Facultad de Informática

Universidad Nacional de La Plata

Construcción de Aplicaciones en Redes de Sensores Basado en CoAP

Alumno:

Moisés E. Coronado D.

Director

Ing. Luis Marrone

"Trabajo presentado para obtener el Grado de Magister en Redes de Datos”

"Facultad de Informática - Universidad Nacional de La Plata"

Enero - 2020

Licencia Documento

Contenido

Índice de Figuras

1. Introducción

Un término o concepto que hoy en día está de moda es el “Internet de las Cosas” y se presenta como la nueva revolución de Internet. Este internet que utilizamos para enviar o recibir email, leer periódicos, relacionarnos con otras personas mediante redes sociales queda atrás, ahora buscamos relacionarnos con los dispositivos que nos rodean, por ejemplo, deseamos que el refrigerador nos comunique cuando falte algún alimento para el desayuno de mañana, es en este punto donde las redes de sensores son el pilar de esta nueva evolución de internet.

Los avances en redes inalámbricas, micro-fabricación, integración y microprocesadores embebidos permiten que el despliegue de esta nueva generación de redes pueda tener aplicaciones tanto comerciales como militares. Esta tecnología promete revolucionar la forma de vida y de trabajo a tal punto que permitirá relacionarnos de mejor forma con el medio, tomando decisiones con mucha más información, por ejemplo podemos desplegar vía aérea sensores de temperatura sobre bosques de difícil acceso, estos sensores se mantendrán en estado latente hasta que se detecte una variación en la temperatura, cuando ocurra algún evento como por ejemplo un cambio de temperatura, el sensor emitirá una señal que será transmitida por los mismos sensores adyacentes hasta alguna estación remota conectada a internet, donde se dará alerta a los organismos pertinentes sobre un posible foco de incendio, si a estos sensores se les proporciona una unidad de posicionamiento geográfico podrán informar la posición exacta donde ocurrió el evento.

Estos nuevos avances tecnológicos traen consigo nuevos desafíos para la informática, por ejemplo: el diseño de algoritmos de representación y procesamiento de la información, desarrollo de nuevos protocolos, estándares de comunicación, herramientas de procesamiento de señales, sistemas de almacenamiento y nuevas metodologías de construcción de aplicaciones.

A diferencia de las redes tradicionales, una red de sensores está sujeta a restricción en cuanto a capacidad de procesamiento y energía.

Una red de sensores extiende los límites de internet llegando a nuevos lugares y dispositivos. Esta nueva red resultante es de mayor tamaño y más dinámica, haciendo que TCP/IP sea exigido a niveles que hacen necesarios cambios en algunas de sus capas para dar soporte al nuevo tráfico que está surgiendo de estas nuevas redes.

Este proyecto cobra vida por la necesidad de conocimiento sobre esta nueva internet, y para ello se analizan las técnicas de construcción de aplicaciones.

En la primera parte de este documento describiremos el Hardware que utilizaremos con una mirada general profundizando solo en aquellos elementos que intervendrán en el trabajo de tesis.

En una segunda parte se describe el estudio de los Sistemas Operativos que fueron analizados con objetivo de seleccionar aquel que se ajuste de mejor forma al trabajo de tesis, este análisis se realiza de diferentes puntos de vista.

Una sección final donde se estudia la forma de construcción de aplicaciones en diferentes Sistemas Operativos de tal forma de proponer una línea de investigación que permita ser un aporte al desarrollo de estas tecnologías.

2. Presentación del Proyecto.

2.1. Planteamiento del Problema

La filosofía del internet de las cosas es conectar todo tipo de dispositivo de todas las tecnologías, y pretende que su desarrollo se convierta en una tendencia para todos los sectores, comprometer cada vez más actores a unirse a los esfuerzos de la interconexión total, pero la simple conexión no es suficiente, se debe ser capaz de proporcionar una cierta inteligencia a estos dispositivos, de tal manera que puedan interactuar y a partir de los datos que ellos mismos generan o que otros dispositivos comparten, poder obtener información, es en este punto que los algoritmos o aplicaciones cobran vital importancia porque son estos los que proporcionan la inteligencia en todos los ámbitos, en temas de comunicación, recolección y/o procesamiento, siendo una de las principales exigencias un alto nivel de eficiencia desde el punto de vista de la utilización de los recursos de programación.

Desde un punto de vista general las aplicaciones para el internet de las cosas tienen los mismos requerimientos de una red tradicional: recolectar, transportar y almacenar. La diferencia está en la complejidad que existe en las aplicaciones, dado el escenario del Internet de las Cosas en que tenemos un sin número de equipos con muy bajos recursos (procesamiento, memoria, comunicación, energía, etc.). Entonces cualquier marco para el desarrollo de aplicaciones debe considerar estos puntos, ser eficiente con los recursos y junto con ello, debe ser compatible con los dispositivos y las nuevas tecnologías en desarrollo o despliegue, debido a que la idea fundamental es la interconexión total (Zach Shelby, 2009)

Los algoritmos proporcionan a una red de sensores la inteligencia necesaria que permiten respuestas en un tiempo corto, con el objetivo de poder recopilar información que será utilizada en la aplicación y a la vez ahorra energía. Esto requiere una nueva forma de hacer las aplicaciones, nuevas herramientas, mejorar las estrategias o métodos para construir aplicaciones, incluso se tienen que eliminar algunos elementos que bajo este nuevo escenario no son útiles, esto representa un desafío que al ser resuelto proporcionaría un nivel de dominio del tema que podría ser aplicado a una amplia gama de soluciones y si esto lo contextualizamos en un nivel local sería un paso hacia un área no cubierta por el Magister de Redes de Datos de la UNLP, que tiene que ver con el desarrollo de aplicaciones para las redes de datos en general.

Considerando los puntos anteriores la pregunta de investigación que se plantea es: ¿Cuáles son los elementos que se deben tener en cuenta al momento de construir aplicaciones para el Internet de las Cosas?

2.2. Hipótesis

Se plantea como hipótesis, que es posible desarrollar una solución basada en redes de sensores, siendo viable construir e implementar una red de bajo costo, utilizando técnicas, metodologías y tecnologías emergentes, con una equivalencia similar a las aplicaciones de la internet tradicional.

2.3. Justificación

La motivación para el desarrollo de este proyecto surge luego de la lectura de uno de los primeros libros de 6lowpan, el cual presenta un análisis riguroso del protocolo con un enfoque similar a los análisis efectuados durante el desarrollo de los cursos de magister en la UNLP.

Este Trabajo de Tesis busca ser el punto de partida para el estudio de las redes sensores principalmente el protocolo CoAP y sus Aplicaciones, para ello se requiere un estudio de los Sistemas Operativos, que son la componente de software principal de toda red de sensores, son los que darán soporte a las aplicaciones. (Zach Shelby, 2009).

La institución en la que se desarrollará este proyecto (Ministerio de Obras Publicas) forma parte de un grupo de investigación y desarrollo en la cual participan organizaciones de variados ámbitos (educacionales, empresas privadas y estatales), esta asociación cuenta con un departamento de investigación que entrega recursos para nuevos proyectos, y este proyecto fue acogido por una de estas instituciones.

Esta investigación será útil para estudiar posibles líneas de desarrollo en alguna institución que trabaje con estas tecnologías, como una posible solución a problemas que se están enfrentando en la actualidad, en el ámbito agrícola el tema de las sequias, en el terreno del transporte monitoreo de infraestructura vial y temas de seguridad personal entre otros.

Si bien este proyecto puede ser utilizado en forma directa en las áreas descritas en el párrafo anterior, tener acceso a información exploratoria ayudar a que surjan nuevos proyectos que apliquen y requieran de un mayor desarrollo de los temas planteados en este trabajo.

3. La Nueva Internet.

Si al internet que hoy en día conocemos le superponemos las redes de sensores el resultado se aprecia en la *Figura 1: Arquitectura de una nueva Internet*, en términos técnicos estas nuevas redes se enlazan instalando un Router de borde, apoyado por un sistema de almacenamiento con funciones de consulta y búsqueda, para la extracción de información a partir de los datos generados por los nodos sensores. Es de esta forma como se pretende hacer convivir las redes de sensores con las redes tradicionales.

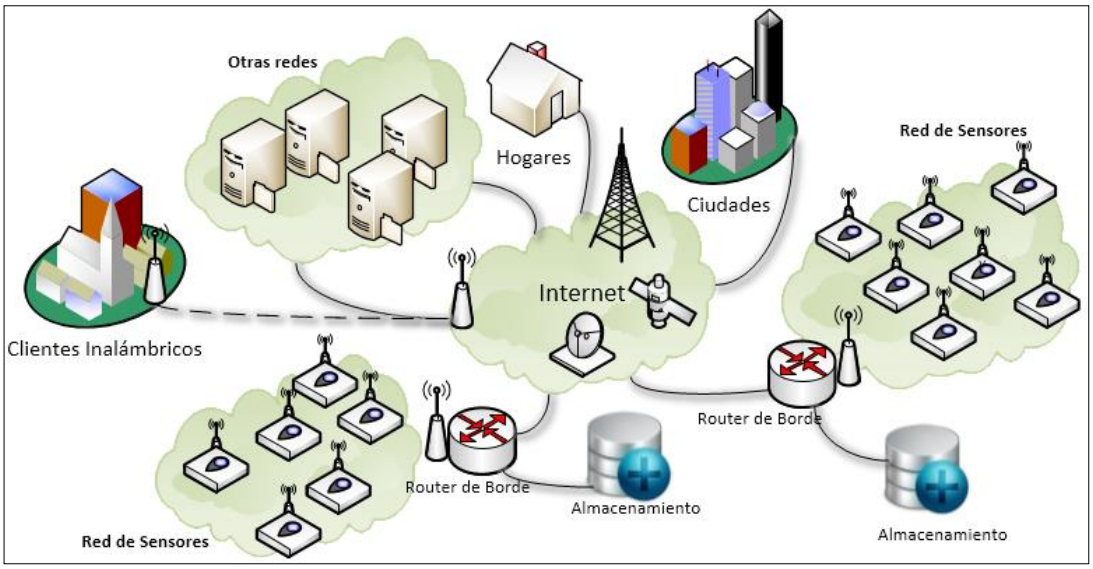

Figura 1 - Arquitectura de una Nueva Internet

En la actualidad en el mercado existe una amplia gama de dispositivos sensores tan pequeños como una moneda y con funcionalidades básicas. Si queremos dispositivos con mayores funcionalidades podemos encontrar equipos con su propio sistema operativo, el cual los hace funcionar de forma autónoma, como por ejemplo las motas TelosB que cuentan con una serie de características que las hacen muy versátiles, y pueden ser utilizadas en una infinidad de aplicaciones basadas en sistemas de sensores inteligentes (Moore, 2013).

4. Glosario

Para mejorar la comprensión de este documento a continuación se describen algunos términos que son utilizados con regularidad en los diferentes apartados.

- Sensor: Dispositivo capaz de transformar o convertir un fenómeno físico como por ejemplo luz, sonido o movimiento en una señal eléctrica u otro tipo de señal que puede ser manipulada por otro dispositivo.

- Nodo Sensor: Es la unidad básica de una red de sensores, tiene uno o más sensores, procesador, memoria, transmisor inalámbrico y una unidad de energía. Cuando en la placa se tiene un solo sensor en alguna literatura el nodo sensor es referenciado como sensor, creando alguna confusión, otros términos con el cual se denomina un nodo sensor es mota porque se hace la analogía con una partícula de polvo.

- Routing: Es el proceso de determinar el camino que debe seguir un paquete de acuerdo a su red de destino.

- Router de Borde o Gateway: Puente entre redes, en el contexto de las redes se refiere al punto de acceso de las aplicaciones para su comunicación con los servicios externos o de otras redes.

- TelosB: Crossbow's TelosB Mote, es un dispositivo sensor (nodo sensor), diseñado para estudio y experimentación Incluye: Capacidad para puerto USB, una antena integrada que trabaja con el protocolo IEEE 802.15.4, tiene bajo consumo de energía, cuenta con la posibilidad de incluir más sensores temperatura, presión.

- Mota: Es una denominación para los dispositivos sensores (redes sensores) que surge de la comparación de estos dispositivos con una partícula de polvo diseminada por el ambiente.

- 6LoWPAN: (IPv6 over Low power Wireless Personal Area Networks) es un estándar utilizado en redes de sensores, que posibilita la comunicación de estos dispositivos con otras redes basadas en IP.

- ContikiOS: Sistema operativo para redes de sensores desarrollados a partir de una tesis de doctorado de Adam Dunkels, que tiene como principal característica el soporte completo para redes basadas en IP.

- TinyOS: Sistema Operativo para redes de sensores con una amplia comunidad de desarrollo.

- Protothreads: Es una implementación de hilos para el desarrollo de aplicaciones en dispositivos de muy bajos recursos.

- Macro en C: En programación en C son utilizados para hacer alias que podemos incluir en nuestro código, el que al momento de compilar, será reemplazado por lo que hayamos definido.

- BSD: Berkeley Software Distribution o BSD (en español, distribución de software Berkeley) fue un sistema operativo derivado del sistema Unix nacido a partir de los aportes realizados a ese sistema por la Universidad de California en Berkeley.

- Plataforma: Se refiere a los distintos tipos de hardware que existen en el mercado y que son utilizados como nodos sensores.

- Makefiles: Son los Archivos de texto que utiliza GNU/make para llevar la gestión de la compilación de programas.

- CoAP (Constrained Application Protocol): Protocolo a nivel de aplicación pensado para ser usado en dispositivos electrónicos simples, y que proporciona la capacidad de comunicación sobre Internet.

5. Sensores Inteligentes

En términos generales un sensor inteligente es un equipo en que cuenta con: uno o más elementos sensores y algún método de procesamiento de señal que son integrados en la misma pastilla de silicio y forman un pequeño sistema miniaturizado (Ruiz, 1999).

Se debe notar que los sensores inteligentes se han fabricado en ambos dominios: El digital y el analógico. El caso analógico es importante debido a que sólo este tipo de señal está normalizado a nivel industrial, tanto en corriente como en tensión. En el dominio digital se cuenta con múltiples interfaces que no permiten una interoperabilidad al 100%, y los proyectos hoy en día son acotados a un solo tipo de dispositivo para una solución, es decir, no es posible asegurar la interoperabilidad entre equipos y es recomendable utilizar un modelo en cada solución, esto se intenta solucionar muchas veces instalando sensores inteligentes que permiten uniformizar el tipo de salida de la señal ya sea entregando una señal analógica normalizada, o una digital (Nikoi, 2000).

Según IEEE un sensor inteligente debe tener una estructura completa como se muestra en la figura siguiente.

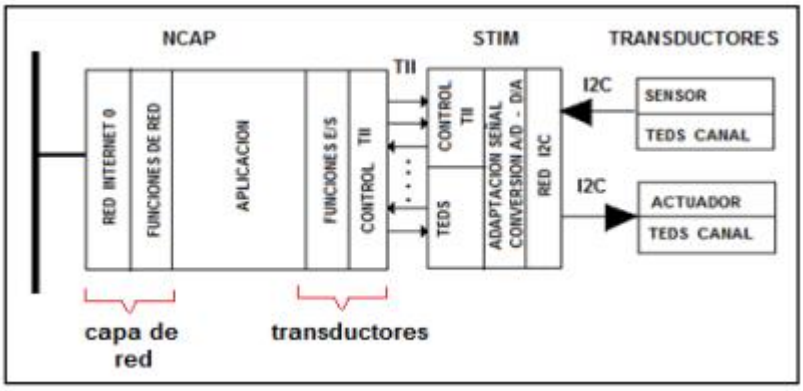

Figura 2 - Estructura de un nodo inteligente basado en el estándar IEEE 1451

La Figura 2 muestra una unidad funcional compuesta por: El Network Capable Application Processor (NCAP) unidad que se encarga de la comunicación, el Smart Transducer Interface Module (STIM) que se encarga del procesamiento de la información y finalmente los transductores (Sensor – Actuador) que son los encargados de medir y/o controlar/alterar el entorno, a este dispositivo la IEEE denomina nodo inteligente, siendo inteligente por ser autónomo y tener la capacidad suficiente para realizar una actividad específica independiente de la red, pero simultáneamente forma parte de una red compuesta por otros nodos inteligentes, además posee la capacidad para interactuar con ellos y constituir de esta manera un sistema más complejo y con mayor cantidad de funcionalidades o aplicaciones que comúnmente se denomina Red de Sensores (Zach Shelby, C. B., 2009).

6. Red de sensores

Una red de sensores es un conjunto de dispositivos autónomos, que cooperan entre sí para efectuar tareas de monitoreo y/o control de variables, que pueden ir desde algo tan simple como el monitoreo de temperatura del ambiente hasta la detección de elementos nocivos. Los orígenes de estas tecnologías son relativamente nuevos y está asociado al mundo militar, (SOSUS[1]) otras aplicaciones como por ejemplo monitoreo de la producción en industrias, aplicaciones de domótica, sensores agrícolas, elemento de seguridad, estaciones medioambientales etc. hacen que los trabajos de investigación y desarrollo apunten en el sentido de las redes de sensores (Waher,2015).

6.1. Ventajas de las redes de sensores

Las redes de sensores tienen muy marcadas ventajas respecto a las redes centralizadas, por ejemplo, en términos de comunicaciones una red de sensores es capaz de mitigar los efectos perjudiciales del índice señal ruido (SNR), manipulando la distancia entre sus nodos que generalmente son móviles y autónomos. Otro punto que se puede mencionar como ventaja es la eficiencia en la administración de la energía en la comunicación producto de la configuración de topologías multisalto, aumentando las distancias cubiertas sin necesidad de equipos de gran potencia y de consumo constante, si a esto agregamos que durante la transmisión de nodo a nodo los datos pueden ser alterados (mejorados) con nueva información proporcionada por los demás nodos sensores, tenemos enriquecimiento de información a medida que esta viaja. Pero la ventaja más significativa es la escalabilidad y robustez de la red que proporciona la gran cantidad de nodos; por ejemplo, la robustez se aprecia al comunicar dos puntos de la red que ahora tienen una gran cantidad de caminos posible, y la escalabilidad podemos visualizarla en los algoritmos descentralizados de las aplicaciones que proporcionan una gran autonomía a cada nodo (Ruiz, M. l. ,1999).

Este trabajo de tesis tiene la mirada puesta en del desarrollo de las aplicaciones en una red de sensores, es por esto que primero se analizarán desde un punto de vista práctico las redes de sensores.

6.2. Arquitectura de dispositivos para Redes de Sensores

En esta sección se describen los dispositivos que conforman una red de sensores, sus formas de despliegue y su funcionamiento a nivel de hardware.

[1] **SOSUS**, acrónimo de **Sound Surveillance System**, del inglés: **Sistema de Vigilancia Sónica**. Consiste en una cadena de puestos de escucha submarinos que se reparten en una línea que va desde Groenlandia hasta el Reino Unido

6.3. Sensores Inteligentes

En términos generales un sensor inteligente es un equipo en que cuenta con: uno o más elementos sensores y algún método de procesamiento de señal que son integrados en la misma pastilla de silicio y forman un pequeño sistema miniaturizado (Ruiz, 1999).

Se debe notar que los sensores inteligentes se han fabricado en ambos dominios: El digital y el analógico. El caso analógico es importante debido a que sólo este tipo de señal está normalizado a nivel industrial, tanto en corriente como en tensión. En el dominio digital se cuenta con múltiples interfaces que no permiten una interoperabilidad al 100%, y los proyectos hoy en día son acotados a un solo tipo de dispositivo para una solución, es decir, no es posible asegurar la interoperabilidad entre equipos y es recomendable utilizar un modelo en cada solución, esto se intenta solucionar muchas veces instalando sensores inteligentes que permiten uniformizar el tipo de salida de la señal ya sea entregando una señal analógica normalizada, o una digital (Nikoi, 2000).

Según IEEE un sensor inteligente debe tener una estructura completa como se muestra en la figura siguiente.

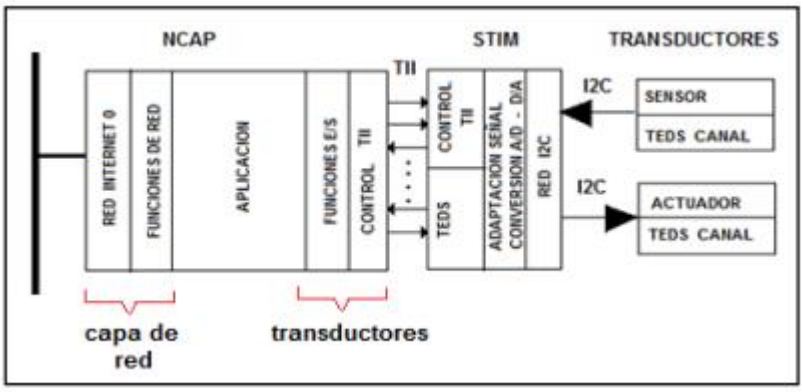

Figura 3 - Estructura de un nodo inteligente basado en el estándar IEEE 1451

La Figura 2 muestra una unidad funcional compuesta por: El Network Capable Application Processor (NCAP) unidad que se encarga de la comunicación, el Smart Transducer Interface Module (STIM) que se encarga del procesamiento de la información y finalmente los transductores (Sensor – Actuador) que son los encargados de medir y/o controlar/alterar el entorno, a este dispositivo la IEEE denomina nodo inteligente, siendo inteligente por ser autónomo y tener la capacidad suficiente para realizar una actividad específica independiente de la red, pero simultáneamente forma parte de una red compuesta por otros nodos inteligentes, además posee la capacidad para interactuar con ellos y constituir de esta manera un sistema más complejo y con mayor cantidad de funcionalidades o aplicaciones que comúnmente se denomina Red de Sensores (Zach Shelby, C. B., 2009).

6.4. Estructura de sensores

En términos simples un nodo sensor está constituido por un transceptor de radio, un pequeño micro controlador, un circuito analógico, una fuente de energía comúnmente una batería y por último una memoria. En la siguiente figura se aprecia la interconexión entre cada uno de sus componentes.

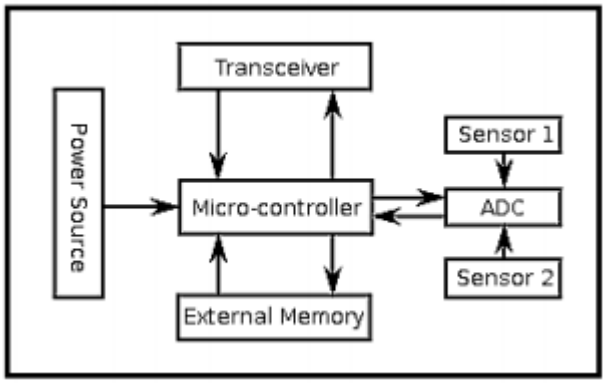

Figura 4 - Estructura de Sensores (Nikoi, 2000)

El tamaño de los sensores varía de acuerdo a la evolución de la tecnología, en este sentido, hoy en día tenemos dispositivos tan pequeños como los denominados biosensores que monitorean los latidos del corazón con un tamaño de unos pocos milímetros, teniendo este ejemplo como base se puede proyectar una extensa lista de posibles aplicaciones.

En general, el principal objetivo de un sensor es la recolección de datos del entorno y el envío a uno o varios recolectores para su análisis y en determinados casos, es posible incluir mecanismos de actuación que responden ante ciertas condiciones del entorno, esto se denomina redes inalámbricas de sensores y actuadores.

La evolución de sensores hoy en día está limitada por las restricciones de consumo y capacidades de comunicación de los nodos. Esto obliga a desarrollar componentes hardware de muy bajo consumo y con considerables limitaciones de cómputo, de la misma forma el software se ve restringido a algoritmos y protocolos de bajo consumo, esto hace que las redes de hoy en día sean parte principalmente de sistemas de monitoreo orientada a la transferencia de la información y no al procesamiento de ella. Los sistemas de almacenamiento de energía están teniendo un fuerte auge en temas de miniaturización que influirá positivamente en el desarrollo y despliegue de las redes de sensores.

6.5. Estructura y Funcionamiento básico de la red

En un escenario común del despliegue de redes de sensores se identifican dos tipos de nodos, nodos sensores o host (H) y router (R) [Figura 4].

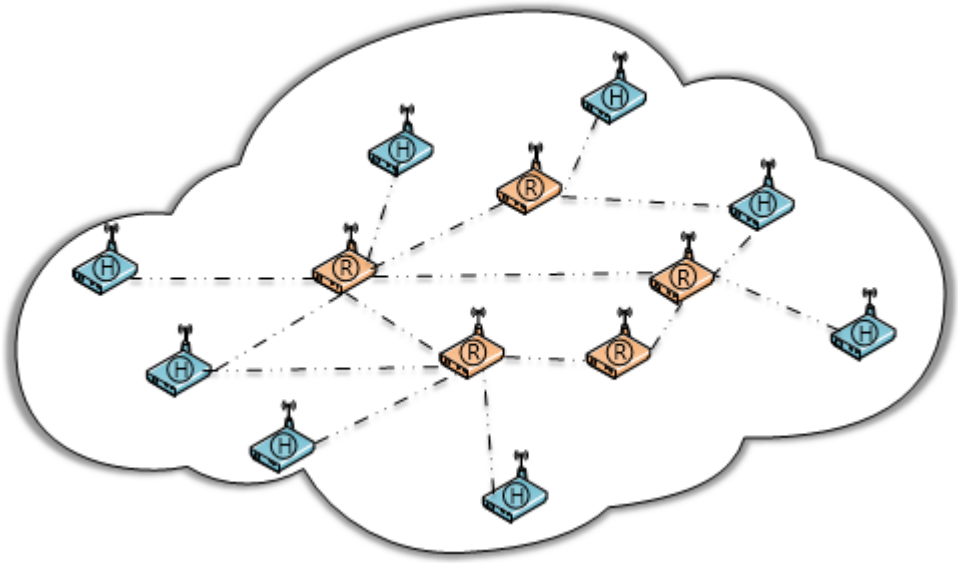

Figura 5 - Despliegue común de una red de sensores

Los Router (R) son la columna vertebral de la red, su principal tarea es la distribución de los paquetes entre el origen y el destino de la transmisión, aunque a veces pueden funcionar como enlace a redes heterogéneas adyacentes que no tienen por qué ser de la misma naturaleza inalámbrica. Por otro lado, los nodos terminales o Host (H) son los encargados de generar la información que transmite la red a partir de las medidas de los sensores incorporados en el hardware de los nodos.

