)

// RXCTRL0 Register Bit Positions
#define CC2420_RXCTRL0_BUFCUR        12  // 2 bits (12,13) : Rx mixer buffer bias
current
#define CC2420_RXCTRL0_HILNAG        10  // 2 bits (10,11) : High gain, LNA current
#define CC2420_RXCTRL0_MLNAG          8  // 2 bits (8,9)   : Med gain, LNA current
#define CC2420_RXCTRL0_LOLNAG         6  // 2 bits (6,7)   : Lo gain, LNA current
#define CC2420_RXCTRL0_HICUR          4  // 2 bits (4,5)   : Main high LNA current
#define CC2420_RXCTRL0_MCUR           2  // 2 bits (2,3)   : Main med  LNA current
#define CC2420_RXCTRL0_LOCUR          0  // 2 bits (0,1)   : Main low LNA current

// RXCTRL1 Register Bit Positions
#define CC2420_RXCTRL1_LOCUR         13  // Ref bias current to Rx bandpass filter
#define CC2420_RXCTRL1_MIDCUR        12  // Ref bias current to Rx bandpass filter
#define CC2420_RXCTRL1_LOLOGAIN      11  // LAN low gain mode
#define CC2420_RXCTRL1_MEDLOGAIN     10  // LAN low gain mode
#define CC2420_RXCTRL1_HIHGM          9  // Rx mixers, hi gain mode
#define CC2420_RXCTRL1_MEDHGM         8  // Rx mixers, hi gain mode
#define CC2420_RXCTRL1_LNACAP         6  // 2 bits (6,7) Selects LAN varactor array
setting
#define CC2420_RXCTRL1_RMIXT          4  // 2 bits (4,5) Receiver mixer output current
#define CC2420_RXCTRL1_RMIXV          2  // 2 bits (2,3) VCM level, mixer feedback
#define CC2420_RXCTRL1_RMIXCUR        0  // 2 bits (0,1) Receiver mixer current

// FSCTRL Register Bit Positions
#define CC2420_FSCTRL_LOCK           14 // 2 bits (14,15) # of clocks for synch
#define CC2420_FSCTRL_CALDONE        13 // Read only, =1 if cal done since freq synth
turned on
#define CC2420_FSCTRL_CALRUNING      12 // Read only, =1 if cal in progress
#define CC2420_FSCTRL_LOCKLEN        11 // Synch window pulse width
#define CC2420_FSCTRL_LOCKSTAT       10 // Read only, = 1 if freq synthesizer is loced
#define CC2420_FSCTRL_FREQ            0 // 10 bits, set operating frequency

// SECCTRL0 Register Bit Positions
#define CC2420_SECCTRL0_PROTECT       9 // Protect enable Rx fifo
#define CC2420_SECCTRL0_CBCHEAD       8 // Define 1st byte of CBC-MAC
#define CC2420_SECCTRL0_SAKEYSEL      7 // Stand alone key select
#define CC2420_SECCTRL0_TXKEYSEL      6 // Tx key select
#define CC2420_SECCTRL0_RXKEYSEL      5 // Rx key select
#define CC2420_SECCTRL0_SECM          2 // 2 bits (2..4) # of bytes in CBC-MAX auth
field
#define CC2420_SECCTRL0_SECMODE       0 // Security mode

// SECCTRL1 Register Bit Positions
#define CC2420_SECCTRL1_TXL           8 // 7 bits (8..14) Tx in-line security
#define CC2420_SECCTRL1_RXL           0 // 7 bits (0..7)  Rx in-line security

// BATTMON  Register Bit Positions
#define CC2420_BATTMON_OK             6 // Read only, batter voltage OK
#define CC2420_BATTMON_EN             5 // Enable battery monitor
#define CC2420_BATTMON_VOLT           0 // 5 bits (0..4) Battery toggle voltage

// IOCFG0 Register Bit Positions
#define CC2420_IOCFG0_FIFOPOL        10 // Fifo signal polarity
#define CC2420_IOCFG0_FIFOPPOL        9 // FifoP signal polarity
#define CC2420_IOCFG0_SFD             8 // SFD signal polarity
#define CC2420_IOCFG0_CCAPOL          7 // CCA signal polarity
#define CC2420_IOCFG0_FIFOTHR         0 // 7 bits, (0..6) # of Rx bytes in fifo to trg
fifop