La topología lógica de la red de sensores está constituida típicamente por redes ad-hoc sin infraestructura establecida y sin una administración central, de esta forma los router utilizan algoritmos que encaminan los datos mediante saltos punto a punto entre distintos nodos cercanos hacia la estación central, donde la información es almacenada y/o procesada.

Los nodos pueden desempeñar dos papeles, como emisor o receptor en caso de ser necesario, esto se denomina auto configuración, además ofrecen servicio de enrutamiento entre nodos sin visión directa, para lograr esto se registran datos de cada uno de los sensores o nodos locales.

Otro punto importante de mencionar es la gestión energética que se realiza en los nodos de tal manera de optimizar la energía almacenada, generalmente los nodos disponen de varios modos de funcionamiento que regulan el consumo de energía, adaptándose con bastante idoneidad a los estados de transmisión y reposo de la red. Estos modos de bajo consumo no sólo se limitan a la CPU sino también a los periféricos integrados, de tal forma que es posible desactivar aquellos que no se usan, ajustando adecuadamente las capacidades de procesamiento para cada aplicación (Philip Levis, 2008).

6.6. Estándares para Redes de Sensores

Para crear una red de sensores existe una gran variedad de protocolos de comunicación en radio frecuencias, algunos de estos protocolos son propietarios de una organización en particular y otros son un estándar de la industria. Son dos las iniciativas que hoy están tomando la delantera con propuestas que buscan en el liderazgo en el área, por un lado, se encuentra el protocolo propietario ZigBee y por otro el estándar IEEE 802.15.4.

6.6.1. Estándar 802.15.4

802.15.4 (Pat Kinney, 2003) es un estándar de comunicaciones inalámbricas desarrollado por el IEEE (Institute for Electrical and Electronics Engineers) para ser utilizado en comunicaciones de baja velocidad, que simplifica la conectividad entre nodos y optimiza el uso de la batería.

El estándar 802.15.4 define la comunicación dentro de los 868 a 868.8 MHz, 902 a 928 MHz o 2.400-2.4835 GHz. Los dispositivos pueden hacer uso de estas bandas, pero la más comúnmente utilizada por los aparatos que cumplen con 802.15.4, es la banda de 2.4 GHz que en la mayoría de los países es libre.

El estándar 802.15.4 permite la comunicación con una configuración punto a punto o punto multipunto. Una aplicación típica consiste en contar con un coordinador central con múltiples nodos conectados que proporcionan los valores de las variables censadas.

6.6.2. ZigBee

ZigBee es un protocolo que utiliza el estándar 802.15.4 como base y le agrega funcionalidades de ruteo y de red. El protocolo ZigBee fue desarrollado por la ZigBee Alliance, esta alianza está conformada por un grupo de compañías que colaboran en el desarrollo de un protocolo que pueda ser utilizado con aplicaciones industriales de baja velocidad.

El diseño de una red ZigBee agrega las características de una red mallada subyacente a 802.15.4, esto permite que los nodos se comuniquen entre sí, independientemente del punto de acceso. De esta manera los nodos de la red pueden no enviar directamente sus paquetes al punto de acceso, sino que pueden delegarlos a otros nodos de la red para que lleguen a su destino.

En resumen, estas son las dos propuestas más importantes, las dos iniciativas buscan la conectividad en redes de sensores, ZigBee lo hace con una mirada industrial y con una propuesta que involucra toda la plataforma desde el hardware a las aplicaciones. Por otro lado, IEEE se centra principalmente en la capa física junto a la de acceso a datos y es IETF (Internet Engineering Task Force) la entidad que trabaja en las otras capas superiores.

Aunque la combinación del protocolo Zigbee sobre IEEE 802.15.4 proporcionan una arquitectura de protocolo bastante completa para trabajar en redes de sensores, Zigbee presenta el inconveniente de ser incompatible a la hora de lograr una interconexión con redes externas debido a que su esquema de direccionamiento, no es compatible con el protocolo IP y esto im-

pide una conexión directa entre los dispositivos de estas redes, la puesta en el mercado de ZigBee 3.0 (Alliance, Z. 2014) promete una interoperabilidad pero para efecto de este trabajo no se logró obtener un chip con esta tecnología.

Por otro lado, el desarrollo del protocolo IP versión 6 ha demostrado ofrecer mayores y mejores ventajas que IPv4, por ejemplo, un espacio de direccionamiento mucho más escalable, autoconfiguración sin estado, entre otras, con esto pretende convertirse en el protocolo preferido para lograr la interconexión directa de las redes de sensores inalámbricas con las redes externas, resolviendo de esta manera la limitante que posee ZigBee en este aspecto.

Sin embargo, el uso de IPv6 en este tipo de redes impone ciertos requerimientos como el incremento de tamaño de las direcciones IPv6 y del MTU (Maximum Transmission Unit) en 1280 bytes. Por tal motivo, se dio origen a 6LoWPAN con el fin de eliminar los inconvenientes que tenían los paquetes IPv6 para ser transportados sobre las redes inalámbricas de bajo consumo (LowPAN), específicamente en las redes basadas en IEEE 802.15.4. (Alliance, Z ,2014)

6.6.3. 6LoWPAN

El origen del estándar 6LoWPAN está en los esfuerzos de eliminar los inconvenientes que tenían los paquetes IPv6 para ser transportados sobre las redes inalámbricas de bajo consumo (LoWPAN), específicamente en las redes basadas en IEEE 802.15.4. Para lograr este objetivo, se definió una capa de adaptación, que tratara los requerimientos impuestos por IPv6 como el incremento de tamaño de las direcciones IPv6 y del MTU en 1280 bytes, de la misma forma se crearon un conjunto de encabezados que permitieron una codificación más eficiente.

En la siguiente figura [Figura 5] se aprecia una comparativa de los dos protocolos que fueron descritos en este apartado.

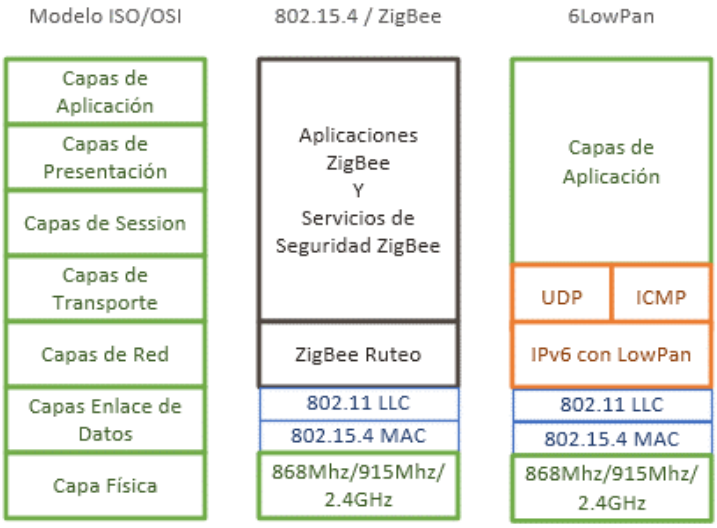

Figura 6 - Comparativa de Tecnologías y Estándares

6.6.4. Protocolo CAP (Compact Application Protocol)

Esta es una iniciativa que busca implementar un punto de encuentro entre 6LoWPAN y ZigBee rescatando de 6LoWPAN la compatibilidad con IPv6 y de ZigBee rescatar la penetración que ha tenido en los dispositivos para la industria (estandarizaciones) de tal forma que los fabricantes no sean afectados por un cambio radical en el protocolo.

Hasta el momento es una propuesta borrador que ha sido sometida a la IEEE para su evaluación, en ella se describe una adaptación UDP/IP del IEEE 802.15.4 basado en la capa de aplicación del protocolo ZigBee. Esto permitiría que los hosts IP puedan comunicarse usando los perfiles de aplicación de ZigBee y modelos de datos descritos por el protocolo sobre una amplia gama de dispositivos de diversos fabricantes.

Este protocolo escapa a los objetivos de este trabajo de tesis, pero se pude obtener más información en (David E. Culler, 2009)

7. Plataforma TelosB

Los elementos básicos que conforman una Red de Sensores se denominan nodos, comúnmente se les denomina también motas (hace relación a las pequeñas partícula de polvo presentes en el ambiente), y pueden clasificarse en dos categorías: nodos activos (que miden o actúan), y nodos pasivos (que transmiten información o que la recolectan como concentradores de datos), en el desarrollo de este trabajo de tesis se utiliza un solo modelo de nodo, que cumple con el requerimiento de ser lo más estándar posible y que no está restringidos a ningún fabricante en especial, es por este motivo que se hace un análisis de los dispositivos presentes en el mercado tanto Argentino como Chileno, luego de indagar varias opciones se opta por la importación de un dispositivo TelosB.

La Mota TelosB es una plataforma Open Hardware desarrollada en la University of California, Berkeley con propósitos de investigación y hoy en día incluso está siendo utilizada en aplicaciones en entornos productivos. Son dispositivos inalámbricos que permiten muchas funcionalidades. Entre ellas están la medición de parámetros en sus sensores (temperatura, humedad y luminosidad) además es posible adosarle otros dispositivos que permitan control sobre variables, esto es posible porque cuenta de manera opcional con varias interfaces que permiten incrementar sus funcionalidades.

Según las especificaciones las motas TelosB permiten una tasa de transferencia de datos por medio del chip de radio de 250kbps. Trabaja en la banda de frecuencia de 2.4 GHz a 2.483Ghz. Los Equipos TelosB son fácilmente programables gracias a su interfaz USB. Incorporan en su interior un procesador TI MSP430 que opera a 8MHz y una RAM de 10kbytes. La memoria flash para los programas compilados es de 48kbytes. Y posee también una memoria flash de 1Mbyte para almacenamiento de aplicaciones.

Para el desarrollo de este trabajo de tesis se utiliza un dispositivo basado en las especificaciones originales de las TelosB, construida por la empresa española Advanticsys[2] bajo el nombre CM5000 [Figura 6].

[2] https://www.advanticsys.com

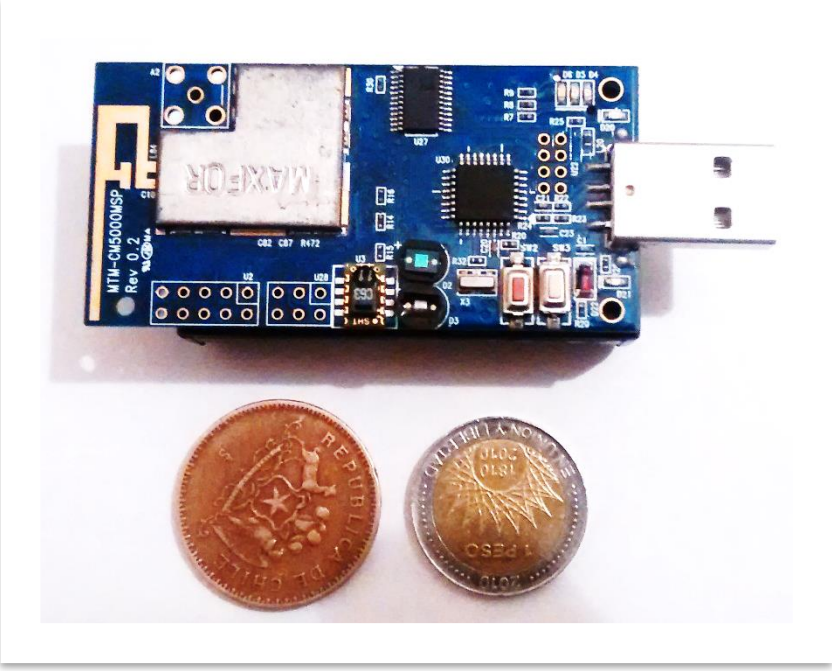

Figura 7 - Dispositivo Sensor CM5000 Basado en TelosB (Comparativa Moneda CL & AR)

Las principales características que influyeron en la elección de esta plataforma fueron en un primer lugar su compatibilidad con TinyOS y ContikiOS que son los dos sistemas operativos con mayor desarrollo en los últimos años. En segundo lugar, las características de open hardware que presenta esta plataforma que hace posible la implementación de soluciones sin mayores problemas de licencias, proporciona mayor libertad para posibles aplicaciones y compatibilidad en el tiempo.

Esta Mota posee una placa de sensores que incorpora: sensores de luz, temperatura y humedad. Esto facilita la realización de experimentos y el aprendizaje al no tener que preocuparse por la electrónica y el conexionado de elementos.

Otras características que son relevantes para el desarrollo de este trabajo de tesis son:

- Compatibilidad con IEEE 802.15.4.
- Una Interface USB que facilita la configuración, monitoreo y carga de aplicaciones
- Dos tipos de alimentación uno por medio pilas 2xAA y vía USB.

Un análisis técnico con mayor detalle se presenta en la ahora tesis de Grado de Ayyaz Mahmood Salekin Imran (Ayyaz Mahmood, 2016)

Desde un punto de vista técnico, el diagrama de bloques [Figura 7] del equipo se aprecia en la siguiente figura

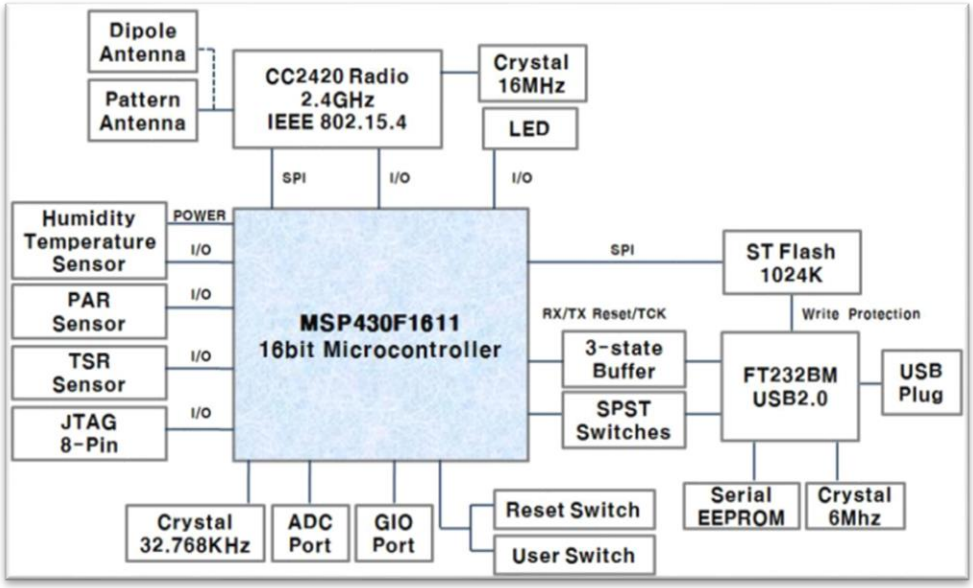

Figura 8 - Diagrama de Bloques CM5000

La Unidad de procesamiento de la mota CM5000 es un microcontrolador MSP430F1611, este módulo permite que la mota pueda funcionar de tres modos de bajo consumo, para incrementar la vida de las baterías en aplicaciones donde es indispensable la movilidad y la autonomía.

Otra característica física interesante de comentar es que su Digitally Controlled Oscillator (DCO) permite regresar a modo activo desde los modos de bajo consumo en aproximadamente 6 µs (García Serrano, 2018).

Desde el punto de vista del consumo energético el dispositivo está diseñado con tres modos de funcionamiento y su consumo aproximado es:

- Modo Activo: 330 µA at 1 MHz, 2.2 V
- Modo Standby: 1.1 µA
- Modo Off (RAM activa): 0.2 µA

En la siguiente figura se aprecia el diagrama mecánico de las motas, donde se detallan las dimensiones y sus múltiples interfaces que hacen de este equipo especialmente interesante para el desarrollo de soluciones de integración y control.

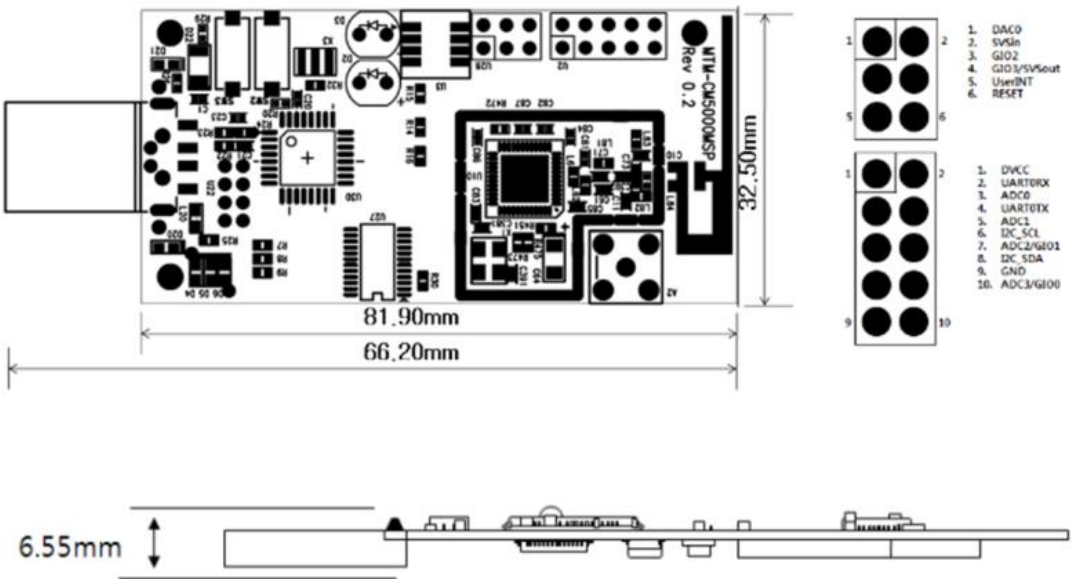

Figura 9 - Diagrama Mecánico CM5000

7.1. Características del Sistema de Transmisión

La Mota CM5000 está equipada con un chip CC2420 diseñado para aplicaciones en redes de bajo consumo, este chip trabaja a 2.4 GHz y cumple con IEEE 802.15.4.

Las características del modem de trasmisión permiten tener una velocidad efectiva de transmisión de 250 kbps y una ganancia de 9 dB.

Una importante característica que tiene el Chip CC2420 es su potencia de salida programable, esto es relevante dado que el mayor consumo de energía está en la transmisión de datos, entonces, al poder administrar las potencias de acuerdo los requerimientos permiten que podamos llegar a un correcto equilibrio entre el consumo y la potencia de transmisión. A continuación, una tabla que muestra el consumo v/s la potencia de salida (Advanticsys, 2015).

Output Power [dBm]	Current Consumption [mA]
0	17.4
-1	16.5
-3	15.2
-5	13.9
-7	12.5
-10	11.2
-15	9.9
-25	8.5

En condiciones de bajo consumo el chip se apaga cuando no está trasmitiendo, es posible mediante aplicaciones eliminar la administración de la potencia, pero requiere algunos cambios en registros y reconfiguración de la RAM.

7.2. Características Antena

Este dispositivo cuenta con dos opciones de antena, una interna empotrada en el módulo y una conexión SMA para conectar antenas externas, por defecto CM5000 opera con la antena interna. Si se requiere utilizar una conexión externa, por ejemplo, alguna antena especial, solo es necesario instalar (soldar) el conector SMA Hembra.

7.3. Características de los Sensores de Temperatura y Humedad

El equipo CM5000 está equipado con un chip Sensirion® SHT11 que proporciona el censado de temperatura ambiente y humedad, las características electrónicas y específicas de este sensor escapan al objetivo de este trabajo, para mayor detalle referirse ver la documentación especializada (Advanticsys, 2015)

El sensor almacena los valores leídos en 2 bytes y deben ser convertidos utilizando la siguientes formula:

Temperatura (Tc) = -39.60 + 0.01 * VT_Sensor

Humedad (%) = -4+0.0405 * VH_Sensor + (-2.8 * 10 -6) * (VH_Sensor ^{2})

VT_Sensor: Valor temperatura entregada por el sensor

VH_Sensor: Valor de la humedad entregada por el sensor

Nota: Estas fórmulas de conversión asumen una tensión estable de 3V. Cuando el voltaje es más bajo puede ser necesario corregir algunos valores, para mayor detalle consultar el manual del sensor.

7.4. Características sensor de Luz

El CM5000 tiene un sensor de luminosidad Hamamatsu® S1087 que corresponde en realidad a un luxómetro, que permite medir simple y rápidamente la iluminancia real y no subjetiva de un ambiente. La unidad de medida es el lux (lx).

El sensor consta de una célula fotoeléctrica que capta la luz y la convierte en impulsos eléctricos entregando valores que deben ser ajustados utilizando la siguiente formula:

Luz(lx) = 2.5 * (Lx_value) / 4096) * 6250

Nota: en el manual del S1087 (Photonics, 2014) se presenta un análisis detallado de las variaciones y factores que influyen en el desarrollo de esta fórmula (Advanticsys, 2015).

7.5. Características Interface USB

La interface USB es administrada por un chip FTDI® FT232BM que permite la comunicación, la configuración y el cambio de firmware de la mota conectada a cualquier dispositivo usb compatible, otra función que se facilita al contar con esta interface es funcionar como estación de recolección de datos provenientes de la red de sensores (Advanticsys, 2015).

Características básicas de la interface USB:

- Versión de USB 1.1 y 2.0
- EEPROM programable on-board via USB
- Alimentación de la Mota vía USB interface 5V y 3.3V

7.6. Características de Botones de Usuario

Dos botones disponibles proporcionan interacción con el Usuario/Programador, el primero de ellos (reset) permite regresar el dispositivo a un estado inicial sin perder la configuración, es decir, se inicia nuevamente el dispositivo, se borran todos los registros, pero se mantiene la configuración. El Segundo botón denominado user puede ser programado para desempeñar alguna función específica, como por ejemplo enviar a la red algún dato al momento de presionar el botón user (Advanticsys, 2015).

8. Sistemas Operativos para Redes de Sensores

Un sistema operativo, es una capa de software que tiene por objetivo proporcionar a los programas de usuario un modelo de computadora ideal, más simple y pulcro, así como encargarse de la administración de todos los recursos (Tanenbaum, 2009). Esta misma definición se aplica para los sistemas operativos de sensores.

En esta sección estudiaremos los diferentes sistemas operativos disponibles en el mercado con una mirada simple solo con el propósito de conocer alternativas.

En la actualidad contamos con una amplia y variada lista de sistemas operativos para redes de sensores, las diferencias principales radican en las arquitecturas de hardware sobre las cuales se pueden implantar.

A continuación, se muestra una recopilación y breve descripción de aquellos que cumplen con dos criterios: primero tienen un historial amplio de aplicaciones e implementaciones y segundo tienen un grupo de desarrollo activo, es decir, cuentan con una institución o grupo de personas que están actualmente trabajando en la manutención y actualización del sistema.

- Bertha: The Pushpin OS - Pushpin es una plataforma de Hardware y Software diseñada e implementada para modelar, testear y desplegar una red de sensores distribuida de muchos nodos idénticos (Joshua Lifton, 2002).

- Nut/OS: Es un pequeño sistema operativo de código abierto para aplicaciones en tiempo real, que trabaja con AVR, ARM, AVR32. (Proconx, 2008).

- Contiki: Es un Sistema Operativo de libre distribución que hoy en día tiene un gran desarrollo, puede ser instalado desde equipos de 8 bits a sistemas embebidos en microcontroladores (Adam Dunkels, 2004).

- CORMOS: Es un sistema operativo específico para redes de sensores orientado principalmente al desarrollo de aplicaciones de comunicación entre sensores (John Yannakopoulos)

- eCos: (embedded Configurable Operating System) es un sistema operativo gratuito, para trabajo en tiempo real, diseñado para aplicaciones y sistemas embebidos que sólo necesitan un proceso (Welton, 2004).

- MagnetOS: es un sistema operativo distribuido, cuyo objetivo es ejecutar aplicaciones de red que requieran bajo consumo de energía, adaptativas y fáciles de implementar (Computer Science Department - Cornell University, 2010).

- MANTIS: El sistema MANTIS promueve la flexibilidad multimodal y facilidad de uso a través de su apoyo a la detección multimodal incluyendo la ubicación con GPS (Hector Abrach, 2003)

- TinyOS: sistema operativo desarrollado por la universidad de Berkeley se caracteriza por estar escrito en lenguaje de programación nesC (P. Levis, 2001)

• t-Kernel: es un sistema operativo de tiempo real y de propósito específico para microcontroladores de 8 y 16 bit (Lin Gu, 2009).

• LiteOS: sistema operativo desarrollado en principio para calculadoras, en el ámbito de las redes de sensores destaca por que ofrece un ambiente de trabajo similar a un GNU/Linux (Masaaki Takahashi, 2009).

El Listado anterior no pretende ser un listado completo, sino que un listado parcial de los sistemas operativos más conocidos. además, no se contemplan Sistemas Operativos propietarios, estos están fuera del alcance de este documento, para efectos de este trabajo la plataforma seleccionada limita también la posibilidad de efectuar pruebas y validar las actividades que se desarrollan en este trabajo en otros sistemas operativos no compatibles.

En los próximos apartados de este trabajo de tesis se hacer un análisis de los sistemas operativos compatibles con la mota TelosB (Hardware) estudiada en las secciones anteriores. Es por esta razón que nos limitaremos al estudio de dos sistemas operativos, que pueden ser cargados en un dispositivo de estas características y gozan de una aparente popularidad frente a los demás Sistemas.