// IOCFG1 Register Bit Positions
```

```
#define CC2420_IOCFG1_HSSD            10 // 2 bits (10,11) HSSD module config
#define CC2420_IOCFG1_SFDMUX           5 // 5 bits (5..9)  SFD multiplexer pin settings
#define CC2420_IOCFG1_CCAMUX           0 // 5 bits (0..4)  CCA multiplexe pin settings

// Current Parameter Arrray Positions
enum{
 CP_MAIN = 0,
 CP_MDMCTRL0,
 CP_MDMCTRL1,
 CP_RSSI,
 CP_SYNCWORD,
 CP_TXCTRL,
 CP_RXCTRL0,
 CP_RXCTRL1,
 CP_FSCTRL,
 CP_SECCTRL0,
 CP_SECCTRL1,
 CP_BATTMON,
 CP_IOCFG0,
 CP_IOCFG1
} ;

// STATUS Bit Posititions
#define CC2420_XOSC16M_STABLE 6
#define CC2420_TX_UNDERFLOW   5
#define CC2420_ENC_BUSY              4
#define CC2420_TX_ACTIVE      3
#define CC2420_LOCK           2
#define CC2420_RSSI_VALID     1

#endif /* _CC2420CONST_H */
```

K.2 CC2420ControlM.nc

```
includes byteorder;

module CC2420ControlM {
  provides {
    interface SplitControl;
    interface CC2420Control;
  }
  uses {
    interface StdControl as HPLChipconControl;
    interface HPLCC2420 as HPLChipcon;
    interface HPLCC2420RAM as HPLChipconRAM;
    interface HPLCC2420Interrupt as CCA;
  }
}
implementation
{

  enum {
    IDLE_STATE = 0,
    INIT_STATE,
    INIT_STATE_DONE,
    START_STATE,
    START_STATE_DONE,
    STOP_STATE,
  };

  uint8_t state = 0;
  norace uint16_t gCurrentParameters[14];

   /************************************************************************
   * SetRegs
   *  - Configure CC2420 registers with current values
   *  - Readback 1st register written to make sure electrical connection OK
   ************************************************************************/
  bool SetRegs(){
    uint16_t data;

    call HPLChipcon.write(CC2420_MAIN,gCurrentParameters[CP_MAIN]);
    call HPLChipcon.write(CC2420_MDMCTRL0, gCurrentParameters[CP_MDMCTRL0]);
    data = call HPLChipcon.read(CC2420_MDMCTRL0);
    if (data != gCurrentParameters[CP_MDMCTRL0]) return FALSE;
```

```
    call HPLChipcon.write(CC2420_MDMCTRL1, gCurrentParameters[CP_MDMCTRL1]);
    call HPLChipcon.write(CC2420_RSSI, gCurrentParameters[CP_RSSI]);
    call HPLChipcon.write(CC2420_SYNCWORD, gCurrentParameters[CP_SYNCWORD]);
    call HPLChipcon.write(CC2420_TXCTRL, gCurrentParameters[CP_TXCTRL]);
    call HPLChipcon.write(CC2420_RXCTRL0, gCurrentParameters[CP_RXCTRL0]);
    call HPLChipcon.write(CC2420_RXCTRL1, gCurrentParameters[CP_RXCTRL1]);
    call HPLChipcon.write(CC2420_FSCTRL, gCurrentParameters[CP_FSCTRL]);

    call HPLChipcon.write(CC2420_SECCTRL0, gCurrentParameters[CP_SECCTRL0]);
    call HPLChipcon.write(CC2420_SECCTRL1, gCurrentParameters[CP_SECCTRL1]);
    call HPLChipcon.write(CC2420_IOCFG0, gCurrentParameters[CP_IOCFG0]);
    call HPLChipcon.write(CC2420_IOCFG1, gCurrentParameters[CP_IOCFG1]);

    call HPLChipcon.cmd(CC2420_SFLUSHTX);    //flush Tx fifo
    call HPLChipcon.cmd(CC2420_SFLUSHRX);

    return TRUE;

  }

  task void taskInitDone() {
    signal SplitControl.initDone();
  }

  task void taskStopDone() {
    signal SplitControl.stopDone();
  }

  task void PostOscillatorOn() {
    //set freq, load regs
    SetRegs();
    call CC2420Control.setShortAddress(TOS_LOCAL_ADDRESS);
    call CC2420Control.TuneManual(((gCurrentParameters[CP_FSCTRL] << CC2420_FSCTRL_FREQ)
& 0x1FF) + 2048);
    atomic state = START_STATE_DONE;
    signal SplitControl.startDone();
  }

  /****************************************************************************
   * Init CC2420 radio:
   *
   ****************************************************************************/
  command result_t SplitControl.init() {

    uint8_t _state = FALSE;

    atomic {
      if (state == IDLE_STATE) {
    state = INIT_STATE;
    _state = TRUE;
      }
    }
    if (!_state)
      return FAIL;

    call HPLChipconControl.init();

    // Set default parameters
    gCurrentParameters[CP_MAIN] = 0xf800;
    gCurrentParameters[CP_MDMCTRL0] = ((0 << CC2420_MDMCTRL0_ADRDECODE) |
       (2 << CC2420_MDMCTRL0_CCAHIST) | (3 << CC2420_MDMCTRL0_CCAMODE)  |
       (1 << CC2420_MDMCTRL0_AUTOCRC) | (2 << CC2420_MDMCTRL0_PREAMBL));

    gCurrentParameters[CP_MDMCTRL1] = 20 << CC2420_MDMCTRL1_CORRTHRESH;

    gCurrentParameters[CP_RSSI] =     0xE080;
    gCurrentParameters[CP_SYNCWORD] = 0xA70F;
    gCurrentParameters[CP_TXCTRL] = ((1 << CC2420_TXCTRL_BUFCUR) |
       (1 << CC2420_TXCTRL_TURNARND) | (3 << CC2420_TXCTRL_PACUR) |
       (1 << CC2420_TXCTRL_PADIFF) | (CC2420_DEF_RFPOWER << CC2420_TXCTRL_PAPWR));

    gCurrentParameters[CP_RXCTRL0] = ((1 << CC2420_RXCTRL0_BUFCUR) |
       (2 << CC2420_RXCTRL0_MLNAG) | (3 << CC2420_RXCTRL0_LOLNAG) |
       (2 << CC2420_RXCTRL0_HICUR) | (1 << CC2420_RXCTRL0_MCUR) |
       (1 << CC2420_RXCTRL0_LOCUR));
```

```
    gCurrentParameters[CP_RXCTRL1]  = ((1 << CC2420_RXCTRL1_LOLOGAIN) |
       (1 << CC2420_RXCTRL1_HIHGM) |  (1 << CC2420_RXCTRL1_LNACAP) |
       (1 << CC2420_RXCTRL1_RMIXT) |  (1 << CC2420_RXCTRL1_RMIXV)  |
       (2 << CC2420_RXCTRL1_RMIXCUR));

    gCurrentParameters[CP_FSCTRL]   = ((1 << CC2420_FSCTRL_LOCK) |
       ((357+5*(CC2420_DEF_CHANNEL-11)) << CC2420_FSCTRL_FREQ));

    gCurrentParameters[CP_SECCTRL0] = ((1 << CC2420_SECCTRL0_CBCHEAD) |
       (1 << CC2420_SECCTRL0_SAKEYSEL)  | (1 << CC2420_SECCTRL0_TXKEYSEL) |
       (1 << CC2420_SECCTRL0_SECM));

    gCurrentParameters[CP_SECCTRL1] = 0;
    gCurrentParameters[CP_BATTMON]  = 0;

    // set fifop threshold to greater than size of tos msg,
    // fifop goes active at end of msg
    gCurrentParameters[CP_IOCFG0]   = (((127) << CC2420_IOCFG0_FIFOTHR) |
        (1 <<CC2420_IOCFG0_FIFOPPOL)) ;

    gCurrentParameters[CP_IOCFG1]   =  0;

    atomic state = INIT_STATE_DONE;
    post taskInitDone();
    return SUCCESS;
  }


  command result_t SplitControl.stop() {
    result_t ok;
    uint8_t _state = FALSE;

    atomic {
      if (state == START_STATE_DONE) {
    state = STOP_STATE;
    _state = TRUE;
      }
    }
    if (!_state)
      return FAIL;

    call HPLChipcon.cmd(CC2420_SXOSCOFF);
    ok = call CCA.disable();
    ok &= call HPLChipconControl.stop();