Los sistemas operativos que se estudian a lo largo de este trabajo son ContikyOS y TinyOS ambos están soportados por la plataforma TelosB.

8.1. ContikiOS

ContikiOS es un sistema operativo open source desarrollado bajo el alero de una tesis doctoral de Adam Dunkels, hoy en día es desarrollado por un grupo de trabajo conformado por personas de la industria y la academia siempre liderados por Adam, en sus inicios bajo el apoyo de Swedish Institute of Computer Science (SICS) y varias organizaciones empresariales como SAP, Cisco, Atmel, NewAE Technology y Universidad Técnica de Munich.

Este sistema operativo está disponible para la comunidad de forma gratuita bajo una licencia BSD, en sus inicios solo se ejecutaba en dispositivos de 8 bit, hoy en día ha sido implementado para varias plataformas de redes de sensores entre ellas TelosB. uno de sus principales objetivos es solucionar los problemas de interconexión de las redes de sensores con las redes tradicionales (internet). Es así como tiene integrado una pila de protocolos TCP/IP denominada uIP desarrollada también por Adam Dunkels como un proyecto separado y que en la actualidad forma parte del core de ContikiOS (Dunkels, 2012).

El lenguaje de programación utilizado para el desarrollo de ContikiOS es C, técnicamente soporta la programación orientada a eventos, está diseñado para sistemas embebidos con escasa memoria. Cuenta con un núcleo orientado a eventos sobre el cual los programas pueden ser cargados y descargados de forma dinámica en tiempo de ejecución. Los procesos en ContikiOS usan Protothreads (protohilos), un mecanismo de abstracción ideado para proporcionar un estilo de programación secuencial sobre el núcleo orientado a eventos (esto será tratado en detalle más adelante en este documento). También soporta multihilado apropiativo opcional por proceso. La comunicación entre procesos se realiza mediante la técnica de paso de mensajes, está implementada mediante el sistema de eventos del núcleo. Tiene un subsistema GUI

opcional, con soporte de gráficos para terminales locales, también soporta terminales virtuales en red mediante VNC o sobre Telnet, cuenta con un amplio rango de primitivas de comunicación. También soporta IPv6, junto con protocolos como RPL y 6LoWPAN.

ContikiOS tiene muchas posibles líneas de investigación y desarrollo, en este documento analizaremos el sistema operativo con un objetivo en mente: La Construcción de aplicaciones para el trabajo en red con los sensores.

8.1.1. Módulo de Comunicaciones – Internet en ContikiOS

El éxito de TCP/IP en el mundo de las comunicaciones lo ha convertido en el estándar global, y aquel dispositivo que no sea capaz de brindar soporte a esta plataforma queda inmediatamente fuera de internet. Es por ello que para el éxito de ContikiOS en las redes de sensores modernas es imperativo contar con una versión de TCP/IP que le permita participar de internet. En la actualidad ContikiOS provee soporte para IPv4 e IPv6, en IPv6 tiene implementado el stack uIPv6 con certificación IPv6 fase uno y se trabaja para escalarlo en la certificación. Para IPv4 posee dos stack: uIP y Rime. uIP cumple con las RFC de TCP/IP y hace posible la comunicación de ContikiOS en internet. Por otra parte, Rime, es un stack de comunicaciones ligero especialmente construido para comunicaciones de radio de baja potencia.

Uno de los problemas que se deben resolver al momento de las implementación de TCP/IP tiene que ver con el tamaño de las memoria que se requiere y el tamaño en disco, en general en redes de sensores son estos dos recursos muy escasos, entonces cualquier implementación debe resolver el problema haciendo optimizaciones que requieren un análisis profundo para elegir aquellas funcionalidades que se deben implementar, y cuáles se deben dejar fuera porque no son necesarias en los entornos comunes de redes de sensores.

La implementación de uIP en ContikiOS actualmente cuenta con un grupo imprescindible de protocolos del stack TCP/IP entre los que figuran: IP, ICMP, UDP.

En uIP también están implementados protocolos de bajo nivel como por ejemplo ARP que hace la traducción entre direcciones físicas y direcciones IP. También cuenta con protocolos de alto nivel como por ejemplo SMTP que es el encargado de la gestión de correo electrónico. En uIP los protocolos son divididos en dos grupos. Los de alto nivel denominados “la aplicación” como por ejemplo IP y TCP y los de bajo nivel que son implementados en su gran mayoría a nivel de hardware o firmware, son denominados “Dispositivo de red” y son controlados por el driver del hardware.

En el RFC 1122 que alberga los requerimientos de los host en Internet, se puede dividir en dos partes, la primera tiene relación con los requisitos que se deben cumplir en la comunicación entre dos host en la red y el segundo grupo contiene los requisitos que debe cumplir el stack de red para comunicarse con las aplicaciones dentro del dispositivo, el no cumplimiento de las primeras representa un problema que afectaría irremediablemente la comunicación con otro dispositivo que implemente TCP/IP, en cambio la violación de alguno de los requisitos del segundo grupo afectará solo la comunicación dentro del dispositivo, no afectando las comunicaciones entre dispositivos. Entonces este grupo de protocolos pueden ser prescindibles en sensores.

En uIP son implementados todos los requerimientos de los RFC que están relacionados con la comunicación de dispositivo a dispositivo, sin embargo, con el objetivo de reducir tamaño del código se eliminaron algunos mecanismos en la interface entre el stack y las aplicaciones, como por ejemplo los algoritmos que reportan errores o mecanismos que configuran dinámicamente opciones, dado que existen muy pocas aplicaciones que hacen uso de estos mecanismos pueden ser eliminados sin perder generalidad (Dunkels, A., Gronvall,B., Voigt T., 2004).

8.1.2. Estructura del Sistema de Archivos

A continuación, exploraremos y analizaremos las secciones más importantes dentro del código fuente de ContikiOS con el objetivo de comprender como está organizado internamente, donde están las herramientas y aplicaciones que facilitan el desarrollo de aplicaciones.

Al momento de descomprimir o descargar por algún medio el código de ContikiOS se obtienen una serie de herramientas, simuladores y documentación que ayudan a la compilación, ejecución y depuración de aplicaciones, estas se almacenan en siete carpetas que conforman la columna vertebral del sistema:

1. apps – Se almacenan alrededor de 40 aplicaciones que pueden ser seleccionadas para ser incluidas en la compilación y cargadas sobre el sistema operativo, dentro del listado se cuenta con aplicaciones como clientes y servidores web, servidor dhcp, aplicaciones para ftp, además herramientas como ping6, netconf entre otras.
2. core – En esta sección se almacena el código fuente de ContikiOS y todas las librerías necesarias para la compilación del Sistema, en el directorio core/net se encuentran las implementaciones de los protocolos de red que son interés del presente documento.
3. cpu – Se almacenan las implementaciones para los 15 tipos de chips de CPU sobre las cuales es posible correr ContikiOS, para este trabajo de tesis las plataformas de interés es la que se implementa dentro de cpu\msp430.
4. doc – Contiene fuentes para generar la documentación oficial
5. examples – Como su nombre lo indica aquí se almacenen los ejemplos que pueden ser compilados y cargados sin mayores configuraciones.
6. Platform – Contiene los controladores para los diferentes hardware sobre llas que se pude cargar Contiki, para este trabajo de tesis las motas TelosB tiene su software especializado en el directorio platform\sky.
7. Tools – Conjunto de herramientas útiles para en trabajo con redes de sensores como cooja (tools\cooja), simuladores de dispositivos telosB (tools\sky).
8. Regression-tests – En la versión 2.7 de ContikiOS se agregó un directorio que contiene 42 pruebas que se hacen en 9 emuladores con 4 diferentes arquitecturas, todo esto con el fin de comprobar compatibilidad hacia atrás con las aplicaciones nuevas.

8.1.3. Protothreads: Procesos en ContikiOS

En esta sección se efectuará un estudio de funcionamiento interno de ContikiOS, es decir, a nivel de procesos.

ContikiOS a nivel de procesos del sistema operativo trabaja con Protothreads. Los Protothreads son hilos muy ligeros diseñados para sistemas con muy poca memoria, como por ejemplo un sistema embebido o nodos de una red de sensores.

Los sensores gestionan la concurrencia con ejecución **basada en eventos** esto principalmente por la limitada cantidad de memoria y porque las tareas realizadas son muy dependientes de la E/S, esto trae como consecuencia que al momento de construir una aplicación se debe tener muy clara la máquina de estados, esto requiere mucho trabajo por parte del programador tanto a nivel de desarrollo, mantención y debug. Es en este punto donde los Protothreads juegan un papel importante como veremos a continuación.

Para ilustrar como protothreads reducen los problemas de la máquina de estados de la aplicación simplificando la codificación, se estudiará una comparativa de un pequeño mecanismo/protocolo simplificado que busca reducir el consumo de la batería de un dispositivo.

El mecanismo/protocolo tiene las siguientes reglas

1.- Encender equipo

2.- Esperar t_espera milisegundos

3.- Apagar equipo, pero sólo si todas las comunicaciones han terminado

4.- Si la comunicación no se ha completado, esperar hasta que se complete para apagar la radio

5.- Esperar t_dormido

6.- Ir a 1

Para implementar este protocolo es necesario bajo una ejecución basada en eventos primero identificar cuáles serán los estados para el diseño de la máquina, en este caso se identifican tres estados ENCENDIDO cuando el equipo está efectuando una comunicación, en ESPERA cuando el equipo está a la espera por alguna comunicación entrante o saliente y APAGADO cuando el equipo está en estado de ahorro de energía.

En términos gráficos la máquina de estado que describe el protocolo seria la siguiente:

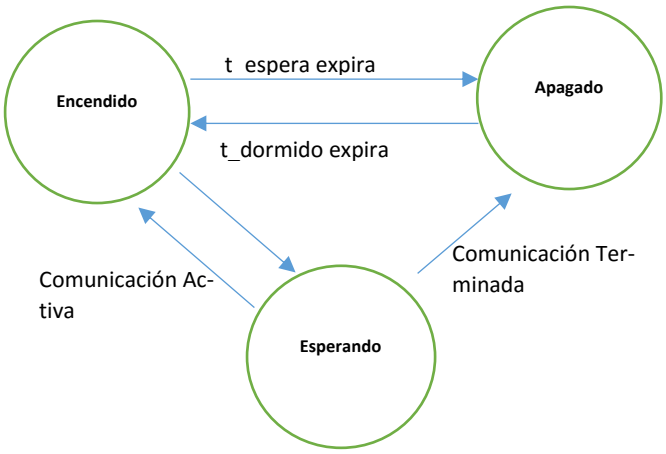

Diagrama 1: Máquina de Estado.

A continuación, se implementa en lenguaje C el protocolo orientado a eventos basado en la máquina de estados (Diagrama 1).

```
1  enum {encendida,esperando,apagada} antena;
2
3  void protocolo_de_radio() {
4      switch(antena) {
5          case apagada:
6              if(timer_expired(&timer)) {
7                  radio_on();
8                  antena = encendida;
9                  timer_set(&timer, t_espera);
10             }
11             break;
12         case encendida:
13             if(timer_expired(&timer)) {
14                 timer_set(&timer, t_dormido);
15                 if(!communication_complete()) {
16                     antena = esperando;
17                 } else {
18                     radio_off();
19                     antena = apagada;
20                 }
21             }
22             break;
23         case esperando:
24             if(communication_complete()
25                 || timer_expired(&timer)) {
26                 antena = encendida;
27                 timer_set(&timer, t_espera);
28             } else {
29                 radio_off();
30                 antena = apagada;
31             }
32             break;
33     }
34 }
```

Figura 10 - Implementación Protocolo máquina de estado

Por medio de la variable antena se maneja el paso entre un estado y otro, además se utiliza las siguientes funciones:

- timer_expired() para comprobar los tiempos que se permanece en cada uno de los estados.

- timer_set() que establece los contadores dependiendo del estado

- comunication_complete() que entrega el estado de todas la comunicaciones ya sean entrantes o salientes.

- radio_on() y radio_off() que encienden y apagan las comunicaciones en el dispositivo.

En este trozo de código, se puede apreciar que un mecanismo muy simple requiere de mucha planificación y codificación al momento de su implementación, resultando algo no tan claro en su utilización.

En el siguiente listado de código se implementará el mismo mecanismo/protocolo, pero utilizando la librería de protothreads.

```
1  PT_THREAD(protocolo_de_radio(struct pt *pt)) {
2      PT_BEGIN(pt);
3          while(1) {
4              radio_on();
5              timer_set(&timer, t_espera);
6              PT_WAIT_UNTIL(pt, timer_expired(&timer));
7              timer_set(&timer, t_dormido);
8              if(!communication_complete()) {
9                  PT_WAIT_UNTIL(pt, communication_complete()
10                                   || timer_expired(&timer));
11             }
12             if(!timer_expired(&timer)) {
13                 radio_off();
14                 PT_WAIT_UNTIL(pt, timer_expired(&timer));
15             }
16         }
17     PT_END(pt);
18 }
```

Figura 11 - Protocolo de radio implementado con protothreads

En el listado anterior [Figura 10] se muestra una implementación mucho más completa del mismo protocolo y con un flujo de control mucho más limpio y fácil de mantener.

Ahora analizaremos la implementación de la macro PT_WAIT_UNTIL, PT_BEGIN y PT_END de la librería de protothreads

8.1.4. Anatomía de las aplicaciones para ContikiOS

En esta sección estudiaremos como trabajan los protothreads a nivel de macros, es decir, como el procesador C expande las macros y como es ejecutado el código de las aplicaciones con el fin de comprender como se cambia el flujo de ejecución.

La operación de los protothreads se basa en la definición de un conjunto de macros (PT_WAIT_UNTIL, PT_BEGIN, PT_END etc) que son interpretadas por el preprocesador de C logrando cambiar o administrar el flujo de la ejecución de la aplicación. La definición de esa macro se presenta a continuación [Figura 11] (extraído de las librerías de ContikiOS).

```
1  struct pt { unsigned short lc; };
2  #define PT_THREAD(name_args)  char name_args
3  #define PT_BEGIN(pt)          switch(pt->lc) { case 0:
4  #define PT_WAIT_UNTIL(pt, c)  pt->lc = __LINE__; case __LINE__: \
5                                if(!(c)) return 0
6  #define PT_END(pt)            } pt->lc = 0; return 2
7  #define PT_INIT(pt)           pt->lc = 0
```

Figura 12 - Comparativa de macros vs código expandido

Estas siete definiciones de macros representan el núcleo de los protothreads y los analizaremos línea a línea.

Línea 1	Se define una estructura que contienes una variable lc (local continuation) que en la literatura y foros de desarrollo es muy comentada por su tamaño de dos bytes de overhead.
Línea 2	Se define la macro PT_THREAD que sólo agrega la palabra reservada “char” al argumento.
Línea 3	Se define PT_BEGIN que tiene el inicio del switch y el inicio del bloque Case, ambos se ejecutan la primera vez que es llamado el protothread.
Línea 4	Se especifica la macro principal que define el funcionamiento del protothread, primero establece el valor de lc (local continuation) al valor de la macro __LINE__ que corresponde al número de línea actual, de la misma forma define el inicio de un nuevo bloque de código **case** para cuando el valor del lc coincide con el número de línea, agrega también una segunda línea con la sentencia if que comprobara la condición de termino de ejecución del protothread
Línea 6	Sa macros PT_END cierra el switch que solo se ejecuta una vez que se cumple la condición
Línea 7	Se inicializa lc

Ahora veremos las macros detalladas en el párrafo anterior con un ejemplo en que mostraremos de forma paralela la implementación de un protothreads básico y lo extenderemos tal como lo hace el preprocesador de C [Figura 12].

```
#include "pt.h"
                                    | #include "pt.h"
                                    |
static int counter;                 | static int counter;
static struct pt example_pt;        | static struct pt example_pt;
                                    |
static                              | static
PT_THREAD(example(struct pt *pt))   | char example(struct pt *pt)
{                                   | {
  PT_BEGIN(pt);                     |   switch(pt->lc) { case 0:
                                    |
  while(1) {                        |   while(1) {
    PT_WAIT_UNTIL(pt,               |     pt->lc = 12; case 12:
      counter == 1000);             |     if(!(counter == 1000)) return 0;
    printf("FIN\n");                |     printf("FIN\n");
    counter = 0;                    |     counter = 0;
  }                                 |   }
                                    |
  PT_END(pt);                       |   } pt->lc = 0; return 2;
}                                   | }
                                    |
int main(void)                      | int main(void)
{                                   | {
  counter = 0;                      |     counter = 0;
  PT_INIT(&example_pt);             |   pt->lc = 0
                                    |
  while(1) {                        |   while(1) {
    example(&example_pt);           |     example(&example_pt);
    counter++;                      |     counter++;
  }                                 |   }
  return 0;                         |   return 0;
}                                   | }
```

Figura 13 - Código Protoheads v/s código extendido por el preprocesador

En resumen, podemos apreciar que PT_THREAD es traducido en una simple función que retorna un valor de tipo char manteniendo los argumentos sin cambio, el valor de retorno puede ser utilizado para determinar si el protothread está bloqueado, a la espera de que algo suceda o si el protothread ha salido o terminado.

Otro punto importante es el control del flujo, como es administrado por el **case** que cambia el flujo de la ejecución mientras el contador no complete los mil incrementos dentro del main(), entonces da la impresión que el código queda detenido dentro del PT_WAIT_UNTIL pero en realidad no es así, sino que es utilizado solo para comprobar el estado de alguna variable o de algún elemento.

Si proyectamos la utilización de PT_WAIT_UNTIL a las aplicaciones para sensores, son utilizados para comprobar el estado de los sensores.

```
PT_WAIT_UNTIL(pt, etimer_expired(&et));
```

Al concluir el estudio de la forma en que se construyen las aplicaciones en equipos especiales para redes sensores, observamos que la solución planteada por los protothread es muy limpia y fácil de implementar además permite una fácil manutención de las aplicaciones. El mayor costo o la dificultad está en que el programador debe entender el funcionamiento de los protothread de tal manera de saca provecho a estos equipos tan limitados en recursos.

Este estudio se basó en (Dunkels, SODA, the Software institutes' Online Digital Archive, s.f.) y (Dunkels, Dunkels, s.f.).

8.2. TinyOS

Otro de los sistemas operativos más utilizados para redes inalámbricas de sensores es TinyOS (Levis P. M., 2004) (Hill, 2000). Es un sistema basado en eventos bloqueantes y tareas no bloqueantes. Debido a que fue uno de los primeros sistemas operativos diseñados para redes de sensores tiene una gran difusión y popularidad. TinyOS está escrito en nesC, lenguaje basado en C y creado por investigadores de Universidad de California, Berkeley (Gay, 2003). TinyOS es compilado estáticamente, es por esto que las aplicaciones no son modificables una vez instaladas. Además, TinyOS no permite memoria dinámica, es decir, el tamaño definitivo de la memoria RAM queda definido al momento de compilación.

8.2.1. Documentación de TinyOS.

TinyOS cuenta con una documentación muy bien mantenida por la comunidad destacando principalmente los TinyOS Extension Proposal (TEP) (TEPS, 2013), estos documentos componen la columna vertebral de la comunidad que mantiene TinyOS, son presentados como documentos ordenados bajo una numeración que representa diferentes características.

TEP 1: Documento que describe como se estructuran todos los documentos.

TEP 2 a TEP 100: Documentos que describen las mejores prácticas.

TEP 100 en adelante: Documentos de experimentos y engloba la principal fuente de información del Sistema Operativo.

Otro documento que es muy importante es el manual de programación oficial que tiene la comunidad (Levis P. , 2006). En este documento se describen las principales características de TinyOs al momento de construir sus aplicaciones y será utilizado en este estudio.

8.2.2. Estructura Sistema de Archivos

La estructura de directorio básica al descargar TinyOS por cualquier medio es la siguiente:

1. Apps – Este directorio es la principal fuente de información para la construcción de aplicaciones porque cuenta con mucho ejemplo funcionales de aplicaciones y/o soluciones que se implementan con TinyOS. En el interior está la carpeta “test” que alberga aplicaciones que son utilizadas para testear algunas funcionalidades importantes que se requieren estén operativas para un correcto funcionamiento de Tinyos, por ejemplo, la aplicación TestsSerial que verifica la correcta conexión del dispositivo al equipo anfitrión o de desarrollo.
2. Doc – En este directorio se albergan los TEPs que fueron definidos en aparatados anteriores, los elementos a destacar dentro de este directorio es que dentro de la carpeta pdf se encuentras los documentos más extensos y detallado entre ellos el manual oficial de programación de TinyOS con las últimas modificaciones del sistema y detallas en extenso.
3. Support – Este directorio contiene software que no es utilizable en los nodos de la red, solo son aplicaciones relacionadas a TinyOS por ejemplo, están disponibles los SDK para los diferentes lenguajes y se declara explícitamente como el mejor SDK para TinyOS al que da soporte Lenguaje Java, los demás como C y Python no están completamente implementados.
4. Tools – Este directorio como su nombre lo indica contiene herramientas que son útiles en las diferentes plataformas a las que da soporte el sistema operativo como por ejemplo intelmote2, mica, msp430, sam3 y ucmote.
5. Tos – Directorio que contiene el código fuente de TinyOS. al efectuar un análisis del código fuente se puede apreciar que el sistema operativo esta principalmente desarrollado en c y nesC. Un subdirectorio muy importante que contiene una herramienta que ayuda mucho al momento de la depuración de aplicaciones es “printf” que permite imprimir pequeños mensajes desde las aplicaciones y desplegarlos en el puerto serial.

8.3. ContikiOS versus TinyOS

En este apartado se efectúa una comparativa de los dos Sistemas Operativos desde diferentes perspectivas, siempre teniendo como norte el desarrollo de aplicaciones para redes de Sensores, es decir, se estudiarán todas aquellas características que jueguen un papel importante a la hora de diseñar construir y probar aplicaciones en las redes de sensores.

8.3.1. Simuladores

Una de las herramientas que se utilizan con mayor frecuencia en el estudio de redes de sensores corresponde a los simuladores, en la actualidad tenemos varios simuladores que proporcionan una variada cantidad de funciones que permites a los investigadores desplegar sus soluciones en entornos controlados previo a su implementación real, en esta sección se expondrá la importancia y las limitaciones de los simuladores que nos ayudará a valorar el soporte que tenga un determinado sistema operativo para ser utilizado sobre un simulador.

En la mayoría de los casos, los nodos disponibles comercialmente tienen limitaciones tanto en energía como en capacidad de procesamiento, esto impone condiciones tanto sobre las características que debe tener el firmware instalado en el micro controlador para maximizar el tiempo de operación de la red de sensores, como sobre los protocolos de comunicaciones que minimizaran el número de transmisiones (Derpsch, 2012).

El desarrollo del firmware o aplicaciones para los sensores está sujeta a una serie de restricciones, pero dos son las que marcan el camino:

1.- Gestionar a los distintos componentes electrónicos del nodo con la precaución de mantener un mínimo consumo energético y en este sentido se habla de equipos que tiene un estado de bajo consumo en el cual pasan su mayor tiempo de operación. Si a esto se le agrega otro nivel de dificultad que se pone de manifiesto con la aparición de múltiples dispositivos que requieren que las aplicaciones soporten una infinidad de plataformas de Hardware.

2.- La segunda restricción tiene que ver con la estabilidad del firmware instalado. Debe comprobarse que el firmware a instalar en el nodo esté completamente libre de errores que puedan perjudicar el funcionamiento normal de la red. Pruebas exhaustivas del firmware deben realizarse con redes similares a las que se utilizarán en la realidad. Sin embargo, producir una red de prueba que permita depurar errores para posteriormente instalar el nuevo firmware en todos los nodos es impracticable, debido a su alto costo y a la imposibilidad de reproducir todos los escenarios bajo los que operará la red.

La solución a este problema típicamente se aborda mediante simulaciones. Los simuladores son programas computacionales que permiten modelar ciertas características de un sistema real, recreando su comportamiento en un tiempo y costo reducido. Los simuladores además permiten generar y recrear condiciones que pueden ser difíciles de obtener en la realidad, como condiciones climatológicas específicas o redes con un número de nodos de alto costo total. Una simulación de una red completa permite comprobar que los protocolos, algoritmos y funciones que se implementarán en el firmware trabajan correctamente, a un costo aceptable. Los simuladores existentes no son aptos para estos propósitos. Un primer grupo utiliza el mismo código escrito para el firmware, permitiendo una simulación del funcionamiento del nodo consistente con la operación real, pero requieren de extensiones que simulen el ambiente de operación de los nodos en forma fidedigna (canal inalámbrico, entorno físico y variables observadas). Otro grupo son simuladores que representan de manera más precisa el canal de radiofrecuencia, pero están basados en un lenguaje diferente al utilizado por el firmware. Esto implica que toda la funcionalidad simulada debe ser re-implementada en firmware para su uso en terreno.

TinyOS incorpora como parte de su distribución una extensión llamada TOSSIM que permite simular aplicaciones completas a nivel de protocolos de comunicaciones. Existe también una extensión a este, llamado PowerTOSSIM, que simula además los consumos energéticos de los algoritmos de la aplicación. Si bien TOSSIM permite realizar simulaciones de las aplicaciones de TinyOS, está desarrollado en lenguaje Python, por lo que su uso junto a TinyOS requiere interoperabilidad de código escrito en dos lenguajes distintos. Finalmente, TOSSIM no soporta la simulación de tecnologías de múltiples antenas.

Por su parte ContikiOS incluye en su distribución un simulador llamado Cooja, basado en Java. Este simulador utiliza probabilidades de error entre enlaces para simular canales de variadas estadísticas, y está diseñado en forma modular; permitiendo incorporar nuevos modelos de canal en las simulaciones. Si bien Cooja permite realizar simulaciones de las aplicaciones de ContikiOS, está desarrollado en un lenguaje distinto que las aplicaciones y el sistema operativo. Al igual que TOSSIM, Cooja no soporta simulaciones de tecnologías de múltiples antenas.