    TOSH_CLR_CC_RSTN_PIN();
    ok &= call CC2420Control.VREFOff();
    TOSH_SET_CC_RSTN_PIN();

    if (ok)
      post taskStopDone();

    atomic state = INIT_STATE_DONE;
    return ok;
  }

/*****************************************************************************
 * Start CC2420 radio:
 * -Turn on 1.8V voltage regulator, wait for power-up, 0.6msec
 * -Release reset line
 * -Enable CC2420 crystal,          wait for stabilization, 0.9 msec
 *
 *****************************************************************************/

  command result_t SplitControl.start() {
    result_t status;
    uint8_t _state = FALSE;

    atomic {
      if (state == INIT_STATE_DONE) {
    state = START_STATE;
    _state = TRUE;
      }
    }
    if (!_state)
      return FAIL;
```

```
    call HPLChipconControl.start();
    //turn on power
    call CC2420Control.VREFOn();
    // toggle reset
    TOSH_CLR_CC_RSTN_PIN();
    TOSH_wait();
    TOSH_SET_CC_RSTN_PIN();
    TOSH_wait();
    // turn on crystal, takes about 860 usec,
    // chk CC2420 status reg for stablize
    status = call CC2420Control.OscillatorOn();

    return status;
  }

  /**************************************************************************
   * TunePreset
   * -Set CC2420 channel
   * Valid channel values are 11 through 26.
   * The channels are calculated by:
   *  Freq = 2405 + 5(k-11) MHz for k = 11,12,...,26
   * chnl requested 802.15.4 channel
   * return Status of the tune operation
   **************************************************************************/
  command result_t CC2420Control.TunePreset(uint8_t chnl) {
    int fsctrl;
    uint8_t status;

    fsctrl = 357 + 5*(chnl-11);
    gCurrentParameters[CP_FSCTRL] = (gCurrentParameters[CP_FSCTRL] & 0xfc00) | (fsctrl
<< CC2420_FSCTRL_FREQ);
    status = call HPLChipcon.write(CC2420_FSCTRL, gCurrentParameters[CP_FSCTRL]);
    // if the oscillator is started, recalibrate for the new frequency
    // if the oscillator is NOT on, we should not transition to RX mode
    if (status & (1 << CC2420_XOSC16M_STABLE))
      call HPLChipcon.cmd(CC2420_SRXON);
    return SUCCESS;
  }

  /**************************************************************************
   * TuneManual
   * Tune the radio to a given frequency. Frequencies may be set in
   * 1 MHz steps between 2400 MHz and 2483 MHz
   *
   * Desiredfreq The desired frequency, in MHz.
   * Return Status of the tune operation
   **************************************************************************/
  command result_t CC2420Control.TuneManual(uint16_t DesiredFreq) {
    int fsctrl;
    uint8_t status;

    fsctrl = DesiredFreq - 2048;
    gCurrentParameters[CP_FSCTRL] = (gCurrentParameters[CP_FSCTRL] & 0xfc00) | (fsctrl
<< CC2420_FSCTRL_FREQ);
    status = call HPLChipcon.write(CC2420_FSCTRL, gCurrentParameters[CP_FSCTRL]);
    // if the oscillator is started, recalibrate for the new frequency
    // if the oscillator is NOT on, we should not transition to RX mode
    if (status & (1 << CC2420_XOSC16M_STABLE))
      call HPLChipcon.cmd(CC2420_SRXON);
    return SUCCESS;
  }

  /**************************************************************************
   * Get the current frequency of the radio
   */
  command uint16_t CC2420Control.GetFrequency() {
    return ((gCurrentParameters[CP_FSCTRL] & (0x1FF << CC2420_FSCTRL_FREQ))+2048);
  }

  /**************************************************************************
   * Get the current channel of the radio
   */
  command uint8_t CC2420Control.GetPreset() {
    uint16_t _freq = (gCurrentParameters[CP_FSCTRL] & (0x1FF << CC2420_FSCTRL_FREQ));
    _freq = (_freq - 357)/5;
    _freq = _freq + 11;
    return _freq;
```

```
  }

  /****************************************************************************
   * TxMode
   * Shift the CC2420 Radio into transmit mode.
   * return SUCCESS if the radio was successfully switched to TX mode.
   ****************************************************************************/
  async command result_t CC2420Control.TxMode() {
    call HPLChipcon.cmd(CC2420_STXON);
    return SUCCESS;
  }

  /****************************************************************************
   * TxModeOnCCA
   * Shift the CC2420 Radio into transmit mode when the next clear channel
   * is detected.
   *
   * return SUCCESS if the transmit request has been accepted
   ****************************************************************************/
  async command result_t CC2420Control.TxModeOnCCA() {
   call HPLChipcon.cmd(CC2420_STXONCCA);
   return SUCCESS;
  }