Para redes de sensores también existen simuladores stand-alone. Uno de ellos es el simulador Worldsens, compuesto por un simulador de nodos inalámbricos llamado WSim, y un simulador para la red denominado WSNet, es un simulador de canales inalámbricos y de entorno. Ambos se conectan mediante sockets TCP/IP. WSim permite emular el comportamiento exacto a nivel de ciclos de la operación de un microcontrolador MSP430 esto posibilita utilizar para la simulación el mismo firmware que se utiliza para los nodos. WSim y WSNet son útiles cuando se requieren simulaciones precisas para la evaluación de algoritmos en el microcontrolador. Al igual que con TOSSIM y Cooja, WSNet tampoco soporta simulaciones con tecnologías de múltiples antenas.

Este apartado está basado principalmente en la recolección de información proporcionada en la tesis de magister de Sergio Campama Derpsch (Derpsch, 2012).

8.3.2. Análisis de las Licencias de los Sistemas Operativos

En este apartado estudiaremos las licencias con las que se distribuyen los dos sistemas operativos en estudio.

Una licencia es un contrato entre el creador de una aplicación y el usuario que lo adquiere. Todo software tiene una licencia, aunque sea llamado de dominio público, no importa si el software es comprado en un soporte físico o es descargado de internet. La licencia específica que podemos hacer con el programa.

Las licencias al ser unilaterales, es decir, son presentadas al usuario como una serie de condiciones para utilizar el software, requieren de un estudio antes de seleccionar una aplicación, en este caso un sistema operativo para ser utilizado en un proyecto, porque los derechos que tiene el usuario en relación al programa pueden variar según el tipo de licencia que se emplee. Hay algunas licencias muy restrictivas y otras que dan mayor libertad de uso.

Licencia ContikiOS

En primer lugar, analizaremos ContikiOS que publica su licencia en su sitio web[3] en ella se especifica que el código fuente es distribuido bajo la licencia BSD. Con esta licencia ContikiOS puede ser utilizado de manera libre en proyectos comerciales y no comerciales solo se requiere mantener los encabezados de los archivos fuentes con el copyright.

El derecho de autor copyright, solo vela por la autoría de los archivos fuentes que forman parte de ContikiOS.

La licencia open source de ContikiOS no exige que el código desarrollado sea compartido o distribuido con otros. Si se utiliza el código de ContikiOS, las aplicaciones (código fuente) que se generen no es obligación compartirlas.

En el caso que se quiera colaborar con el proyecto y contribuir con código fuente, este código debe ser compartido para que sea cubierto por la misma licencia que tiene el resto del código fuente de ContikiOS.

La licencia de ContikiOS es la siguiente [Figura 13].

```
Copyright (c) 20**, X.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
3. The name of the author may not be used to endorse or promote
   products derived from this software without specific prior
   written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS
OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

Figura 14 - Licencia BSD del código fuente de ContikiOS

Esta licencia es denominada BSD modificada (de 3 cláusulas) porque se eliminó una cláusula denominada cláusula de publicidad [Figura 14] que hacía referencia a la BSD original incompatible con la GPL.

```
3. All advertising materials mentioning features or use of this software
   must display the following acknowledgement:
   This product includes software developed by the University of
   California, Berkeley and its contributors.
```

Figura 15 - Cláusula de publicidad eliminada.

El problema de esta cláusula radicaba en que cada contribuidor reemplazaba en los créditos a la "Universidad de California, Berkeley" por su propio nombre o institución, esto trajo

[3] http://www.contiki-os.org/license.html

como consecuencia la existencia de una lista cada vez más larga de diferentes instituciones que se estaba obligando a mostrar cuando se distribuían varios programas juntos.

A la licencia resultante se le conoce como la "nueva licencia BSD" (como la denomina OSI) aunque también se le conoce como la "BSD revisada", "BSD modificada", "BSD-3" o "BSD de 3 cláusulas" y es la licencia que engloba al código fuente de ContikiOS.

La Fundación de Software Libre (FSF) tiene su propia apreciación sobre esta licencia y la publica en su sitio web como una licencia libre compatible con una Licencia de documentación libre de GNU, pero hace la siguiente observación (Free Software Foundation, 2018):

> *Esta es la licencia BSD original, modificada por la eliminación de la cláusula de publicidad. Es una licencia de software libre laxa, permisiva, sin copyleft, compatible con la GPL de GNU.*
>
> *A esta licencia se la llama algunas veces «licencia BSD de 3 cláusulas».*
>
> *Como licencia laxa, permisiva, la BSD modificada no es tan mala, aunque es preferible la Apache 2.0. Sin embargo, es peligroso recomendar el uso de la «licencia BSD», incluso en casos especiales como por ejemplo para programas pequeños, porque fácilmente se podría producir una confusión que llevaría al uso de la defectuosa licencia BSD* original*. Para evitar ese riesgo, se puede sugerir el uso de la licencia X11. Las licencias X11 y BSD modificada son más o menos equivalentes.*
>
> *De todas maneras, para programas de una cierta magnitud es mejor la licencia Apache 2.0 porque previene la trampa de las patentes.*

Licencia TinyOS

Tinyos en sus primeras versiones estaba bajo licencia Open Source de Intel. Pero en su última versión este sistema operativo tiene una licencia del tipo BSD y en su sitio web declara lo siguiente.

> *All code in the main TinyOS tree is licensed under a New BSD license. Unlike the GPL, a New BSD license does not require you redistribute source code. It does have some restrictions, however, such as including copyright notices in documentation and not using the names of the developers to promote or endorse products.*

La fecha de publicación es 4/23/2011 (commit 5543, main SVN T2 repo). En este extracto se aclara porque se opta por la licencia que se encuentra en el repositorio oficial en la siguiente ruta tinyos-main/licenses/bsd.txt.

Licencia BSD

Como los dos sistemas operativos se basan en la misma licencia realizaremos un análisis más exhaustivo de la licencia recopilando diferentes posturas referentes a la libertad de la licencia BSD.

La discusión sobre esta licencia se centra en que BSD es una licencia que permite cerrar una posible mejora (derivados) de algunos de estos Sistemas Operativos, Aun así, esto genera que se pueda mal entender y a menudo se cree que esta capacidad es total, es decir, que cualquiera que haga una modificación por muy mínima que esta sea puede cerrar el código en su totalidad, y por tanto, un usuario que tuviera alguna aplicación que utiliza este código se vería afectado porque le han privado de la libertad de software que tenía hasta el momento en que fue cerrado el código. Esto está muy lejos de ser real y expondremos un excelente punto de vista que se muestra a continuación.

Se expondrá el funcionamiento de la licencia BSD, desde un punto de vista que promueve un software más libre.

Existen al menos 3 tipos de licencia BSD: la original, la modificada de 3 cláusulas y la simplificada de 2 cláusulas.

La licencia original contiene una cláusula que es muy poco útil y que incluso en cierto caso, sería hasta injusta. Esa cláusula ponía como condición que todo software derivado (del tipo que fuera) debería constar que se construyó gracias a software de la Universidad de Berkeley y sus colaboradores. A día de hoy, esto es algo que se aleja bastante de la realidad, y además era una cláusula que no compatibiliza con GPL, entonces la que se usa normalmente debido a su modernidad es la modificada de 3 cláusulas que se muestra a continuación [Figura 14]:

```
/*
 * Copyright (c) 2013 <your name here>
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * - Redistributions of source code must retain the above copyright
 *   notice, this list of conditions and the following disclaimer.
 *
 * - Redistributions in binary form must reproduce the above copyright
 *   notice, this list of conditions and the following disclaimer in the
 *   documentation and/or other materials provided with the
 *   distribution.
 *
 * - Neither the name of the copyright holders nor the names of
 *   its contributors may be used to endorse or promote products derived
 *   from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL
 * THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
 * INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
 * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 */

/*
 * Code description
 *
 * Author: Name here <email address>
 */
```

Figura 16 - Licencia de TinyOS

La parte final, en letras mayúsculas [Figura 15], nos describe la ausencia de garantía del software, con gran similitud a lo que nos podemos encontrar en las GPL.

La sección principal que nos interesa son los 3 puntos. Tras dejar claro el copyright de autoría original (al comienzo de todo), la licencia también clarifica que se puede cerrar el código, esto es verdad, pero no del software en sí mismo (el original), sino una copia/modificación de éste. Esto es importante diferenciarlo, ya que en muchas ocasiones se tiende a creer que se puede cerrar todo el software cuando no es así. Además, el cambio de licencia no afecta a versiones anteriores a dicho cambio y, por tanto, toda versión anterior a la nueva licencia permanece libre.

El autor original del software (el que licenció con BSD) tiene garantizado el derecho de autoría original, y así lo hace constar al comienzo del texto de la licencia. Por supuesto, mantiene toda validez legal en caso de que quiera reclamar autoría por cualquier causa.

Ahora veamos detalladamente las cláusulas:

En la primera cláusula nos dice que, si el producto derivado se distribuye con código fuente, el copyright (y por consiguiente el nombre del autor original) así como las condiciones de la licencia BSD que afectan a las partes originales y la consiguiente renuncia del copyright a dicha parte original, deben ser expuestos en primer lugar, en términos simples: antes que otra cosa se nombra al autor original, y el desarrollador (o empresa) que haya derivado software (libre o no) a partir de algo BSD tiene que renunciar a la parte de posesión intelectual (copyright) de los componentes originales en las que se basó (las licenciadas BSD, y por tanto libres).

La segunda cláusula indica que, si la distribución de obras derivadas se hace en formato binario, el programa debe reproducir todo lo descrito anteriormente, bien en el mismo programa, o bien en la documentación que se incluya en él (o en ambos).

La tercera cláusula protege moralmente al autor original de las derivaciones. Su nombre (y de los contribuidores) no puede ser nombrado en ningún caso para promoción, márketing, etc., de cualquier obra derivada sin su consentimiento. Esto, por ejemplo, exime de responsabilidad moral al autor original en caso de que una copia de un software BSD se torne a licencia no libre. Por tanto, permite que no se relacione al autor original con derivaciones que no tengan nada que ver con él, y de modo contrario, los autores de la obra derivada deberán ponerse en contacto con él y haciendo la petición de permiso por escrito.

La violación de alguna de estas condiciones recae en una ilegalidad, y por tanto se puede reclamar en todo momento con total validez.

Los derechos del usuario y del desarrollador quedan totalmente cubiertos. En ningún caso, la licencia BSD da opción a esclavizar, ni deja de tener en cuenta al usuario. GPL plantea la libertad de modo colectivo (todo lo licenciado GPL es propiedad moral de ese software en sí mismo, como si de un ente se tratara), mientras que BSD plantea un modo individual (el autor tiene que ser nombrado, pero puedes hacer con tu copia lo que quieras). Esto permite mucha flexibilidad tanto en el uso simple como en un futuro desarrollo, y no sólo da poder a los desarrolladores y programadores, sino también al usuario en sí mismo

Como se mencionó en los inicios de este apartado todo software tiene su licencia asociada y no siempre se le da la importancia que requiere, para este trabajo el tema de la licencia

tiene relevancia porque se busca que los avances realizados queden disponibles a la comunidad para ser mejorados y/o reutilizados en otros proyectos. En este sentido las licencias que tiene los dos sistemas operativos son compatibles con las proyecciones que se le busca dar a este trabajo.

8.3.3. Comunidad de Desarrollo

Un punto importante para el éxito de una Aplicación es el soporte que esta tenga en el tiempo. Una aplicación que tenga historia y casos de éxito genera confianza al momento de utilizarla en implementación de soluciones, es por este motivo que se estudiaran los sistemas operativos en cuanto a su prestigio y que tan sólida es su comunidad de desarrollo.

ContikiOS

ContikiOS tiene una comunidad muy activa de programadores que mantiene el sistema operativo tanto así que han mantenido un ciclo de actualizaciones que permiten hoy día contar con la versión 3 del sistema.

Contiki 3.0 (25 August 2015) Ultima versión disponible al 2018
Contiki 2.7 (15 November 2013)
Contiki 2.6 (17 July 2012)
Contiki 2.5 (9 September 2011)
Contiki 2.4 (16 February 2010)
Contiki 2.3 (27 June 2009)
Contiki 2.2.3 (24 March 2009)
Contiki 2.2.2 (17 November 2008)
Contiki 2.2.1 (6 September 2008)
Contiki 2.1 (3 December 2007)

La comunidad se encuentra bastante organizada tras su sitio web oficial presentando una lista de correo que agrupa a todos los desarrolladores que participan del proyecto y esta accesible a cualquiera que dese participar.

https://lists.sourceforge.net/lists/listinfo/contiki-developers

Complementando a esto el sitio cuenta con el historial del movimiento de la lista y se puede apreciar un continuo trabajo de la comunidad en el mejoramiento del sistema operativo.

contiki-developers — List for Contiki developers

You can subscribe to this list here.

2010	Jan (179)	Feb (356)	Mar (266)	Apr (275)	May (367)	Jun (236)	Jul (234)	Aug (148)	Sep (142)	Oct (167)	Nov (316)	Dec (237)
2011	Jan (247)	Feb (344)	Mar (408)	Apr (352)	May (458)	Jun (364)	Jul (372)	Aug (474)	Sep (429)	Oct (430)	Nov (428)	Dec (321)
2012	Jan (397)	Feb (492)	Mar (704)	Apr (488)	May (483)	Jun (487)	Jul (536)	Aug (388)	Sep (271)	Oct (263)	Nov (346)	Dec (363)
2013	Jan (446)	Feb (484)	Mar (634)	Apr (422)	May (395)	Jun (313)	Jul (286)	Aug (302)	Sep (306)	Oct (406)	Nov (388)	Dec (243)
2014	Jan (294)	Feb (278)	Mar (288)	Apr (331)	May (363)	Jun (242)	Jul (218)	Aug (256)	Sep (283)	Oct (296)	Nov (244)	Dec (150)
2015	Jan (132)	Feb (187)	Mar (200)	Apr (237)	May (218)	Jun (200)	Jul (185)	Aug (168)	Sep (149)	Oct (180)	Nov (144)	Dec (154)
2016	Jan (124)	Feb (108)	Mar (181)	Apr (165)	May (183)	Jun (165)	Jul (93)	Aug (102)	Sep (96)	Oct (95)	Nov (92)	Dec (98)
2017	Jan (92)	Feb (112)	Mar (144)	Apr (197)	May (161)	Jun (85)	Jul (87)	Aug (123)	Sep (93)	Oct (71)	Nov (64)	Dec (38)
2018	Jan (47)	Feb (42)	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec

Figura 17 - Movimiento de la Lista de Correo

El sitio de la comunidad es bastante bien mantenido y se aprecia una preocupación por mantenerlo ordenado.

El árbol de desarrollo es mantenido en el servicio Github

https://github.com/contiki-os/contiki.

En la siguiente [Figura 17] se aprecia el movimiento (contribuciones al código de ContikiOS) que ha tenido el árbol de desarrollo.

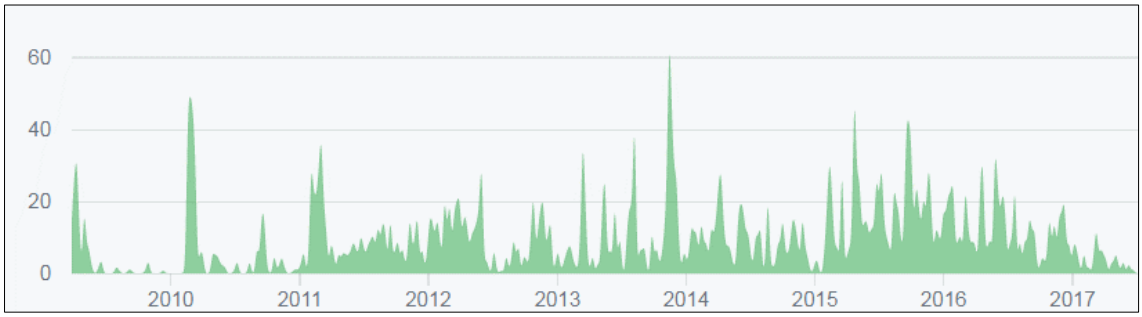

Figura 18 - Movimiento del árbol de desarrollo en GitHub

En octubre de 2017 los desarrolladores que contribuyen regularmente al sistema son once, liderados por Adam Dunkels, Oliver Schmidt y Fredrik Österlind.

TinyOS

Al igual que ContikiOS este sistema operativo cuenta con una comunidad que aporta a la evolución del sistema operativo y se encuentra agrupado en tres listas de interés cada una con un propósito distinto.

- tinyos-help: Es una lista que funciona como un foro donde se pueden plantear preguntas o inquietudes respecto al uso del sistema operativo.

https://www.millennium.berkeley.edu/mailman/listinfo/tinyos-help

- tinyos-devel: Es una lista más especializada en que se resuelven problemas referentes al desarrollo del sistema, en ella no se resuelven dudas, sino que está más bien para la coordinación del grupo principal.

https://www.millennium.berkeley.edu/mailman/listinfo/tinyos-devel

- tinyos-2-commits: Es una lista de sólo lectura que tiene como propósito servir de historial a los aportes de los desarrolladores al árbol del código fuente.

https://www.millennium.berkeley.edu/mailman/listinfo/tinyos-2-commits

Además de las listas de interés la comunidad está organizada en grupos que permiten focalizar los esfuerzos en tareas específicas de tal manera que sea fácil identificar los grupos en los cuales se pude participar de acuerdo a los intereses y experiencias.

Los Grupos actualmente activo son:

TinyOS Core Working Group (core)
TinyOS Network Protocol Working Group (net2)
TinyOS Tools Working Group (tools)
TinyOS IEEE 802.15.4 Working Group (15.4)
TinyOS ZigBee Working Group (zigbee)

Cada grupo tiene a su cargo un directorio del árbol de desarrollo de tinyOS que contiene el código y la documentación del módulo que se está desarrollando manteniendo y documentando.

En el sitio web de la comunidad se publica también un protocolo para la creación de nuevos grupos de trabajo, estos son aprobados por un comité con el objetivo de mantener una estructura y un orden en los esfuerzos.

TinyOS tiene además una sección de aprendizaje donde están muy bien documentada la forma de iniciarse en el mundo de TinyOS, esta documentación está montada sobre una wiki que permite el crecimiento colaborativo de la base de conocimiento.

8.3.4. Plataformas

En este apartado se realizará un estudio documental de los dispositivos o el hardware que soporta cada uno de los sistemas operativos, con el fin de tener una clara idea del alcance que tiene cada uno de los sistemas.

Los criterios para la confección de la lista serán los siguientes, en primer lugar, que sean declarados como soportados por la comunidad que desarrolla, en segundo lugar, tenga una comunidad de desarrollo que la respalde y por último presente una aplicación clara donde se pueda utilizar el dispositivo (p.e. se descartan dispositivos que son clones de otro dispositivo que también tienen soporte para los sistemas operativos)

Plataformas soportadas por TinyOS

EPIC Plataform	Epic es la versión moderna de CIRCA una plataforma para el desarrollo de sistema empotrados de propósito general. Tiene componentes modulares que permiten agregarle funcionalidades según requerimientos de la solución que se desee implementar. http://www.eecs.berkeley.edu/~prabal/projects/epic/
IntelMote2 (iMote2)	Es un nodo sensor de característica avanzada desarrollado por Intel, tiene soporte para IEEE 802.15.4. tiene un diseño modular y escalable, con una interface de conexión para tarjetas de expansión, también proporciona un set de interfaces de conexión de I/O. http://wsn.cse.wustl.edu/images/e/e3/Imote2_Datasheet.pdf
Shimmer	Dispositivo sensor para aplicaciones en prendas o vestimenta deportiva Inteligente, provee extensiones para sensores fisiológicos y de movimiento cinético, la característica más destacable esta su gran sistema de almacenamiento. http://www.shimmersensing.com/research-and-education/applications/#applications-tab
NXTMOTE	Se denomina de esta forma a la combinación de las plataformas NTX de lego mindstorm, de esta forma de desarrollan aplicaciones en neC para utilizar todos los accesorios con los que cuenta la plataforma lego, teniendo con ello una infinidad de posibles aplicaciones de investigación. http://nxtmote.sourceforge.net/

Tmote sky	Es una plataforma de ultra bajo consumo energético, simple de utilizar tiene varios tipos de interfaces que colaboran en hacer de este componente una muy buena plataforma para estudios y pruebas. Cuenta con interfaces USB y sus interfaces de radios son compatibles con IEEE 802.15.4. http://www.eecs.harvard.edu/~konrad/projects/shimmer/references/tmote-sky-datasheet.pdf
tinynode	Dispositivo para sistemas de detección de vehículos en estacionamientos, de fácil instalación y despliegue. http://www.tinynode.com
UCMote Mini	Dispositivo que busca integrar todos aquellos elementos propios de una mota en el menos tamaño posible, es un dispositivo que en su versión más completa cuenta con una batería, sensor de luz, acelerómetro 3D, sensor de humedad, temperatura y presión atmosférica. http://www.ucmote.com/images/uploaded/File/ProgrammingUCMotes.pdf

Plataformas Soportadas por Contiki

RE-Mote	Tarjeta de desarrollo para desarrollo de proyectos relacionados a la internet de las cosas. Además de tener soporte para ContikiOS también cuenta con soporte para Arduino y Rapsberry PI, principalmente orientado a la educación, ciudades inteligentes, seguridad y robótica http://zolertia.io/product/hardware/re-mote
CC2538 Development Kit	Es un conjunto de dispositivos compatibles con ContikiOS que proporcionan un ambiente de desarrollo y test. Entre sus características principales destacan compatibilidad con IEEE 802.15.4, Entre los dispositivos que contiene el kit destacan un dongle USB CC2531 para captura de paquetes en la red, una placa CC2538 equipada con un procesador ARM Cortex M3. http://www.ti.com/tool/cc2538dk
Wismote	Es un completo dispositivo sensor/actuador especialmente diseñado para redes de sensores, soporta 6LoWPAN además IEEE 802.15.4, una característica que destaca es su compatibilidad con dispositivos PLC y además de comunicación RS-485, estas características los hacen especial para aplicaciones de domótica y de control industrial. http://wismote.org/doku.php?id=specification

AVR Raven	Dispositivo para pruebas de aplicaciones, cuenta con transeiver de 2.4GHz, Un procesador PicoPower ABR y una pantalla LCD que lo hacen ideal para desarrollo de aplicaciones embebidas. http://www.atmel.com/tools/avrraven.aspx
MICAz	Es un dispositivo sensor especialmente pensado para operar como estación recolectora en redes de sensores, cuenta con una interface Serial/USB que permiten conectarse a un PC tradicional y funcionar como Gateway, además cuenta con un MIB600 que le permite conectarse a redes basada TCP/IP con ethernet. http://www.memsic.com/userfiles/files/Datasheets/WSN/micaz_datasheet-t.pdf
Tmote Sky	Es un dispositivo muy popular en el tema de redes de sensores, el dispositivo cuanta con muchas opciones de configuración debido a que está equipado con sensores, interfaces de interconexión USB, líneas de expansión para instalarle antenas externas y múltiples opciones de alimentación. Además, su bajo costo hace de este dispositivo muy atractivo para iniciar proyectos en internet de las cosas. http://www.eecs.harvard.edu/~konrad/projects/shimmer/references/tmote-sky-datasheet.pdf
MSB430 Sensor Mote	Es un dispositivo especialmente diseñado para propósitos académicos, la principal característica es que tiene un microcontrolador MSP430 de Texas Instruments, memoria de 60KB divididos en 5KB de RAM y 55KB de ROM y por último un slot de lectura de tarjetas SD-/MM con un soporte de hasta 4GB de almacenamiento. http://www.ti.com/lsds/ti/microcontrollers_16-bit_32-bit/msp/ultra-low_power/msp430f1x/overview.page
SEED-EYE BOARD	Es un nodo sensor multimedia, la principal diferencia con el resto de los dispositivos es que cuenta con cámaras que permiten tomar y enviar imágenes 160x120 16bit. Además, cuenta con una gran cantidad de interfaces de comunicación IEEE802.15.4 / ZigBee, y USB. http://rtn.sssup.it/index.php/hardware/seed-eye
CC2530 Development Kit	Kit de Desarrollo para aplicaciones y soluciones compatibles con 802.15.4, las características que destacan en estos dispositivos son sus placas con pantallas LCD 3x16 Caracteres, otra característica técnica destacable es su compatibilidad con el stack ZigBee http://www.ti.com/tool/cc2530dk

En esta recopilación de dispositivos compatibles con cada sistema operativo, quedan fuera todos aquellos que no tienen una organización o equipo de trabajo que esté dando soporte o esté trabajando en el desarrollo del dispositivo, es el caso (uno de varios) de apple2enh es un proyecto que fue congelado el GitHub por no actividad en el árbol de desarrollo.

Dada las características de este trabajo en que se requiere una plataforma representativa y principalmente un dispositivo que sea los más genérico posible, es decir, entregue soporte a los dos sistemas operativos y sea fácil de encontrar en el mercado y/o construir de manera independiente, en ese sentido el mejor dispositivo que cumple estos requerimientos es el dispositivo TelosB , tienen la particularidad de ser Open Hardware esto permite tener acceso a sus especificaciones ya sea de forma libre o comercial, si bien no tiene un estado de madures como por ejemplo la plataforma Arduino, se presenta como una buena alternativa para hacer disponibles a la comunidad los avances en materia de hardware.

Hoy en día, el término “hágalo usted mismo” (DIY por sus siglas en inglés) se está popularizando, en el hardware gracias a proyectos como Arduino que es una fuente abierta de prototipos electrónicos, una plataforma basada en hardware flexible, fácil de utilizar y en este sentido que la plataforma TelosB se proyecta como el caballo de batalla del Internet de las Cosas, permitiendo hacer que las ideas prosperen sin tener que adaptarse al hardware sino que todo lo contrario, el hardware está a su disposición para ser modificado y adaptado.

9. ContikiOS en Plataforma TelosB

En esta sección se efectuarán pruebas del sistema operativo ContikiOS sobre una plataforma compatible con TelosB que permitirá analizar el comportamiento de alguna característica básica del Sistema Operativo.

Para efectuar las pruebas se establece la siguiente configuración de Hardware:

Equipo anfitrión

-Equipo IBM Lenovo X200 - https://support.lenovo.com/cl/es/documents/migr-73156

Nodo Sensor o Mota

-CM5000 TelosB - http://www.advanticsys.com/shop/mtmcm5000msp-p-14.html

En primer lugar, se realizará la instalación del sistema operativo en el dispositivo telosB, en segundo lugar, se realizarán pruebas de algunos ejemplos de redes de sensores básicos.

9.1. Instalación

Para realizar la instalación se descargar la última versión del sistema operativo disponible en sitio web de ContikiOS:

http://www.contiki-os.org/download.html

Conectar la mota al USB de las maquina e identificar si es reconocido por el sistema operativo anfitrión, -el sistema operativo utilizado (distribución) en estas pruebas es UBUNTU 14.04 LTS.

Para comprobar si la mota fue identificada por el sistema operativo el comando `lsusb` (lista de dispositivos conectados) entrega el resultado que se muestra a continuación

```
Bus 002 Device 010: ID 0403:6001 Future Technology Devices International, Ltd
FT232 USB-Serial (UART) IC
```

La información que entrega es referente al ID del vendedor y el ID del producto

```
idVendor:          0x0403 Future TechnologyDevicesInternational, Ltd
idProduct:         0x6001 FT232 USB-Serial (UART) IC
```

Una vez identificado hardware, se requiere cargar el sistema operativo ContikiOS al dispositivo sensor, para estas tareas Ubuntu no cuenta por defecto con herramientas que le permitan interactuar con el dispositivo, para solucionar este problema el equipo de desarrollo de ContikiOS ofrece un set de herramientas que ayudan a esta tarea.

9.2. Administración de dispositivos sensores desde Ubuntu

9.2.1. Compilación de ContikiOS

Para compilar el sistema operativo ContikiOS y posteriormente cargarlo a la mota se requiere un conjunto de aplicaciones instaladas en el equipo anfitrión.

`binutils-msp430`	aplicaciones para la arquitectura MSP430, familia de la mota CM5000
`gcc-msp430`	gcc para MSP430
`msp430-libc`	Librería de gcc para compilación

Estos paquetes se instalan con el siguiente comando:

```
sudo apt-get install binutils-msp430 gcc-msp430 msp430-libc
```

Instalados estos paquetes, el equipo anfitrión está en condiciones de compilar el Sistema Operativo que será instalado en la mota.

Todo el Sistema Operativo ContikiOS, realiza la gestión de dependencias de su código fuente con la herramienta GNU/Make, para trabajar en el desarrollo de aplicaciones para ContikiOS es esencial un conocimiento medio/avanzado de la herramienta GNU/make.

A continuación, se describe la forma de compilación de una aplicación básica, sólo con fines de ejemplificación.

9.2.2. GNU/Make para compilación de ContikiOS

Para el Desarrollo de ContikiOS se utiliza la herramienta de compilación automatizada Gnu-make (Free Software Foundation, 2014) que se encarga de la gestión de compilación de los códigos fuentes. A continuación, detallaremos elementos relevantes que se requieren conocer para una correcta configuración del entorno de desarrollo.

En términos generales el sistema de compilación está compuesto por un conjunto de Makefiles:

```
Makefile.include
```

Es el makefile principal del sistema, define el lugar donde se almacenan los códigos fuentes en el sistema, está ubicado en la raíz del árbol de directorios del sistema operativo ContikiOS.

```
Makefile.$(TARGET)
```

En este makefile se almacenan las reglas de compilación según la plataforma para la cual se quiera compilar. La variable $(TARGET) hace referencia a el nombre de la plataforma para la que se aplican el Makefile. Este archivo se almacena en el sub-directorio correspondiente a cada plataforma en el directorio /plataform.

```
Makefile.$(APP)
```

Cada aplicación contiene su propio archivo makefile en este archivo se detallan las reglas de compilación propios de la aplicación (donde "$(APP)" es el nombre de la aplicación dentro del directorio apps).

9.3. Compilación de Aplicaciones para ContikiOS

Para una correcta compilación de las aplicaciones que acompañaran a contikiOS es importante mantener un orden en los archivos makefile. Una de las secciones o conjunto de directivas más importantes dentro makefile.include corresponde a las especificaciones de las aplicaciones que serán cargadas al sistema operativo.

```
include $(CONTIKI)/core/net/Makefile.uip
include $(CONTIKI)/core/net/rpl/Makefile.rpl
```

Código 1: Directivas de compilación para agregar aplicaciones a contikiOS

En el código anterior se especifican makefiles de aplicaciones o librerías que serán compiladas junto al sistema operativo en este caso los protocolos uIP y RPL.

9.4. Instalación de nuevas aplicaciones

La construcción de aplicaciones dentro de ContikiOS requiere de directivas que permiten agregar una aplicación independiente. a continuación, se describe un makefile para una aplicación simple denominada ejemploTesis y se almacena en el directorio /apps/ejemploTesis es importante esta última ruta porque refleja que por defecto las aplicaciones se buscan en el directorio Apps

```
CONTIKI = ../..
all: ejemploTesis
include $(CONTIKI)/Makefile.include
```

Código 2: Especificación básica de makefile para aplicaciones

En la primera línea se especifica la variable de ambiente CONTIKI que contendrá la ruta de la ubicación del código fuente de ContikiOS, en la siguiente línea el nombre de la aplicación para la compilación automática y por último la especificación del makefile principal de ContikiOS.

9.5. Definición de Target

La compilación de una aplicación en ContikiOS está fuertemente ligada a una variable de ambiente denominada TARTGET, en esta variable se especifica la plataforma sobre la que se va a instalar ContikiOS. Al no especificarlo, el sistema determina que el código será ejecutado en la

plataforma configurada en el fichero Makefile, o bien en la plataforma por defecto que corresponde a la plataforma anfitrión.

Al compilar utilizando un Target por defecto, se genera un Archivo ejecutable, bajo el formato "<nombre_app>.native".

Existen diferentes Targets implementados para cargar aplicaciones en ContikiOS.

- Native: Plataforma por defecto. Utilizada para ejecutar aplicaciones bajo el Sistema Operativo del equipo.
- Sky: Utilizada para cargar aplicaciones desarrolladas sobre motas TelosB.
- Z1: Utilizada para aplicaciones desarrolladas sobre motas Zolertia1.
- Minimal-net: Plataforma utilizada para ejecutar aplicaciones con los protocolos de comunicación básicos.

Las motas TelosB se basan en la plataforma sky. Por lo tanto, se debe especificar el target en cada compilación, o bien modificar el target por defecto, utilizando la función savetarget, como se muestra a continuación

```
sudo make TARGET=sky savetarget
```

Código 3: Establece la plataforma de Compilación

10. TinyOS en Plataforma TelosB

En esta sección se efectuarán pruebas del sistema operativo TinyOS sobre una plataforma compatible con TelosB que permitirá analizar el comportamiento de alguna característica básica del sistema operativo.

Al igual que en el apartado anterior para efectuar las pruebas se establece la siguiente configuración de Hardware:

Equipo anfitrión

-Equipo IBM Lenovo X200 - https://support.lenovo.com/cl/es/documents/migr-73156

Nodo Sensor o Mota

-CM5000 TelosB - http://www.advanticsys.com/shop/mtmcm5000msp-p-14.html

En primer lugar, se realizará la instalación del sistema operativo en el dispositivo telosB, en segundo lugar, se realizarán pruebas de algunos ejemplos de redes de sensores básicos.

10.1. Instalación y Compilación de TinyOS

El procedimiento de instalación de TinyOS es más simple que el de ContikiOS, y la razón es simple, este sistema cuenta con repositorios para Ubuntu que permiten automatizar el procedimiento.

El repositorio para TinyOS está disponible en:

```
http://github.com/tinyos/tinyos-release/archive/tinyos-2_1_2.tar.gz
```

Una vez cargadas los repositorios correspondientes se procede a instalar algunos paquetes necesarios para el funcionamiento del kit de herramientas del sistema operativo.

```
sudo apt-get install autoconf2.13
sudo apt-get install build-essential
sudo apt-get install openjdk-6-jdk
sudo update-alternatives --config java
sudo apt-get install graphviz
sudo tos-install-jni
sudo apt-get install python2.7 python2.7-dev
```

Para un correcto funcionamiento de las aplicaciones de ejemplo se requiere las configuraciones de variables de ambiente.

```
export TOSROOT="Aqui la Ruta de TinyOS (con comillas)"
export TOSDIR="$TOSROOT/tos"
export CLASSPATH=$CLASSPATH:$TOSROOT/support/sdk/java
export MAKERULES="$TOSROOT/support/make/Makerules"
export PYTHONPATH=$PYTHONPATH:$TOSROOT/support/sdk/python
```

Una forma de comprobar el correcto funcionamiento del ambiente de trabajo es ejecutar el siguiente script que indicara todos los posibles problemas de configuración.

```
$ tos-check-env
```

La mayoría de los mensajes de error que se generaron son autoexplicativos, pero cuando tenemos mensajes que no son muy claros se pude participar de la lista de correo (tinyos-help email list) en la cual están los desarrolladores del script, ellos solucionan los problemas no documentados y que nos están cubiertos por los test que realiza el script.

Otro script importante a ejecutar es el que permite chequear si están habilitadas las opciones de compilación para TinyOS

```
# printenv MAKERULES
```

Si estos dos scripts se ejecutan sin problemas, el sistema está listo para cargar y compilar las aplicaciones a la mota TelosB.

De la misma forma como se compila el ContikiOS se requiere compilar cada aplicación para la plataforma en la cual se cargará y ejecutará.

10.2. Administración de dispositivos sensores desde Ubuntu

El trabajo con dispositivos sobre TinyOS es apoyado con una serie de herramientas que podemos utilizar para hacer tareas tan complejas como simular redes de sensores o tan simples y útiles aplicaciones autónomas que permiten la interacción con los dispositivos.

Una herramienta para visualizar los dispositivos conectados un equipo anfitrión es `motelist`, esta utilidad despliega información precisa para el posterior trabajo de compilación y carga del sistema operativo en el dispositivo.

```
Comando:

#./motelist-linux

Salida:

Reference  Device            Description
---------- ---------------- --------------------------------

FTVW0FO9   /dev/ttyUSB0     FTDI MTM-CM5000MSP
```

Una vez que tenemos conectado el dispositivo al USB y ha sido detectado correctamente podemos iniciar la tarea de la compilación del sistema operativo para ser instalado en una Mota TelosB.

10.3. Compilación de aplicaciones para TinyOS

La compilación de aplicaciones en TinyOS es relativamente simple, cuenta al igual que ContikiOS con una completa configuración de las herramientas GNU/make que permite automatizar este proceso.

Para compilar una aplicación correctamente estructurada el comando es el siguiente:

```
# make [plataforma]
```

Esta instrucción tiene como parámetro la plataforma sobre la cual se cargará la aplicación, esta opción puede ser: epic, eyesIFX, intelmote2, iris, mica, mica2, mica2dot, micaz, mulle, sam3s_ek, sam3u_ek, shimmer, shimmer2, shimmer2r, span, telosa, telosb, tinynode, ucmini, z1.

Para la compilación y carga de la aplicación en la mota telosb el comando es

```
#make telosb install
```

Para el correcto funcionamiento de GNU/make es imperativo la correcta configuración de la variable de ambiente $MAKERULES

```
TOSROOT="/opt/tinyos-2.1.2"
MAKERULES="$TOSROOT/support/make/Makerules"
```

10.4. Interface de desarrollo Integrado (IDE) para TinyOS

En todo proceso de construcción de aplicaciones moderno una herramienta importante que se requiere es la Interface de Desarrollo Integrado (IDE), en este sentido TinyOS se integra bastante bien al entorno de desarrollo Eclipse, para ello se requiere instalar una extensión denominada Yeti, estas dos piezas de software (Eclipse + Yeti) contribuyen de manera muy eficaz a acelerar el proceso de codificación y depuración de aplicaciones, por ejemplo ofrece la posibilidad de auto completado de código como se muestra a continuación. en que las principales opciones de codificación de TinyOS están disponibles de manera rápida y simple.

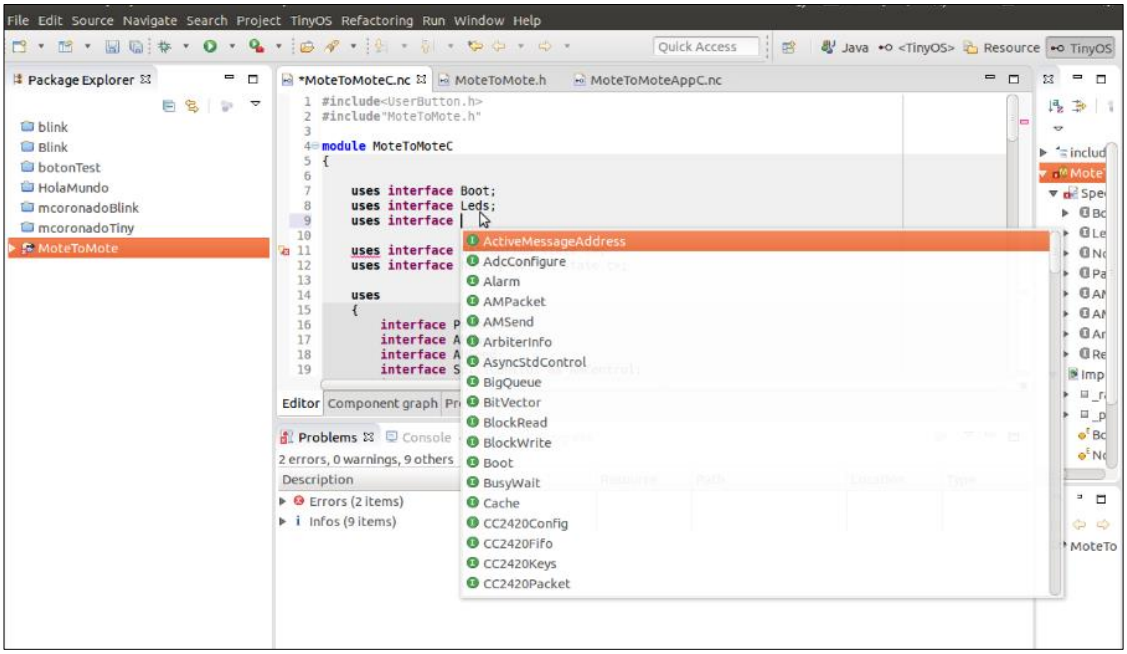

Figura 19 - AutoCompletado Eclipse

Para instalar Yeti está disponible un repositorio que permite instalarlo de forma simple, rápida y en cualquier Sistema Operativo.