  /****************************************************************************
   * RxMode
   * Shift the CC2420 Radio into receive mode
   ****************************************************************************/
  async command result_t CC2420Control.RxMode() {
    call HPLChipcon.cmd(CC2420_SRXON);
    return SUCCESS;
  }

  /****************************************************************************
   * SetRFPower
   * power = 31 => full power    (0dbm)
   *          3 => lowest power  (-25dbm)
   * return SUCCESS if the radio power was successfully set
   ****************************************************************************/
  command result_t CC2420Control.SetRFPower(uint8_t power) {
    gCurrentParameters[CP_TXCTRL] = (gCurrentParameters[CP_TXCTRL] &
(~CC2420_TXCTRL_PAPWR_MASK)) | (power << CC2420_TXCTRL_PAPWR);
    call HPLChipcon.write(CC2420_TXCTRL,gCurrentParameters[CP_TXCTRL]);
    return SUCCESS;
  }

  /****************************************************************************
   * GetRFPower
   * return power seeting
   ****************************************************************************/
  command uint8_t CC2420Control.GetRFPower() {
    return (gCurrentParameters[CP_TXCTRL] & CC2420_TXCTRL_PAPWR_MASK); //rfpower;
  }

  async command result_t CC2420Control.OscillatorOn() {
    uint16_t i;
    uint8_t status;

    i = 0;

    // uncomment to measure the startup time from
    // high to low to high transitions
    // output "1" on the CCA pin
#ifdef CC2420_MEASURE_OSCILLATOR_STARTUP
      call HPLChipcon.write(CC2420_IOCFG1, 31);
      // output oscillator stable on CCA pin
      // error in CC2420 datasheet 1.2: SFDMUX and CCAMUX incorrectly labelled
      TOSH_uwait(50);
#endif

    call HPLChipcon.write(CC2420_IOCFG1, 24);

    // have an event/interrupt triggered when it starts up
    call CCA.startWait(TRUE);

    // start the oscillator
    status = call HPLChipcon.cmd(CC2420_SXOSCON);   //turn-on crystal
```

```
    return SUCCESS;
  }

  async command result_t CC2420Control.OscillatorOff() {
    call HPLChipcon.cmd(CC2420_SXOSCOFF);   //turn-off crystal
    return SUCCESS;
  }

  async command result_t CC2420Control.VREFOn(){
    TOSH_SET_CC_VREN_PIN();                    //turn-on
    // TODO: JP: measure the actual time for VREF to stabilize
    TOSH_uwait(600);  // CC2420 spec: 600us max turn on time
    return SUCCESS;
  }

  async command result_t CC2420Control.VREFOff(){
    TOSH_CLR_CC_VREN_PIN();                    //turn-off
    return SUCCESS;
  }

  async command result_t CC2420Control.enableAutoAck() {
    gCurrentParameters[CP_MDMCTRL0] |= (1 << CC2420_MDMCTRL0_AUTOACK);
    return call HPLChipcon.write(CC2420_MDMCTRL0,gCurrentParameters[CP_MDMCTRL0]);
  }

  async command result_t CC2420Control.disableAutoAck() {
    gCurrentParameters[CP_MDMCTRL0] &= ~(1 << CC2420_MDMCTRL0_AUTOACK);
    return call HPLChipcon.write(CC2420_MDMCTRL0,gCurrentParameters[CP_MDMCTRL0]);
  }

  async command result_t CC2420Control.enableAddrDecode() {
    gCurrentParameters[CP_MDMCTRL0] |= (1 << CC2420_MDMCTRL0_ADRDECODE);
    return call HPLChipcon.write(CC2420_MDMCTRL0,gCurrentParameters[CP_MDMCTRL0]);
  }

  async command result_t CC2420Control.disableAddrDecode() {
    gCurrentParameters[CP_MDMCTRL0] &= ~(1 << CC2420_MDMCTRL0_ADRDECODE);
    return call HPLChipcon.write(CC2420_MDMCTRL0,gCurrentParameters[CP_MDMCTRL0]);
  }

  command result_t CC2420Control.setShortAddress(uint16_t addr) {
    addr = toLSB16(addr);
    return call HPLChipconRAM.write(CC2420_RAM_SHORTADR, 2, (uint8_t*)&addr);
  }

  async event result_t HPLChipconRAM.readDone(uint16_t addr, uint8_t length, uint8_t*
buffer) {
     return SUCCESS;
  }

  async event result_t HPLChipconRAM.writeDone(uint16_t addr, uint8_t length, uint8_t*
buffer) {
     return SUCCESS;
  }

   async event result_t CCA.fired() {
     // reset the CCA pin back to the CCA function
     call HPLChipcon.write(CC2420_IOCFG1, 0);
     post PostOscillatorOn();
     return FAIL;
   }