```
http://tos-ide.ethz.ch/update/site.xml
```

10.5. Anatomía de las aplicaciones en TinyOS.

En TinyOS como se menciona en apartados anteriores el lenguaje de programación es nesC (Network Embedded Systems C) es una versión del lenguaje de programación C optimizado para las limitaciones que presentan los equipos en redes de sensores.

TinyOS al tener como lenguaje de programación nesC hace que la construcción de aplicaciones sea relativamente simple para alguien que este familiarizado con la programación en C, C++ o Java.

A continuación, se estudia cómo se construye una aplicación para TinyOS. No discutiremos los temas relativos al lenguaje porque las similitudes con C y C++ hacen que sea redundante explicar cómo administra el dispositivo. Este apartado se centrará principalmente en mostrar el nivel de dificultad que tiene la construcción de una aplicación para redes de sensores sobre TinyOS.

Una aplicación típica para TinyOS requiere tres archivos fuentes o componentes, el primero denominado Archivos de Configuración e Implementación, el segundo es un Archivo de Módulos y por último los Archivos de Librerías. En la siguiente imagen se aprecia el ciclo de compilación de una aplicación para TinyOS.

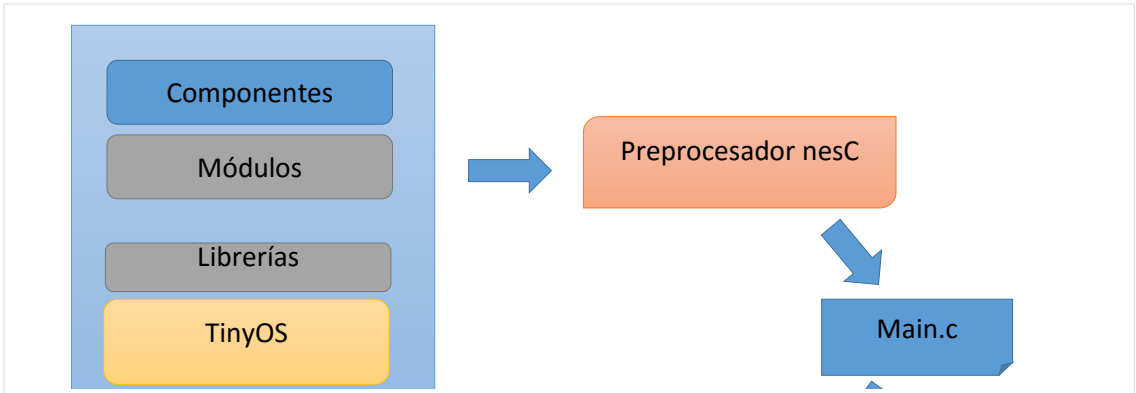

Figura 20 - Ciclo de compilación de una aplicación

Archivos de Módulos: Proveen el código de la aplicación, implementando una o más inte

nfiguraci ión: Las configuraciones son usadas para unir las c sí, conectando las interfaces que algunas componentes proveen con las interfaces que otras usan.

Archivos de Librerías: Corresponden a las librerías típica del lenguaje que permiten hacer algunas tareas especiales, por ejemplo, para el despliegue de mensajes por medio de la interface serial se utiliza una librería especial denominada printf.h, con un modo de uso similar a como se realiza en lenguaje C.

En términos generales una aplicación se verá representada como un conjunto de componentes, agrupados y relacionados entre sí por medio de las interfaces como se aprecia en la siguiente figura.

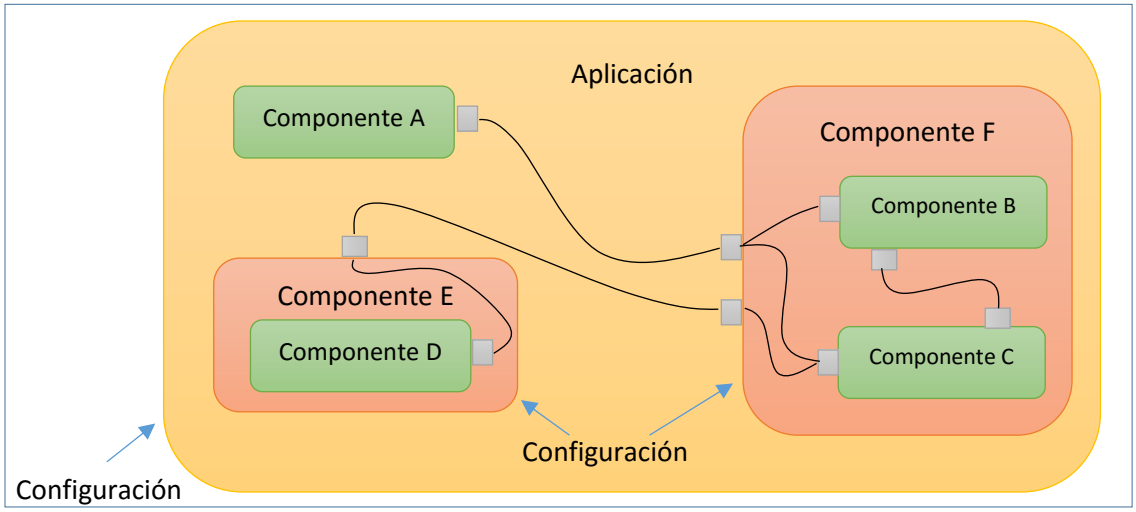

Figura 21 - Diagrama de Componentes de una aplicación genérica para TinyOS

Dentro de una aplicación típica existen dos tipos de componentes:

- Módulos, que implementan especificaciones de una componente.
- Configuraciones, que se encargarán de unir diferentes componentes en función de sus interfaces, ya sean comandos o eventos.

Las interfaces se utilizan para operaciones que describen la interacción bidireccional; el proveedor de la interfaz debe implementar comandos, mientras que el usuario de la interfaz debe implementar eventos.

Para graficar de mejor forma la estructura de aplicaciones analizaremos los Códigos fuente de una miniaplicación que permite comunicar dos motas TelosB.

En el anexo A se muestran las codificaciones de una aplicación en red Multicast que envía una señal a los Led de las demás motas presentes en la red, en esta aplicación se pude observar la separación de la aplicación en tres archivos Módulos (imagen A-1 y A-2), Componentes (A-3) y de Librería (A-4).

10.6. Concurrencia en tareas y en eventos

La metodología de diseño en programación para el trabajo sobre este tipo de sistemas, es conocida como “Programación Orientada a Componentes”, y se encuentra referido a la forma en que se conciben las aplicaciones, estas se diseñan de forma modular, definiendo entradas y salidas, todo esto tiene que ir acompañado de la concurrencia de las tareas.

Las dos fuentes de concurrencia en TinyOS son las tareas y los eventos. Las componentes entregan tareas al planificador, siendo el retorno de éste de forma inmediata, aplazando el cálculo hasta que el planificador ejecute la tarea. Las componentes pueden realizar tareas siempre y cuando los requerimientos de tiempo no sean críticos. Para asegurar que el tiempo de espera no sea muy largo, se recomienda programar tareas cortas, y en caso de necesitar procesamientos mayores, se recomienda dividirlo en múltiples tareas. Las tareas se ejecutan en su totalidad, y no tiene prioridad sobre otras tareas o eventos. De la misma forma funcionan los eventos, pero estos sí pueden interrumpir otros eventos o tareas, con el objetivo de cumplir de la mejor forma los requerimientos de tiempo real (Correa, 2017).

Todas las operaciones de larga duración deben ser divididas en dos estados: la solicitud de la operación y la ejecución de ésta. Específicamente si un comando solicita la ejecución de una operación, éste debiese retornar inmediatamente mientras que la ejecución queda en mano del planificador, el cual deberá señalizar a través de un evento, el éxito de la operación.

Una ventaja secundaria de elegir este modelo de programación es que propaga las abstracciones del hardware en el software. Tal como el hardware responde a cambios de estado en sus pines de entrada/salida, nuestras componentes responden a eventos y a los comandos en sus interfaces.

Ahora se tiene claro que las aplicaciones TinyOS son construidas por componentes. Una componente provee y usa interfaces. Estas interfaces son el único punto de acceso a la componente. Además, una componente estará compuesta de un espacio de memoria y un conjunto de tareas (Levis, 2004).

10.7. Conclusiones

La programación orientada a objetos se focaliza en las relaciones que hay entre las clases combinadas dentro de un gran ejecutable binario, mientras que la programación orientada a componentes se centra en módulos intercambiables que trabajan de forma independiente y de los cuales no es necesario saber nada acerca de su implementación interna, esto haces que la reutilización sea mucho más fácil y no requiera un entendimiento completo del proyecto de programación haciendo que las soluciones sean simples de organizar, muchas veces con solo hacer una interface que reúna varios componentes es suficiente para resolver un problema de comunicación, además al ser los sensores unidades no muy complejas basta una mínima cantidad de componentes para realizar todas las funciones de un dispositivo, es importante dejar claro que las simplicidad a la que se hace mención es en relación a un sensor independiente, esta complejidad cambia al momento de hacer interactuar los equipos dentro de red.

Un problema complejo de resolver en TinyOS es diferenciar un nodo dentro de la red para que tenga un comportamiento distinto con una misma implementación del Sistema Operativo TinyOS, pero se subsana fácilmente al tener diferentes implementaciones para cada mota con el costo relacionado que esto implica.

Haciendo un análisis de esta forma de programación en esta plataforma de Internet de las Cosas (Orientada a Componentes), resulta complejo efectuar el cambio de paradigma para aquellos programadores que fueron instruidos de forma tradicional bajo el paradigma de la Programación Estructural, sin embargo es muchos más rápida la adaptación para aquellos que fueron formados bajos la programación Orientado a Objeto, esta afirmación se basa en la observación de las curvas de aprendizaje en estudiantes con los cuales se realizó transferencia tecnológica.

Concluido este análisis es TinyOS el sistema que ofrece mejores condiciones para el desarrollo de aplicaciones, destacando una mejor curva de aprendizaje bajos ciertas condiciones (p.e. es ideal el dominio de la programación orientada a objetos) y una capacidad de reutilización de código que permite la construcción de aplicaciones de forma simple.

En el siguiente apartado analizaremos las variables que conjuga un programador al momento de comparar dos implementaciones sobre distintos lenguajes de programación, con la salvedad que en este caso las diferencias son más profunda ya que al cambiar el sistema operativo la manera de abordar las soluciones es bastante diferentes.

11. Curvas de Aprendizaje para el Desarrollo de Aplicaciones.

A continuación, se expondrán el análisis y conclusiones referentes a la facilidad que tienen estos dos sistemas operativos al momento de construir aplicaciones.

11.1. Lenguajes de Programación

TinyOS presenta unas características algo diferentes a otros sistemas operativos que podrían ser sus competidores (e.j. ContikiOS, ucOS-II, etc.) y que al principio pueden alargar un poco su curva de aprendizaje. El lenguaje de programación es NesC que es muy parecido a C pero que obliga a la implementación de módulos con interfaces bien definidas y que favorecen de forma extraordinaria la reutilización de componentes. (Villanueva, 2013)

Después del desarrollo de este trabajo las frases del párrafo anterior cobran sentido, el desarrollo de aplicaciones para TinyOS son algo complejas para un programador de la vieja escuela en que la programación Orientada a Objetos no formaba parte de la formación, las dificultades se aprecian desde un inicio en algo tan simple como la cantidad de archivos individuales que se requiere para solo construir una aplicación pequeña, como la que se muestra en apéndice A, en contraposición a ContikiOS en que el desarrollo principal de la aplicación se centra en un solo archivo. En secciones posterior analizaremos otras diferencias.

11.2. Documentación

Un punto importante de TinyOS que lo hace particularmente diferenciador respecto a otros sistemas operativos para redes de sensores es su documentación, TinyOS cuenta con una abundante y ordenada documentación que facilitan el aprendizaje de sus tecnologías y forma de hacer las cosas, en el caso ContikiOS la comunidad que entrega soporte se encuentra trabajando en la construcción de sus sistema de documentación y durante el desarrollo de este documento (2015-2017) presentó bastante movimiento y actualizaciones en la estructura de su documentación, no así en el contenido de sus documentos, razón por la que hoy la mayor fuente de información como se mencionó anteriormente son los foros y listas de correo que hacen que la curva de aprendizaje de ContikiOS no sea la óptima.

11.3. Entornos Integrado de desarrollo

En este documento se presentaron los diferentes entornos de desarrollo que tiene cada uno de los sistemas operativos, ambos basados en tecnología de uno de los IDE más populares de internet en la actualidad Eclipse. Cada lenguaje de los diferentes Sistemas operativos para

sensores tiene extensiones o plugins que permiten optimizar eclipse para apoyar el desarrollo de aplicaciones.

En este sentido es TinyOS es quien lleva la delantera al contar con un plugins que permite apoyar al programador en las tareas más básicas que hoy en día son muy valoradas, como por ejemplos el autocompletado de código, que aceleran bastante la codificación sobre todo cuando se cuenta con una infinidad de componentes, y cada una con muchos métodos que es casi imposible su memorización.

12. Redes de Sensores

12.1. Comunicaciones para redes de Sensores

Para el desarrollo de este trabajo de tesis un elemento importante tiene que ver con las comunicaciones, es decir, resolver el problema del movimiento de datos dentro de red de sensores, es en este punto donde IP juega un papel importante, pero no es el IP tradicional (IPv4) sino que la última versión de este protocolo, y más aún es un IPv6 adaptado a las características de los equipos (6LoWPAN). En este apartado describiremos los elementos más importantes que nos ayudaran a comprender de mejor forma cómo es que se resuelve el problema de las comunicaciones en una red de sensores para la nueva internet.

12.2. Un poco de historia Cronológica

Como se mención en apartados anteriores las redes de sensores tienen sus orígenes en aplicaciones militares bajo el proyecto SOSUS (Whitman, 2005) y hoy en día se proyecta al futuro con el Internet de las cosas, pero para llegar de un punto a otro fueron varios los hitos que han marcado la historia de las redes de Sensores, la cronología es la siguiente.

1952 - Guerra Fría – Red de sensores SOSUS es creada para la Detección de Submarinos.

1970 - Internet - El termino Internet es acuñado por Vinton Cerft

1980 - Fines de la guerra Fría – La tecnología de las redes de sensores SOSUS es reutilizada para el estudio de las corrientes oceánicas submarinas

1990 - Uso 100% Civil – SOSUS es utilizada para el Estudios de Volcanes submarinos, Animales Marinos y Temperatura de los Océanos.

1980 - Primero Esfuerzos Independientes - Comenzó la exploración de redes de sensores con el proyecto Distributed Sensor Networks (Lincoln Laboratory, 1985)

1981 - RFC 791 - Sale al Mundo el Protocolo de Internet (IP).

1983 - TCP/IP - conjunto de protocolos vital para la interconexión de redes.

1990 - Aparece el WWW de los servicios disponibles en la red

1995 - IPv6 - El tamaño de las redes hace necesarios una mejora del protocolo Internet

1999 - El término Internet of Things fue acuñado por Kevin Ashton

2000 - Primer Lanzamiento de TinyOS un sistema operativo para redes de Sensores

2014 - TelosB - lanzado al mercado

2005 - 6loWPAN Group

2017 - Programming Memory-Constrained Networked Embedded Systems. Nombre de las tesis que da Origen a ContikiOS.

2014 - The Constrained Application Protocol (CoAP)

De todos hitos mencionado en la cronología y que están relacionadas a este trabajo, IPv6 es una tecnología que se lleva el peso de hacer posible la conexión de miles de millones de "cosas" a Internet y construir así el Internet de las Cosas.

Es así que IPv6 se despliega por el mundo con el objetivo de conectar todo tipo de dispositivo solucionando las limitaciones de IPv4, incluso contamos con tecnologías que fueron desarrollas solo pensando en IPv6. Un buen ejemplo en el contexto de la internet de las cosas es 6LoWPAN.

12.3. IPv6, el nuevo Protocolo de Internet.

En el estudio de las comunicaciones al conocer IPv4 hoy en día se está a la mitad del camino. En esta sección se abordarán los conceptos de IPv6 que tienen relación con Internet de las cosas y cómo son proyectados a otros protocolos como el 6LoWPAN, del que se hablará más adelante.

Para iniciar este estudio en necesario tener claro desde un principio que IPv6 es un protocolo diferente que no es compatible con IPv4.

IPv6 está en la capa tres del modelo TCP/IP, también llamada capa de red, los datos manejados por la capa 3 se llaman datagramas. Los dispositivos conectados a la Internet pueden ser anfitriones (hosts) o enrutadores (routers). Un host en términos concretos puede ser un computador de escritorio, un celular o en el contexto de las redes de sensores una tarjeta censora de humedad. Los hosts son los destinatarios de los paquetes. Los routers, en cambio, se encargan del transporte de los paquetes y son responsables de seleccionar el próximo enrutador que reenviará los datos a su destino final.

12.4. Propiedades IPv6 importantes para Redes de Sensores

Estudiaremos dos de las propiedades de IPv6 que son explotadas por las redes de sensores y estas son:

- Direccionamiento global
- Autoconfiguración de las direcciones de los hosts.

Dada la cantidad de direcciones que IPv6 entrega, es común ver en la literatura autores que dicen "*Se dispone de direcciones para cualquier cosa que se nos pueda ocurrir conectar a internet"*, esto es explotado por las redes de sensores que buscan desplegarse en todo lugar con muchos dispositivos interconectados (direccionamiento global), el problema está en la administración de esta cantidad de direcciones, de tal manera que no sea un problema para el usuario final estar configurando manualmente cada dispositivo, es por esos que IPv6 dispone de mecanismos que permiten la autoconfiguración:

- DHCPv6. (Configuración Dinámica del Host para IPv6). Es un mecanismo de autoconfiguración en que un equipo se encarga de proporcionar los parámetros de direccionamiento a todos los dispositivos.

- SLAAC (Auto configuración direcciones sin Estado). Este es un mecanismo nuevo desarrollado para IPv6, que permite configurar los parámetros desde el router, este mecanismo es el mejor para redes de sensores, no requiere un dispositivo adicional (DHCPv6) si no que es el router el que tiene el control de toda la configuración de los dispositivos en la Red.

El mecanismo que se utiliza con mayor regularidad en redes de sensores es SLAAC por la simplicidad que significa la autoconfiguración de los nodos, esto impide que el nodo se recargue de actividades apoyándose en el protocolo Neighbor Discovery (ND).

Como lo indica el nombre, SLAAC quiere decir "sin estado". Un servicio sin estado significa que no hay ningún servidor que mantenga la información de la dirección de red. A diferencia de DHCP, no hay servidor de SLAAC que tenga información acerca de cuáles son las direcciones IPv6 que están en uso y cuáles son las que se encuentran disponibles, esto se puede presentar como una desventaja porque se pierde la identidad del nodo, pero en la mayoría de las aplicaciones en que los nodos son utilizados como dispositivos que solo capturan información (sensor) no es necesario conocer la identidad del nodo y muchas veces la identidad del nodo es parte de la información que viaja al recolector (Castellani, A., Loreto, S., Rahman. ,2016).

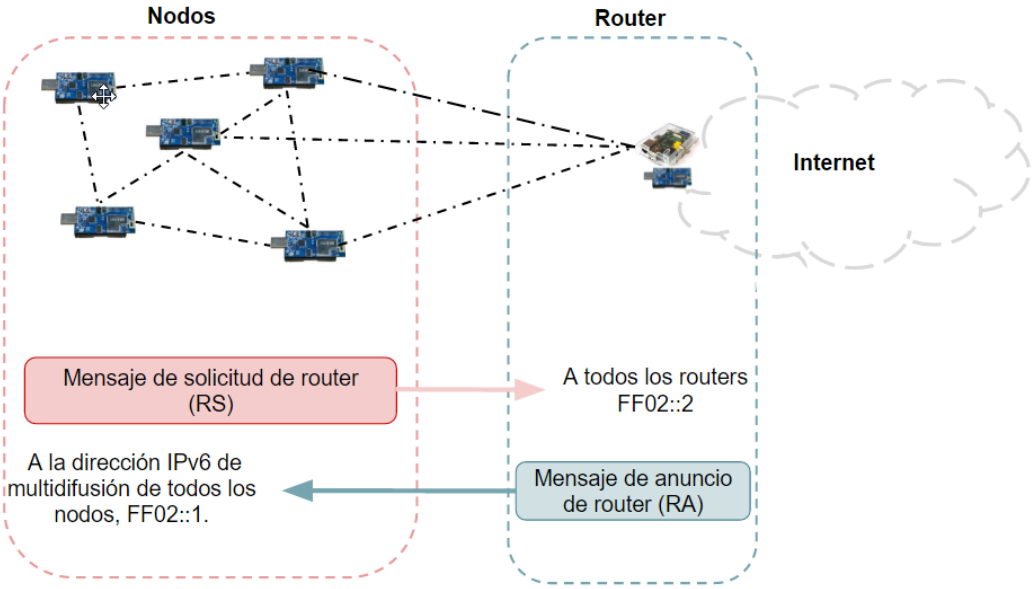

Figura 22 - Mensajes de Autoconfiguración.

Funcionamiento del Mecanismo de Autoconfiguración:

Mensaje de solicitud de router (RS): cuando un cliente está configurado para obtener la información de direccionamiento de forma automática mediante SLAAC, el cliente envía un mensaje RS al router. El mensaje RS se envía a la dirección IPv6 de multidifusión de todos los routers, FF02::2.

Mensaje de anuncio de router (RA): los routers envían mensajes RA para proporcionar información de direccionamiento a los clientes configurados para obtener sus direcciones IPv6 de forma automática. El mensaje RA incluye el prefijo y la longitud de prefijo del segmento local. Un cliente utiliza esta información para crear su propia dirección IPv6 de unidifusión global, Los routers envían mensajes RA de forma periódica o en respuesta a un mensaje RS. De manera predeterminada. Los mensajes RA siempre se envían a la dirección IPv6 de multidifusión de todos los nodos, FF02::1.

En este punto tenemos los nodos conectados al router por con la ayuda de IPv6 el siguiente paso es la implementación de aplicaciones que permitan explotar la red de sensores y subimos en la Pila de Protocolos TCP/IP

12.5. El Intercambio de Información a nivel de aplicaciones mediante CoAP

Una vez implementada la red IPv6 para la red de sensores se requiere un protocolo superior que permita el intercambio de información de manera simple desde las motas a la estación de recolección con un mínimo consumo de recurso de procesamiento, almacenamiento y ancho de banda también que sea fácil de manejar desde el punto de vista del programador. Bajo estos requerimientos y después de un estudio de diferentes experiencias el protocolo que cubre en gran medida los requerimientos es CoAP (Constrained Application Protocol) es un protocolo web genérico que permite el intercambio de información a nivel de aplicación.

El Protocolo CoAP fue concebido para redes de bajo consumo, especialmente diseñado para transferencia de información de forma simple y sin mayores requerimientos para las aplicaciones, es por este motivo que se inserta en este trabajo de tesis que busca contar con todas las componentes necesarias para la construcción de aplicaciones sobre redes de sensores.

CoAP, al igual que HTTP, se basa en la arquitectura REST, pensada para Sistemas Distribuidos, se podría resumir como una interfaz entre sistemas que utiliza directamente HTTP para obtener datos o indicar la ejecución de operaciones sobre los datos, también hace uso del modelo cliente/servidor en que los clientes envían peticiones a los servidores y éstos responden, en general, con una representación del recurso solicitado, por ejemplo, valores de Sensores de la Red.

A parte de las características propias de HTTP el protocolo CoAP ofrece una serie de características que lo hacen apropiado para las redes de sensores como por ejemplo el descubrimiento de recursos de los servidores disponibles en la red, un soporte a las comunicaciones multicast y el intercambio asíncrono de mensajes.

La principal diferencia con HTTP se ve a nivel de protocolo de transporte en el caso de CoAP se utiliza UDP (User Datagram Protocol) para el envío de mensajes de manera asíncrona, en cambio en HTTP utiliza TCP (Transmission Control Protocol) para el envío de peticiones y respuestas (Ludovici,2013).

La Anatomía del protocolo es la siguiente: Una petición en CoAP es enviada por un cliente sobre un recurso almacenado en un servidor, que para nuestro caso podría ser un sensor, la respuesta contiene un código que indica si la petición era válida o si tenía algún problema; esta respuesta puede contener también una representación del recurso.

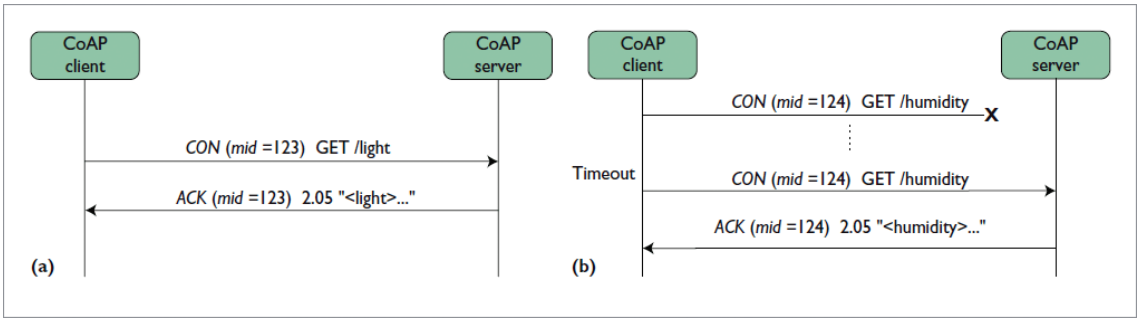

Figura 23 - Ejemplo de Funcionamiento CoAP (Fuente:The IEEE Computer Society)

En la figura anterior se muestran dos escenarios del protocolo CoAP:

a.- El cliente solicita el valor del sensor de iluminación mediante un mensaje CON y recibe una respuesta desde el servidor con un mensaje ACK, se aprecia la utilización de MID (mid=123) para la identificación del intercambio de mensajes.

b.- El cliente envía la solicitud al recurso de humedad (GET /humidity) con el número de identificación (mid=124) este tiene un primer intento que fracasa porque no se recibe la respuesta dentro del tiempo de espera por tanto es reenviada la solicitud con la misma identificación y en la segunda instancia si se obtiene el mensaje ACK y el valor del recurso solicitado (2.05).

Como se observa la comunicación sobre CoAP se efectúa por medio de mensajes codificados, estos serán analizados en el siguiente apartado.

12.5.1.Mensajes CoAP

En este apartado se efectuará un análisis de la anatomía de los cuatro tipos de mensajes CoAP, cada uno de ellos tiene tres secciones la cabecera que es la parte fija, una sección de opcionales y por último una tercera sección que es el payload o contenido del mensaje.

Tipos de Mensaje:

Confirmable (CON):	Los mensajes de este tipo requieren una confirmación de recepción por parte del receptor.
Non-Confirmable (NON):	Este tipo de mensajes se envían cuando no es imperativo que el destinatario reciba el mensaje, puesto que no requiere confirmación.
Acknowledgement (ACK):	Este tipo de mensajes se envían para confirmar la recepción de un mensaje de tipo CON o para responder a una petición de tipo GET.
Reset (RST):	Este mensaje se entrega como respuesta a un mensaje (CON o NON) recibido pero que, el receptor es incapaz de procesar, aunque su contenido sea correcto.

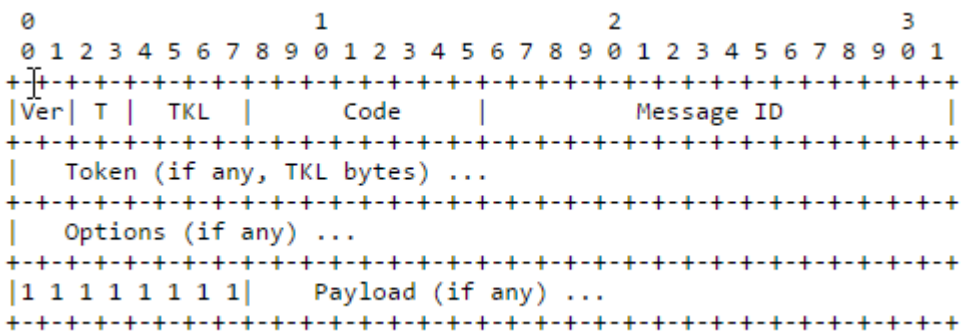

Figura 24 - Estructura del mensaje CoAP - Fuente: RFC 7252

Nombre Campo	Descripción
Ver	Indican la versión de CoAP. Las implementaciones de esta especificación deben establecer este campo con el valor 1. (2 bit)
T	Indican el tipo de mensaje: CON, NON, ACK o RST. (2 bit)
TKL	Este campo indica la longitud del campo Token, el cual está a continuación de la cabecera. (4 bit)

Code	Este valor indica si el mensaje se trata de una petición (valores 131) de una respuesta (64-191) o si está vacío (0). (8 bit)
Message ID	Un valor que identifica al emisor del mensaje
Token	El Token es un valor pensado para ser gestionado de forma local, para que un cliente pueda diferenciar de forma concurrente las peticiones que tiene en curso.
Options	Las opciones contenidas en un paquete CoAP se encuentran a continuación de la cabecera. Estas opciones deben estar ordenadas mediante su Option Number, los Option number impares corresponden a opciones críticas y los pares a opcionales.
Payload	Carga Útil

12.5.2.Funcionamiento del Protocolo CoAP

CoAP se basa en el intercambio de mensajes asíncronos entre dos nodos. Un nodo actuando de cliente envía una o más peticiones sobre uno o más recursos alojados en un determinado servidor, que atenderá la petición. El servidor responderá a la petición indicando el éxito o no de la petición recibida.

Request

Una petición (request) se envía mediante un mensaje de tipo CON o NON y consiste en una petición para ejecutar un método sobre un recurso, el cual viene identificado a través de la opción URI-Path contenida en el paquete.

Los métodos soportados en CoAP son GET, POST, PUT y DELETE. El método viene indicado en el campo Code de la cabecera y tienen las mismas propiedades que en HTTP.

Responses

Análogamente al caso anterior, el tipo de respuesta (response) viene dada por el código del campo Code. Éste puede ser de tres clases diferentes indicando éxito o error y en que parte se produce dicho error:

- 2.xx: Success. La petición fue recibida, entendida y aceptada correctamente.
- 4.xx: Client Error. La petición contiene una sintaxis errónea o no puede ser correctamente tratada por el servidor.
- 5.xx: Server Error. El servidor no puede tratar una petición aparentemente correcta.

Piggy-backed

La respuesta de tipo Piggy-backed se da cuando el servidor responde inmediatamente a una petición recibida (de tipo CON) y envía un mensaje de tipo ACK. Este tipo de respuestas se dan independientemente de si la respuesta indica éxito o fallo al tratar la petición.

Separate

Cuando se recibe una petición puede darse la situación que el servidor no sea capaz de responder al momento, sea porque no dispone de acceso temporalmente al recurso o porque está saturado de tareas.

Las respuestas a peticiones de tipo que no requiere confirmación (NON) son siempre enviadas mediante este tipo de mensajes, puesto que no existe confirmación (mensaje ACK) para este tipo de peticiones.

Si la petición se recibe a través de un mensaje CON y el servidor no puede enviar la respuesta de forma inmediata responde con un ACK confirmando que ha recibido la petición y que ésta será tratada tan pronto como sea posible. Posteriormente, la respuesta con el contenido del recurso se envía con un mensaje de tipo CON que deberá ser confirmado por el cliente para asegurar que éste último ha recibido correctamente la respuesta.

Token

Un nodo puede tener más de un mensaje pendiente de confirmar o simplemente interactuar con diferentes nodos a la vez. Es por esto que se requiere un método para poder determinar que un mensaje recibido es la respuesta a un mensaje concreto y no a otro. Esto se consigue mediante el uso del Token.

El Token es un valor pensado para ser gestionado de forma local, para que un cliente pueda diferenciar de forma concurrente las peticiones que tiene en curso. El valor que toma el Token debe ser actualizado de forma que todas las peticiones que un cliente tenga pendientes de recibir respuesta tengan un valor de Token distinto. Un nodo que recibe un paquete que contiene un Token debe responder siempre manteniéndolo y sin alterar el valor del Token. Si además la respuesta es de tipo Piggy-backed tampoco se debe de alterar el valor del Message ID. Si la respuesta es de tipo Separate se debe de añadir el Token de la petición original.

Con el modelo de mensajes anterior se desprende que la fiabilidad que ofrece CoAP, aunque trabaja sobre un protocolo no fiable como el UDP, se consigue enviando los mensajes de tipo CON, que deberán ser siempre confirmados con un mensaje ACK.

Cuando un cliente envía un mensaje CON y pasado un determinado tiempo no recibe respuesta, volverá a enviar la petición original. El tiempo entre reenvíos sufrirá un back-off exponencial por si no se recibiera el ACK y se tuviera que volver a enviar la petición. Agotadas el número máximo de re-transmisiones el mensaje CoAP será borrado y no se volverá enviar (Ashish Patro, S. B. (2015).

Métodos CoAP

Anteriormente se ha comentado la existencia de cuatro métodos posibles a ser aplicados sobre un recurso, a continuación, se detalla cada uno de ellos.

GET

El método GET pide una representación de la información correspondiente al recurso que viene indicado mediante la inclusión de la opción URI-Path. Es un método seguro e ídempotente al cual se debe responder con un código 2.05 (Content) o 2.03 (Valid) si se trata de una petición válida.

POST

Este método requiere que la representación del recurso que hay incluida con la petición enviada sea procesada. El servidor debe responder con código 2.01 (Created) en caso de haberse creado un nuevo recurso en el servidor y deberá incluir en la respuesta la URI donde se aloja este recurso, que será indicada mediante una o más opciones Location-Path y/o Location-Query.

Si en cambio el recurso no se crea (porque ya existía) y se trata de una petición válida, el servidor contestará con una respuesta 2.04 (Changed), indicando que el recurso ha sido cambiado según las indicaciones incluidas en la petición. Este método no es ni seguro ni ídem-potente.

PUT

Una petición con este método indica al servidor que el recurso indicado mediante su URI debe ser actualizado o creado con la representación que de éste se incluye en el mensaje.

Las respuestas enviadas por el servidor son idénticas al caso del método POST, excepto que en caso en el que se cree un recurso no se enviará la dirección URI en la respuesta, puesto que está ya venía indicada en la petición. Se trata de un método no seguro, pero ídem-potente.

DELETE

Una petición con este método solicita que el recurso identificado mediante la URI adjunta sea eliminado. La respuesta del servidor deberá ser con código 2.02 (Deleted) en caso de que se haya eliminado correctamente o si el recurso no existía al recibir la petición.

CoAP en ContikiOS

En ContikiOS el protocolo CoAP esta implementado en la librería denominada LibCoap que fue desarrollada como extensión de ContikiOS, hoy en día están fuertemente relacionados pero continua sus desarrollos de forma independiente, al momento de la construcción de este documento la versión 3 de ContikiOS no cuenta con soporte 100% de CoAP pero se espera su pronta publicación oficial. La versión 2.7 de ContikiOS cuenta con la implementación de CoAP y este trabajo se basa principalmente en ella.

La implementación de CoAP (libCoap) en ContikiOS respeta el estándar la IETF en el RFC 7252 (Shelby, Z.,2014)

Libcoap está diseñada para ejecutarse en dispositivos embebidos con muy pocos recursos, así como también en sistemas informáticos de gama alta con POSIX OS. Esto permite desarrollar y probar las aplicaciones en computadoras portátiles y luego mover las aplicaciones a los dispositivos destino esta es una característica que no está disponible en otras librerías y que hace más fácil el desarrollo de aplicaciones para redes de Sensores (Chun, S.-M., & Park, J.-T. 2015).

Esta librería ha tenidos gran éxito en el mercado y se utiliza en una gran variedad de proyectos relacionados a la Internet de las cosas tanto comerciales como académicos, esto permite una interoperabilidad entre distintas soluciones.

Una de las principales características que la hacen tan popular es que en su implementación pose una gran variedad de aplicaciones como servidores, clientes y herramientas de monitorización que simplifican bastante el trabajo y el despliegue de soluciones con esta librería.

Para la utilizar la librería en ContikiOS solo se requiere agregar al directorio del sistema operativo la carpeta que contiene LibCoap (contiki/apps) y en cada aplicación incluir la directiva APPS += libcoap en los makefiles de las aplicaciones (Manveer Joshi, B. P.,2015).