}
```

Anexo L: El compilador de nesC

En función del tipo de programador que usaremos tendremos que utilizar la instrucción make, de una forma de otra, esta es la que se encarga de llamar al compilador nesC.

Si utilizáramos el programador serie MIB510 tendríamos que compilar con la siguiente instrucción:

```
make micaz install.X mib510,com1
```

La X va referida a la identificación de nodo que queremos que tenga el nodo que estamos compilando, mib510 identifica la placa programadora que utilizamos y com1 identifica el puerto por el cual tenemos conectado nuestro programador.

Si utilizáramos el programador ethernet MIB600, tendremos que aplicar parámetros distintos.

```
make micaz install.X eprb,192.168.1.1
```

Aquí hemos cambiado el parámetro del programador por *eprb* y en vez de redireccionar a un puerto serie, redireccionamos a una dirección IP.

En ambos casos establecemos la misma plataforma (micaz), para que el compilador pueda adaptar el funcionamiento de los distintos componentes al hardware.

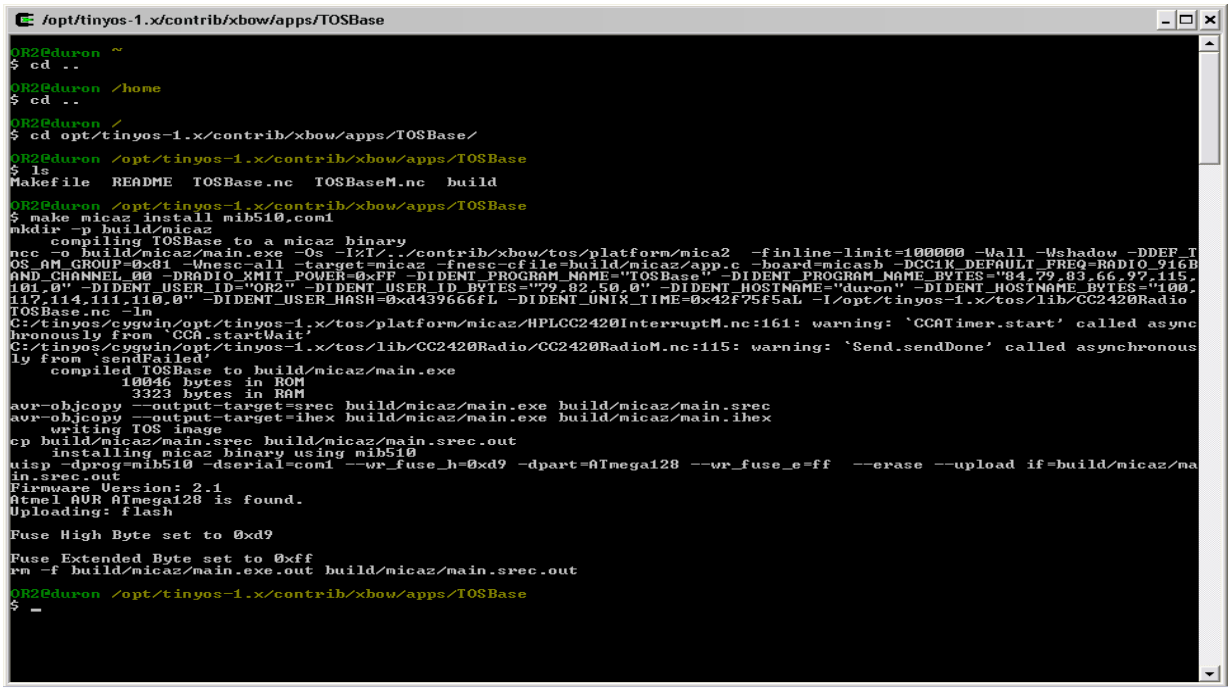


Fig L.1. Compilación hecha en cygwin con programado mib510

Una vez realizado únicamente tendríamos que esperar que se compile y observar si hay errores de compilación o advertencias como en la figura L.1.