```
CFLAGS += -ffunction-sections
LDFLAGS += -Wl,--gc-sections,--undefined=_reset_vector__,--undefined=InterruptVectors,

CFLAGS += #-DSHORT_ERROR_RESPONSE -DNDEBUG

APPS += libcoap

include $(CONTIKI)/Makefile.include
```

Figura 25 - Directiva de librería LibCoap

En la figura anterior se muestra la inclusión de la librería libcoap que permite la utilización del protocolo CoAP y de esta forma construir una aplicación CoAP-Compatible

Ahora estudiaremos las implementaciones principales de CoAP para ContikiOS y que permiten la utilización de forma simple de funciones y métodos.

Si tenemos que mencionar alguna debilidad de esta librería es la escasa documentación, la mayoría de las referencias se direccionan a los ejemplos desarrollados es por esto que este trabajo de tesis cobra importancia porque representa una contribución al avance de estas implementaciones que al carecer de información clara y centralizada muchas veces es descartada como una solución viable.

Implementación de CoAP.

En apartados anteriores se estudió la forma en que se construyen las aplicaciones en ContikiOS mediante macros en C, basados en ello, ahora estudiaremos los métodos, funciones y macros que permiten la implementación del protocolo CoAP en cualquier aplicación.

El análisis de la implementación se efectuará sobre ejemplo fáciles de entender y que permitan marcar el punto de inicio a la construcción de aplicaciones más complejas.

Las operaciones que realiza el servidor se resumen en la lectura de los sensores para luego enviarlos a la red.

La definición de librerías para la utilización de las macros, funciones y métodos marca el punto de partida en la programación de aplicaciones.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include "contiki.h"
5 #include "contiki-net.h"
6 #include "rest.h"
```

Librerías ContikiOS Para Aplicaciones CoAP

En Listado de código anterior en las tres primeras líneas se declaran las librerías típicas de cualquier aplicación en lenguaje C y tiene relación con las funciones de entrada/salida y procesamiento de texto. en la línea 4 se declara las funciones de ContikiOS y la inclusión de las funciones de red en la línea 5, estas fueron analizadas en la sección de Sistema Operativo de

este documento. En la línea 6 se incluye la librería que hace posible la utilización del protocolo CoAP.

Quedan para futuras investigaciones elementos de seguridad (P. Porambage, A. B.,2015)

Codificación lado del Servidor

En la construcción de aplicaciones para ContikiOS se debe respetar la forma en que se definen los métodos a nivel de código (nomenclatura de codificación), por ejemplo, por cada recurso que se maneje debe también definirse un método que manipular dicho recurso, es así que se define el método de la siguiente forma

`[NombreDelRecurso]_handler`

Para ejemplificar el manejo de recursos estudiaremos el siguiente fragmento de código:

```
0 RESOURCE(helloworld, METHOD_GET, "helloworld");
1 void
2 helloworld_handler(REQUEST* request, RESPONSE* response)
3 {
4  sprintf(temp,"Hello World!\n");
5
6  rest_set_header_content_type(response, TEXT_PLAIN);
7  rest_set_response_payload(response, (uint8_t*)temp, strlen(temp));
8 }
```

Codificación Genérica de un Recurso – Fuente: Documentación ContikiOS

En la línea 0 se define el recurso por medio de una macro que tiene como parámetros el nombre del recurso y el método http que será el encargado de manejar dicho recurso. En este caso se define un método (helloworld_handler) que se encarga de manejar el recurso helloword, la interacción con el mundo exterior se realiza por medio de sus parámetros que pueden ser redefinidos dentro de cada método (línea 2).

Al ser un método del tipo `METHOD_GET` se ejecuta por medio de `GET` (llamada html) y es la línea 7 la encargada de construir la respuesta que será entregada al cliente por medio del parámetro response.

A continuación, veremos la descripción anterior por medio de un ejemplo aplicado a un sensor que entrega los valores de iluminación y está ejecutando una versión de ContikiOS 2.7 sobre una Mota TelosB (es importante este punto porque en la nueva versión de ContikiOS hasta la construcción de este documento no tenía soporte 100% para CoAP).

```
 1 RESOURCE(light, METHOD_GET, "Iluminacion");
 2 void
 3 light_handler(REQUEST* pregunta, RESPONSE* respuesta)
 4 {
 5  read_light_sensor(&light_photosynthetic, &light_solar);
 6  sprintf(Temp,"%u;%u", light_photosynthetic, light_solar);
 7  char etag[4] = "ABCD";
 8  rest_set_header_content_type(respuesta, TEXT_PLAIN);
 9  rest_set_header_etag(respuesta, etag, sizeof(etag));
10  rest_set_response_payload(respuesta, temp, strlen(temp));
11}
```

Server: Codificación del manejo del sensor de luz - Fuente: Doc LibCoAP

En la codificación anterior se muestra el manejo de un sensor de luz en una Mota TelosB en este caso el cliente envía la solicitud para obtener el valor del recurso (sensor de luz) y el servidor (TelosB) utiliza el manejador de recursos (función) `light_handler (línea 3)` para responder los valores asociados al sensor que son enviados a la red con un etag simple.

Codificación Lado del Cliente

Ahora el manejo de la recepción por el lado de cliente es tratado de la siguiente forma:

```
 1 static void
 2 handle_incoming_data()
 3 {
 4   PRINTF("Datos: %u \n", (uint16_t)uip_datalen());
 5   if (init_buffer(COAP_DATA_BUFF_SIZE)) {
 6     if (uip_newdata()) {
 7       coap_packet_t* repuesta =  (coap_packet_t*)allo-
cate_buffer(sizeof(coap_packet_t));
 8     if (respuesta) {
 9        parse_message(respuesta, uip_appdata, uip_datalen());
10
11        response_handler(respuesta);
12      }
13    }
14    delete_buffer();
15  }
16}
```

Recepción de datos por parte del cliente - Fuente: Documentación Contiki/LibCoAP

Cuando el servidor responde, El cliente invoca a la función `handle_incoming_data` (línea 2) que primero imprime el largo del paquete y se encarga de tomar el paquete de datos y procesar los mediante la función `parse_message` (línea 10) en este punto se tiene control total sobre el paquete de datos y se pueden hacer todo tipos de operaciones para controlar la recepción. Es importante mencionar que en los códigos expuestos no se hace manejo de excepciones eso escapa al alcance de este trabajo de tesis.

Otra sección importe de tener en cuenta en las aplicaciones, es la que controla la recepción y se encarga del control de los tiempos en que el sistema operativo preguntara por los mensajes entrantes o en caso contrario enviara respuesta a la red.

```
 1  etimer_set(&et, 5 * CLOCK_SECOND);
 2  while(1) {
 3    PROCESS_YIELD();
 4    if (etimer_expired(&et)) {
 5      send_data();
 6      etimer_reset(&et);
 7    } else if (ev == tcpip_event) {
 8      handle_incoming_data();
 9    }
10  }
```

Control de Tiempo Contiki - Fuente: Documentación Contiki

En el código anterior primero se establece el timer que permitirá marcar el paso en el envío y recepción de datos en la red (línea 1) luego se implementa la estructura de protohilos discutido en secciones anteriores, y por último la utilización de la función `handle_incoming_data` (línea 8) cuando tenemos datos.

Funciones de Red para CoAP

En esta sección veremos las funciones de red involucradas en el cliente porque las funciones asociadas al servidor CoAP están asociadas al Servicio Web entonces no requiere de algoritmos distintos a los ya conocidos en un típico Servidor Web, mientras que por el lado del cliente la aplicación requiere inicializar las funciones de red, las direcciones del servidor al cual se efectuara la conexión, la construcción de la consulta y por último el envío del requerimiento a la red.

En el siguiente listado de código describiremos lo elementos que se requieren para que el cliente consuma recursos al servidor

```
 1 static void send_data(void)
 2
 3{
 4  char buf[MAX_PAYLOAD_LEN];
 5  if (init_buffer(COAP_DATA_BUFF_SIZE)) {
 6    int data_size = 0;
 7    int service_id = random_rand() % NUMBER_OF_URLS;
 8    coap_packet_t* request =  (coap_packet_t*)allo-
cate_buffer(sizeof(coap_packet_t));
 9    init_packet(request);
10    coap_set_method(request, COAP_GET);
11    request->tid = xact_id++;
12    request->type = MESSAGE_TYPE_CON;
13    coap_set_header_uri(request, service_urls[service_id]);
14    data_size = serialize_packet(request, buf);
15    PRINTF("Client sending request to:[");
16    PRINT6ADDR(&client_conn->ripaddr);
17    PRINTF("]:%u/%s\n", (uint16_t)REMOTE_PORT,                ser-
vice_urls[service_id]);
18    uip_udp_packet_send(client_conn, buf, data_size);
19    delete_buffer();
20  }
21}
```

Codificación Clientes - Fuente: Documentación LIbCoAP

En el siguiente listado de código se definen constates de configuración necesarias para el funcionamiento del cliente

```
1 #define SERVER_NODE(ipaddr)   uip_ip6addr(ipaddr, 0xfe80, 0, 0, 0,
0x0212, 0x7401, 0x0001, 0x0101)
2 #define LOCAL_PORT 61617
3 #define REMOTE_PORT 61616
4 #define MAX_PAYLOAD_LEN   100
5 #define NUMBER_OF_URLS 3
6  char*  service_urls[NUMBER_OF_URLS]  =  {"light",  ".well-known/core",
"helloworld"};
```

Constantes de Configuración

```
PROCESS(coap_client_example, "cliente COAP");
AUTOSTART_PROCESSES(&coap_client_example);

PROCESS_THREAD(coap_client_example, ev, data)
{
  PROCESS_BEGIN();

  SERVER_NODE(&server_ipaddr);

  /* new connection with server */
  client_conn = udp_new(&server_ipaddr, UIP_HTONS(REMOTE_PORT), NULL);
  udp_bind(client_conn, UIP_HTONS(LOCAL_PORT));

  PRINTF("Created a connection with the server ");
  PRINT6ADDR(&client_conn->ripaddr);
  PRINTF(" local/remote port %u/%u\n",
  UIP_HTONS(client_conn->lport), UIP_HTONS(client_conn->rport));

  etimer_set(&et, 5 * CLOCK_SECOND);
  while(1) {
    PROCESS_YIELD();
    if (etimer_expired(&et)) {
      send_data();
      etimer_reset(&et);
    } else if (ev == tcpip_event) {
      handle_incoming_data();
    }
  }

  PROCESS_END();
}
```

Configuración de cliente

Con estas configuraciones es perfectamente implantable una aplicación en ContikiOS con capacidades de despliegue.

CoAP en TinyOS

En secciones anteriores se estudió la forma modular que tiene TinyOS para la construcción de sus aplicaciones, en esta sección y establecer una comparativa se abordara la implementación de una aplicación que utilice el protocolo CoAP.

La Implementación o librería de CoAP para TinyOS se denomina TinyCoAP (Ludovici, 2013) que está construida según la especificación del RFC Constrained Application Protocol (Shelby, 2014) está completamente construido en lenguaje NesC de forma nativa, existe otra implementación para TinyOS del protocolo CoAP denominada CoapBlip pero esta no se estudia en este trabajo porque es una implementación genérica para pequeños dispositivos y no forma parte de la librería de red de TinyOS. Pero la razón más importante es que al ser una librería compilada en forma externa no tiene el mismo rendimiento que TinyCoAP según algunos estudios concluyen que tiene un mayor consumo de memoria dentro del dispositivo (Ludovici, 2013).

La estructura del protocolo CoAP se puede entender como una estructura de dos niveles (ver figura 25) por un lado una capa que establece una interfaz de intercambio de mensajes con el nivel inferior de transporte UDP (Messages), y por otro lado una capa superior que establece una interfaz de peticiones y respuestas con la aplicación (Requests/Responses).

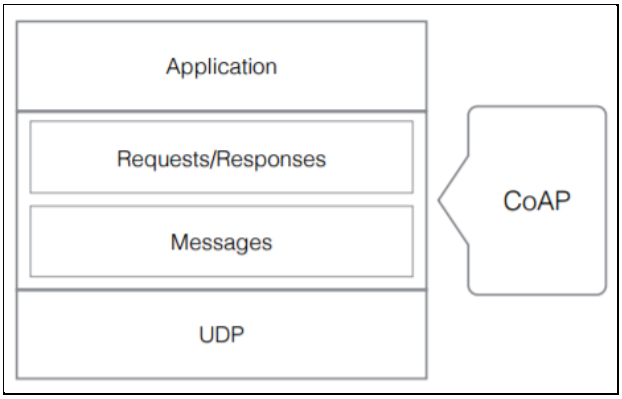

Figura 26 - Modelos en Capas del Protocolo CoAP

Como se discutió en la sección 13.5 las aplicaciones están constituidas por componentes, en este caso, La librería TinyCoAP está constituida por una serie de componentes que trabajan principalmente en las dos capas centrales (Messages y Requests/Responses).

Iniciaremos el análisis con la **Capa de Mensajes (**Message**)** que está compuesta por tres sub-componentes:

- CoapPDU: Esta componente es la encargada de proporcionar las interfaces para crear, leer y escribir los paquetes CoaAP que serán enviado a la red.
- CoapOption: Es una componente que ofrece las interfaces para configurar las opciones de los paquetes que son enviados o recibidos en la red.
- CoapList: Es la componente que permite administra una estructura de datos (lista enlazada) que almacena los mensajes.

Estas tres componentes están enlazadas entre ellas permitiendo que CoapPDU pueda tener acceso a las opciones de los paquetes, la componente que enlaza a estas tres sub-componentes se denomina TinyOSPoolC, esta componente tiene como tarea establecer en tiempo

de compilación los tamaños y la cantidad de paquetes soportado por el nodo (en la siguiente sección se estudiaran ejemplos de estas implementaciones).

Por otro lado, la **Capa de Consulta/Repuesta** (Requests/Responses) está constituida por dos componentes denominados CoapServer y CoapClient.

- CoapServer: Componente Proporciona al nodo las funciones y características para funcionar como servidor compitiendo algún recurso (Responses).
- CoapClient: Componente que permiten implementar funciones de cliente en un nodo (Requests).

Como se estudió en la sección 13 las componentes tienen por objetivo simplificar la implementación de aplicaciones y se exhiben implementaciones (funciones o métodos) que permiten interactuar con otros componentes.

A continuación, estudiaremos ejemplos funcionales equivalentes a los estudiados en ContikiOS que nos permita tener una comparación clara de las dos formas de construir aplicaciones.

Implementación de CoAP.

Antes de iniciar el estudio es importante mencionar que la versión de TinyOS que se utiliza en este trabajo es la 2.1.1.

Para iniciar el trabajo con CoAP en TinyOS el primer paso es definir las directivas de compilación que están disponibles en el Makefile (apps/TinyCoAPServer/Makefile), estas permiten definir una serie de parámetros de las aplicaciones que hacen optimizar el comportamiento de los dispositivos.

```
1 CFLAGS += -DMAX_PAYLOAD
2 CFLAGS += -DMAX_OPT_DATA
3 CFLAGS += -DMAX_OPT
4 CFLAGS += -DMAX_COAP_CONNECTIONS
5 CFLAGS += -DMAX_PDUS
6 CFLAGS += -DWAITING_LIST
7 CFLAGS += -DMAX_PACKET_LEN
```

Estas directivas permiten administrar la operación y las capacidades de una red básica, su descripción es las siguiente:

- DMAX_PAYLOAD: Establece el número máximo de carga útil que puede tener el paquete.
- DMAX_OPT_DATA: Establece el máximo largo de las Opciones del protocolo CoAP.
- DMAX_OPT: Establece el máximo número de opciones que se pueden configurar.

- DMAX_COAP_CONNECTIONS: Establece el número de conexiones que la estructura coap_connection_t puede almacenar.
- DMAX_PDUS: Establece el máximo número de CoAP PDU que pueden ser almacenados.
- DWAITING_LIST: Establece el tamaño de la cola de espera.
- DMAX_PACKET_LEN: Establece el máximo tamaño del paquete (encabezado más las opciones y la carga Útil).
- DMAX_URI_LEN: Establece el máximo tamaño de los recursos disponibles (URI).

Dos observaciones imponentes referente a las directivas de compilación primero el listado antes mencionado no es exhaustivo, se puede acceder a un listado completo en este este artículo que representa la columna vertebral de este análisis (Pol Moreno, 2013), en este trabajo de tesis solo se presentan las más relevantes y permiten iniciar un trabajo con sensores sin mayor complejidad. La segunda observación tiene que ver con el alcance de estas directivas, es importante tener claro que las directivas descritas son para equipos que actúan como cliente, así como también para aquellos dispositivos que actúan como servidores.

Para ejemplificar la facilidad que significa publicar un recurso en una red de sensores basada en TinyOS a continuación se presentan las configuraciones y algoritmos de dos equipos actuando uno como cliente y otro como servidor análogamente a como se realizó con ContikiOS en la sección anterior.

Codificación lado del Servidor

Para la construcción de aplicaciones con TinyOS en la mayoría de los casos se define un recurso a compartir en la red sobre un dispositivo que actúa como servidor, se puede definir un listado de recursos los que estarán disponibles para que los clientes se puedan conectar y consumir dichos recursos (URI por sus siglas en ingles).

```
typedef struct key_uri {
    uint8_t key;         /* Parametrized resource */
    uint8_t uri[MAX_URI_LEN]; /* URI of the resource */
    uint8_t len;         /* Length of the URI*/
    uint8_t rt[10];           /* Resource type attribute */
    uint8_t iff[10];     /* Interface Description 'if' */
    uint8_t sz;               /* Maximum size estimate attribute */
    uint8_t ct;               /* Content-format attribute*/
    uint8_t is_observe; /* Observe attribute */
  } key_uri_t;
```

Tipos de dato que almacena los recursos (coap_resource.h)

Para visualizar la relación que existe entre las estructuras de datos y la forma de definir los recursos es que en el listado anterior se muestra la definición de un tipo de datos que permitirá almacenar una lista de recursos, estos recursos se definen de la siguiente forma.

```
 1 #ifndef _COAP_RESOURCES_H_
 2 #define _COAP_RESOURCES_H_
 3 #include <coap_resource.h>
 4 #include <coap_general.h>
 5 #include <coap_pdu.h>

 6 /* Definicion de recursos que el servidor tiene disponibles*/
 7 enum {
 8     Clave_Ejemplo = 1,
 9 };
10
11 /* Definicion de recursos */
12 key_uri_t urikey_map[1] = {
13   {Clave_Ejemplo, "URI_del_Recurso", sizeof("URI_del_Re-
curso"),"Tipo Atributo", "Descripcion",
14     MAX_PAYLOAD, COAP_MEDIATYPE_TEXT_PLAIN, 1},
15};
16 #endif
```

En la línea 14 se define un recurso de ejemplo con todos los campos que exige la estructura de datos antes definida, en este caso y por motivos de simplicidad solo se define uno, pero la cantidad se puede ampliar dependiendo solamente de las capacidades del dispositivo y las necesidades de la Red.

De acuerdo al modelo de programación de TinyOS estudiado en apartados anteriores debemos crear el archivo EjemploResourceP.nc donde se definen las interfaces que serán utilizadas, en el siguiente listado se ejemplifican las definiciones de las interfaces utilizadas a partir de las componentes antes descritas.

```
 1 #include <coap_pdu.h>
 2 #include <coap_general.h>
 3 #include <IPDispatch.h>
 4 #include <lib6lowpan.h>
 5 #include <ip.h>
 6 #include "string.h"

 7 module EjemploResourceP {
 8     provides interface CoapResource;
 9     uses {
10         interface CoapList as ObserversCtrList;
11         interface CoapPdu;
12         interface Pool<coap_observe_t> as addrPool;
13         interface Timer<TMilli> as Timer0;
14         interface Random;
15  }
```

En el listado anterior línea 10 a 14 se establece las interfaces que se utilizaran asociadas a las componentes entes mencionadas, todas ellas están definidas en la librería.

```
 1implementation {
 2
 3          coap_list_t list;
 4          uint8_t obsvalue = 0, media [sizeof(uint8_t)];
 5          int obs_list = 0;
 6          /* Initialize the Message ID and Token values*/
 7          uint16_t token_gen = 1;
 8          uint16_t id_gen = 1;
 9 command void CoapResource.handle(coap_pdu_t *request, int send_sep
arate) {
10        coap_list_index_t *li;
11        char tmp[3];
12        uint8_t tk_value[MAX_OPT_DATA];
13        uint16_t len = 0;
14        /* ... */
15        request->hdr.type = COAP_MESSAGE_ACK;
16          request->hdr.code = COAP_RESPONSE_205;
17          media[0] = COAP_MEDIATYPE_TEXT_PLAIN;
18          call CoapPdu.insertOption(request, COAP_OPTION_CONTENT_FOR-
MAT, media, sizeof(uint8_t));
19          if (len != 0)
20                call CoapPdu.insertOption(request, COAP_OPTION_TOKEN,
tk_value, len);
21          memcpy(request->payload, "Esto es Ejemplo\0", strlen("Esto
es Ejemplo\0"));
22          request->payload_len = strlen("Esto es Ejemplo\0");
23          signal CoapResource.isDone(request, 0);
                }
}
```

La siguiente sección que se debe especificar es la implementación de cada una de las acciones que se pueden realizar el sensor, en este caso se ejemplifica con una acción, (las demás fueron eliminadas línea 14) esta acción permite realizar el envío de las respuestas al cliente (línea 9), en la línea 15 a 21 se ejemplifica algo muy importante y simple de realizar, el manejo de las opciones de los paquetes. Para finalizar en la línea 23 se encuentra la instrucción que envía el mensaje a la red.

```
 1 #include "resources/resources.h"

 2 module CoapEjemploServerP {
 3  uses {
 4    interface Boot;
 5    interface CoapServer;
 6    interface CoapPdu;
 7  }
 8  } implementation {
 9     uint32_t rate = 10048;

10   event void Boot.booted() {

11   }

12    event void CoapServer.booted() {
13        call CoapServer.init(COAP_DEFAULT_PORT, urikey_map, 1);
14        call CoapServer.state(rate, Clave_Ejemplo);
    }
}
```

En la Listado anterior se aprecia un extracto del archivo principal de la aplicación con el objetivo de mostrar como son las llamadas a los componentes (líneas 13 y 14) que permiten implementar un servidor en un sensor, en la línea 14 se inicializa el servidor entregándole como parámetro el puerto del servidor y la estructura donde se almacena el listado de recursos disponibles y el ultimo parámetro define la utilización de Multicast.

Lado del Cliente

En el caso que se quiera construir una aplicación en que el nodo funcione como cliente las librerías e interfaces serían las siguientes

```
 1 #include <IPDispatch.h>
 2 #include <lib6lowpan.h>
 3 #include <ip.h>
 4 #include <string.h>
 5 #include <stdio.h>

 6 #include "coap_pdu.h"
 7 #include "coap_general.h"
 8 #include "coap_option.h"
 9 #include "printf.h"

10 module ClientCoapP {
11   uses {
12    interface Boot;
13    interface CoapClient;
14    interface CoapPdu;
15    interface Timer<TMilli> as Timer0;
16    interface Random;
17  }
```

El siguiente paso es la implementación de las funciones del cliente, la primera función es el arranque de las configuraciones del equipo, podemos ver las estructuras necesarias para la entrada la red IPv6.

```
} implementation {

      coap_pdu_t *request = NULL;
      uint16_t token_gen, id_gen;
      struct sockaddr_in6 dest;
      char uri[MAX_URI_LEN];
      uint8_t ok_send = 0;

      event void Boot.booted() {
            call Timer0.startOneShot(12400);
            /* Datos para el Token*/
            token_gen = call Random.rand16();
            id_gen = call Random.rand16();
            /* Direccion de Servidor */
            dest.sin6_port = hton16(COAP_DEFAULT_PORT);
            inet_pton6("fec0::1", &dest.sin6_addr);
}
```

En la siguiente imagen se muestra el procesamiento de un mensaje por medio de la creación de un evento (línea 42) la extracción de las opciones y las funciones que se utilizan comúnmente a partir de la librería.

```
42      event void CoapClient.messageReceived(coap_pdu_t *response){
43          coap_option_t *token, *ct;
44          uint8_t tk_buffer[MAX_OPT_DATA], uri_len = 0;
45          uint16_t len = 0;
46          char *p = NULL, *q = NULL;
47
48          if (response->hdr.code <= COAP_RESPONSE_205){
49              switch(response->hdr.code){
50
51                  case COAP_RESPONSE_205:
52                      if (call CoapPdu.getOption(response, COAP_OPTION_CONTENT_FORMAT, &ct) == SUCCESS){
53                          switch(ct->value[0]){
54                              case COAP_MEDIATYPE_TEXT_PLAIN:
55                                  printf("%s\n", response->payload);
56                                  printfflush();
57                                  break;
58                              case COAP_MEDIATYPE_APPLICATION_LINK_FORMAT:
59                                  p = &response->payload;
60                                  q = p;
61                                  if (*p == '<'){
62                                      p++;
63                                      q++;
64                                  }
65                                  while (*p != '>')
66                                      p++;
67                                  uri_len = p-q;
68                                  memcpy(uri, q, uri_len);
69                                  ok_send = 1;
70                                  call Timer0.startOneShot(1024);
71                                  break;
72                              default:
73                                  break;
74                          }
75                      }
76                      break;
77                  default:
```

Discusión Comparativa

Luego de implementar en este trabajo aplicaciones cliente servidor en los dos sistemas operativos y con las mismas funcionalidades las conclusiones desde un punto de vista técnico son:

Depuración y Pruebas: Ambos sistemas operativos presentan herramientas que persiguen la implementación de aplicaciones de forma rápida y fácil pero falta mucho por recorrer, los IDE de programación que se estudiaron ayudan al trabajo, pero la integración con los dispositivos está en sus fases iniciales o simplemente no está, por ejemplo, desarrollar un algoritmo, probarlo y depurarlo implica una serie de interacciones con la mota, esta interacción en este momento depende mucho de la disponibilidad de simuladores y de la experiencia del programador en cuestiones que tienen que ver con administradores de sistema de tal manera se probar las aplicaciones en forma separada. Si en este punto nos preguntamos ¿Cuál es el sistema que tiene más avances al momento de facilitar el trabajo de depuración y pruebas? la respuesta sería ContikiOS.

Curva de Aprendizaje: Una característica importante al momento de la elección de un lenguaje de programación o una herramienta de desarrollo es la curva de aprendizaje que tiene y en el desarrollo de este trabajo la curva de aprendizaje es mejor en el lenguaje de programación que ofrece TinyOS y se debe principalmente a la disponibilidad de documentación.

Codificación y librerías: Finalmente en la etapa de codificación una característica significativa que muchas veces hace la diferencia en la elección de un entorno de desarrollo es la forma de codificación y la disponibilidad de librerías. En los algoritmos desarrollados en este trabajo se utilizó una serie funcionalidades y características de los entornos de programación donde cada entorno ofrece un conjunto funciones y librerías relacionadas a las comunicaciones

en este caso al protocolo CoAP, ambos entornos (ContikiOS y TinyOS) cuentan con soporte para este protocolo, pero es ContikiOS el que ofrece mejores interfaces de programación y esto se debe principalmente a que cuenta con librerías integradas (fácil integración) y con un nivel de implementación del protocolo que permite utilizar la gran mayoría de características de CoAP, por otro lado TinyOS al momento en que se desarrolla este trabajo la versión del Protocolo que implementa es la TinyOS CoAP (-13) y está basada en una implementación de C para CoAP , pero si pasmos por alto el nivel de desarrollo de la librería ambos entornos ofrecen las mismas funciones la diferencias radical la presenta ContikiOS al permitir probar la aplicación en dispositivos POSIX, por ejemplo un pc portátil esto permite generar un ambiente de prueba mucho más amigable que los entornos con simuladores incluso con mejores prestaciones que montarlos en la misma red.

Una última observación a las plataformas de desarrollo en general y tiene relación con la complejidad que presentan las plataformas, en estos momentos para aquellos usuarios y/o programadores que desean participar de estas tecnologías, el esfuerzo inicial por configurar el equipamiento, luego comprender el sistema operativo y por último el levantamiento de la red hace que no sea una tarea fácil, requiere conocimientos previos sólidos en sistemas operativos y comunicaciones.

Este apartado fue construido principalmente con información obtenida de (Waher, 2015)

13. Propuesta para la Construcción de Aplicaciones

13.1. Descripción General

En esta sección del documento se realizará una propuesta que permita la construcción de aplicaciones para redes de sensores, si bien los estudios realizados abarcaron diferentes alternativas en este apartado se abordan aquellos elementos que fueron seleccionados como las mejores opciones por cada uno de los estudios y/o análisis.

El éxito de una tecnología, un modelo o una determinada solución depende de muchos factores como por ejemplo la facilidad en su utilización, los costos de despliegue, la disponibilidad, la documentación disponible entre otros. En el caso de las redes de sensores y de esta tesis en particular uno de los factores principales que se abordan tiene que ver con la construcción de soluciones de software que permitan sacar el mejor provecho a la implantación de una red, es decir, podamos construir aplicaciones útiles de forma rápida y sin desgaste de recursos. En este sentido la propuesta presenta una forma de construir las aplicaciones, pero no pretende ser la única solución, por el contrario, es un enfoque adaptable que permite la búsqueda y elección de un conjunto de tareas, tecnologías y herramientas que permitan optimizar el proceso.

En una red de sensores una vez que se resuelve el tema del transporte de los datos, es decir, podemos llegar con los valores censados desde el propio sensor a la estación recolectora tenemos gran parte del problema resuelto, por tanto, proporcionar las herramientas necesarias para la construcción de aplicaciones que logren este objetivo sin un desgaste de recursos es una de las tareas de esta propuesta. Bajo esa premisa se presentará en primera instancia una estructura general de la propuesta para luego desglosar por etapas y profundizar cada una de ellas (García, E. 2009).

13.2. Estructura General de la Propuesta

Para la definir un marco de trabajo que permita el desarrollo de aplicaciones se requiere un conjunto de técnicas, procedimientos, herramientas y soportes documentales que apoyen las actividades correctamente ordenadas, con este objetivo se presenta esta metodología agrupando los elementos de acuerdo a como fueron abordado en el presente estudio.

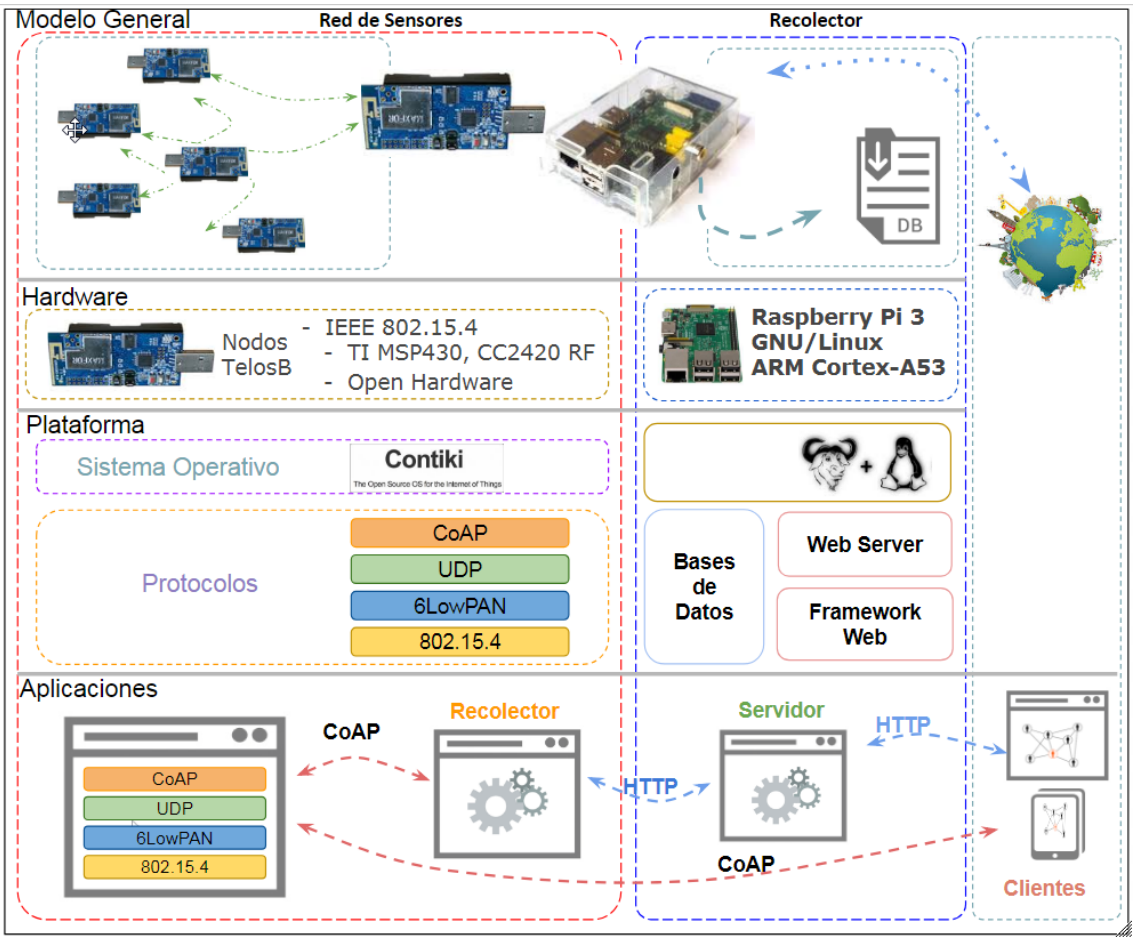

Figura 27 - Diagrama Aplicaciones basadas en CoAP (Con Imágenes libres iconfinder.com))

La construcción de aplicaciones para redes de sensores involucra una serie de elementos entre ellos tecnologías, protocolos y aplicaciones como se aprecia en la figura anterior, las que deben ser consideradas al momento de planificar una aplicación, describiremos cada sección de tal forma que: podamos visualizar un panorama general y uno particular que permita abordar la construcción de aplicaciones desde una perspectiva que simplifique la comprensión.

13.3. Etapas en la construcción de aplicaciones:

Cuando se enfrenta por primera vez la implementación de una red de sensores es complejo visualizar el diseño general que satisfaga los requerimientos, y si eso se logra tener claro, el siguiente problema es el trasfondo técnico que tiene cada una de las componentes, a continuación, se describen las etapas del modelo general en orden y relacionando cada una con los correspondientes estudios que se realizaron en este trabajo.

En la construcción de aplicaciones para redes de sensores se requiere adoptar un esquema de desarrollo que permita sobre llevar el ciclo de vida de la aplicación y en este sentido la propuesta es adoptar un modelo incremental, es decir, desarrollar un esquema básico de funcionamiento para luego enriquecerlo con más funcionalidades, en la primera instancias se busca la conectividad, es decir, la red mantenga su conexión en el tiempo y permita la transmisión de

los datos, luego de eso los esfuerzos giran la mirada hacia la estación recolectora primero mirando su capacidad y luego el rendimiento. Para terminar los esfuerzos se centran en la forma en que se consumen los datos por parte del cliente. A continuación, analizaremos cada una de esas etapas.

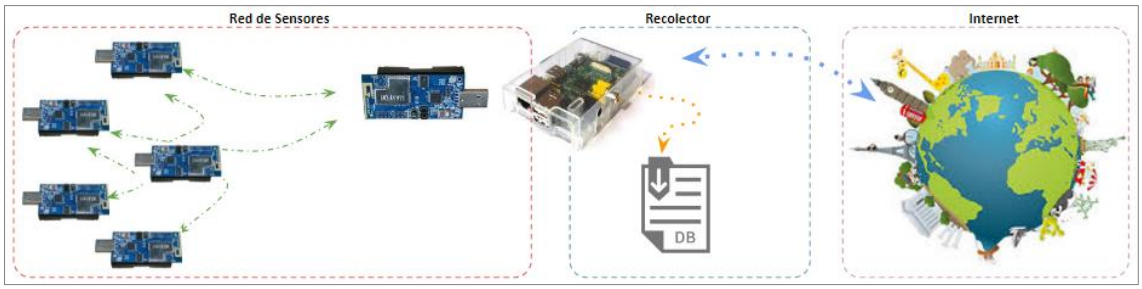

Figura 28 - Modelos General Simplificado de una red de sensores

La primera etapa que denominaremos estudios del problema, consiste principalmente en abordar los requerimientos definiendo los parámetros que se desean medir en la red y cómo es que estos se desean medir, en este documento en el apartado referente al hardware (Sección 1) se definen las características y las capacidades que tiene las motas TelosB, además de las posibilidades de expansión por medio de sus interfaces, en esta sección también se fundamenta las razones por la cuales se proponen estos dispositivos en particular.

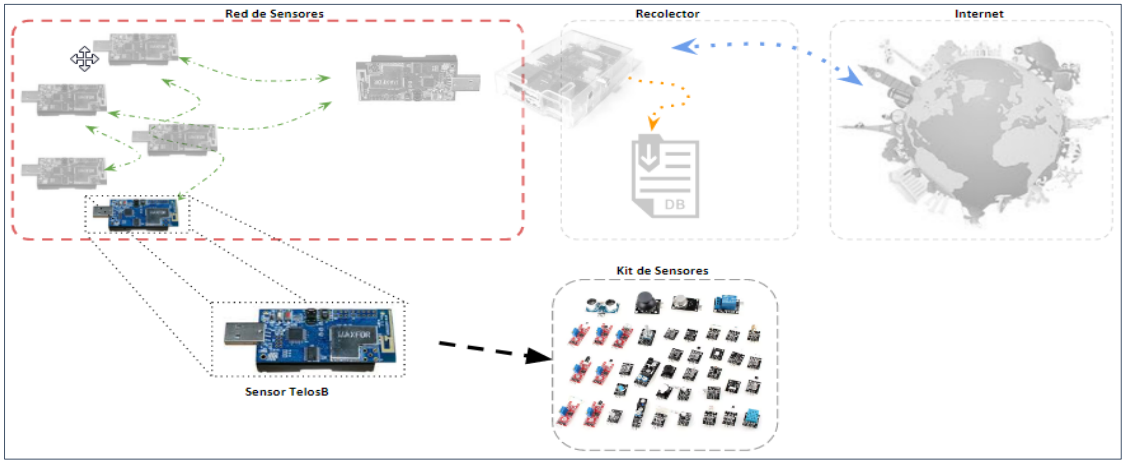

Figura 29 - Extensiones para TelosB

La siguiente etapa que se aborda, tiene que ver principalmente con los datos que se requiere censar y el comportamiento de la red o las formas en que se recolectaron los datos, por ejemplo, definir cuanto tiempo estar apagado un determinado sensor o por cuanto tiempo estará a la espera de una transmisión. Para apoyar esta tarea el capítulo 2 estudia pormenorizado el lenguaje de programación y las estructuras que deben tener las aplicaciones.

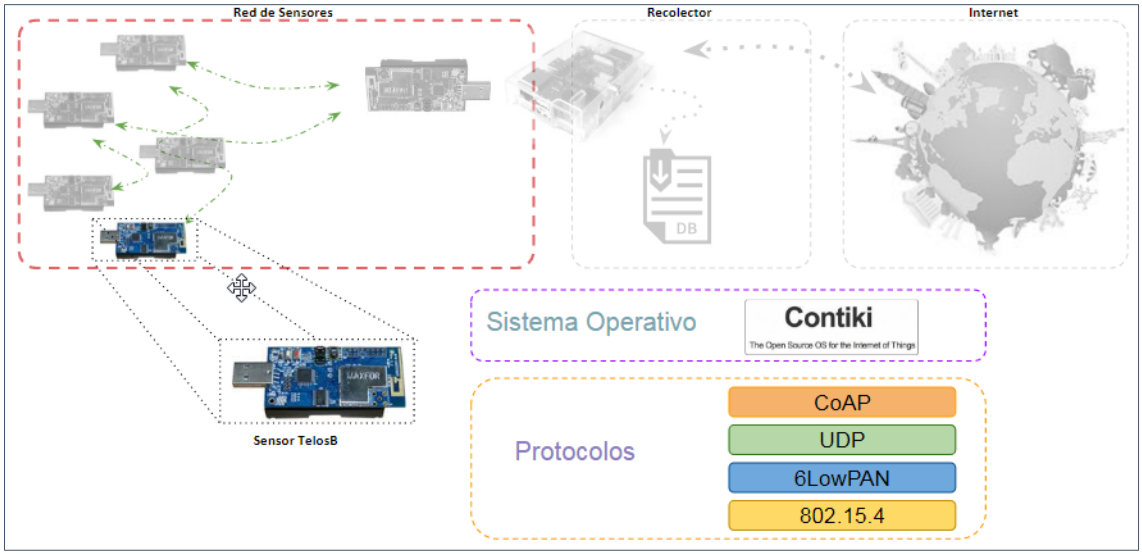

Figura 30 - Redes sensores y protocolos que intervienen

Otras de las etapas para la construcción de la red de sensores es proyectar cual será el comportamiento de la red, esto contempla definir cuál es la estrategia de recolección, por ejemplo: definir si los sensores automáticamente enviaran los datos o será la estación receptora la que solicitara los datos, y aquí juega un papel importante los conocimientos de NecC y con ello reprogramar los algoritmos definidos en el Sección 3.

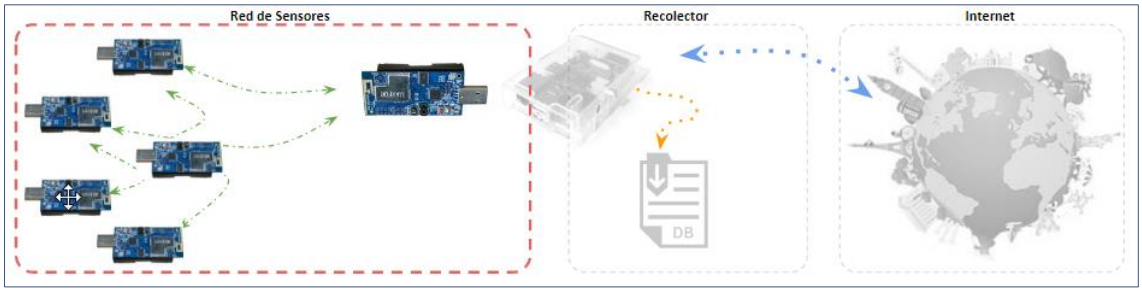

Figura 31 - Red de Sensores

El siguiente paso sistema es el nodo recolector, es decir, tenemos que definir toda la estrategia de almacenamiento de datos, por lo general la estación recolectora hace uso de una base de datos externa para el almacenamiento de los datos.

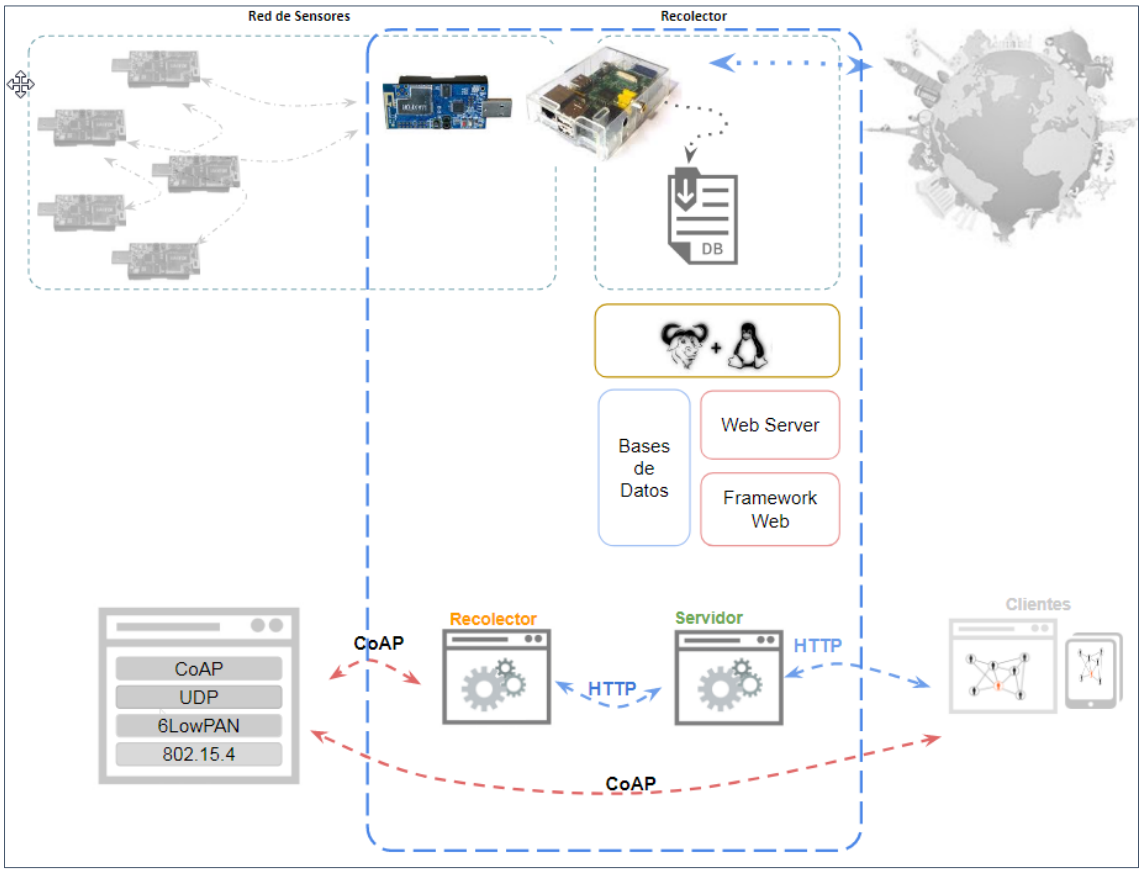

Figura 32 - Red sensores y aplicaciones Clientes

La última etapa tiene que ver con los clientes o aplicaciones que consumen la información, estos no significan mayor problema, porque las prácticas de hoy en día hacen que el despliegue se realice vía web (http) entonces no importan la plataforma o el cliente que requiera la información, siempre estará disponible. También existe la opción que los clientes puedan acceder directamente a los sensores y en este caso son los clientes los que procesan los datos solicitados a los nodos de la red esto se estudia en detalle en la sección 4.

14. Problema Integrador

14.1. Definición del Problema

Para efectos de demostrar la propuesta presentada en este proyecto, se implementará un escenario de red 6lowpan para la monitorización de la salud estructural de puentes, este prototipo forma parte de un sistema de gestión de puentes o BMS (Bridge Management System) que será alimentado con los datos recolectados por la red de sensores.

De esta forma, a través de implementación de un escenario de red 6lowpan se busca demostrar la utilización de la pila de protocolos necesarios para establecer la comunicación IP entre dispositivos y así recolectar información a partir de múltiples puntos de monitoreo en un puente de la red pública vial.

En términos generales, la red estará compuesta por un conjunto de sensores distribuidos estratégicamente sobre un puente, teniendo como principal tarea la centralización de los datos en una estación de monitoreo ubicada fuera del puente.

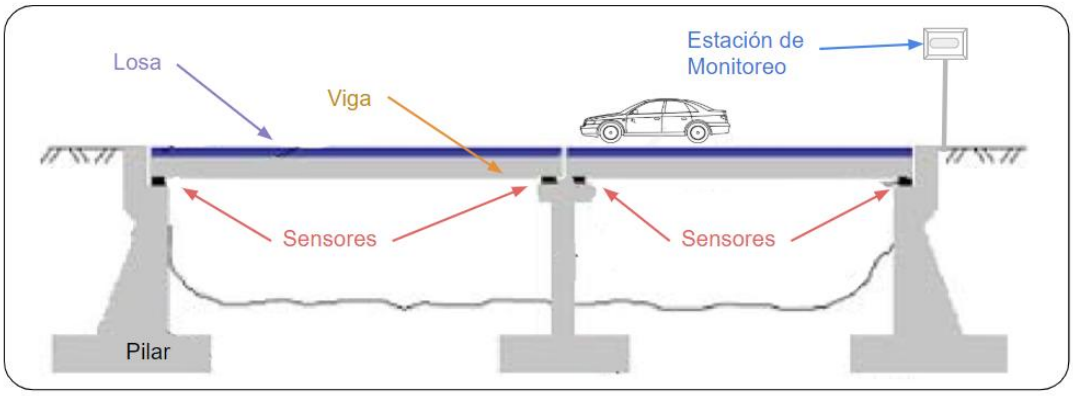

Figura 33 - Diagrama General del Escenario del Problema

Los detalles sobre el número, la ubicación de los sensores, las características técnicas de los nodos sensores (Ultrasonic module HC-SR04 distance measuring transducer sensor), la información que se obtiene a partir de los datos recolectados y características técnicas de la implementación físicas de la solución escapan a los objetivos de este documento.

Al efectuar el análisis del problema, tenemos una estructura de red que se acomoda perfectamente a las características del modelo planteado en este trabajo, por lo tanto, podemos abordar la solución con los elementos que fueron analizados.

14.2. Diseño Solución

Al proyectar la estructura de red propuesta en este trabajo sobre la problemática definida tenemos una relación como la que se describe en la figura siguiente:

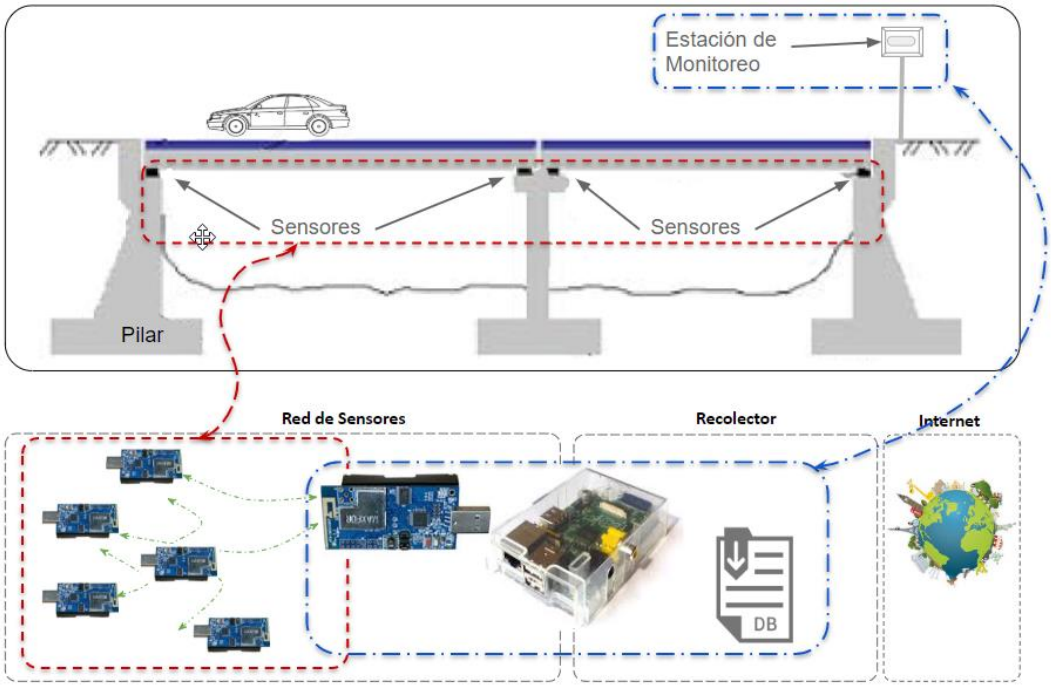

Figura 34 - Arquitectura general red de sesnores del Sistema Gestor de Puentes

14.3. Hardware

Para este ejemplo se utilizará una Mota TelosB descrita en secciones anteriores de este documento, en este caso se utilizan los pines de expansión para conectar sensores genéricos, para efecto de este ejemplo solo se utiliza un dispositivo de tención el que entrega un valor entero, el valor que tiene será transportado a la estación de monitoreo para su almacenamiento (centralización).

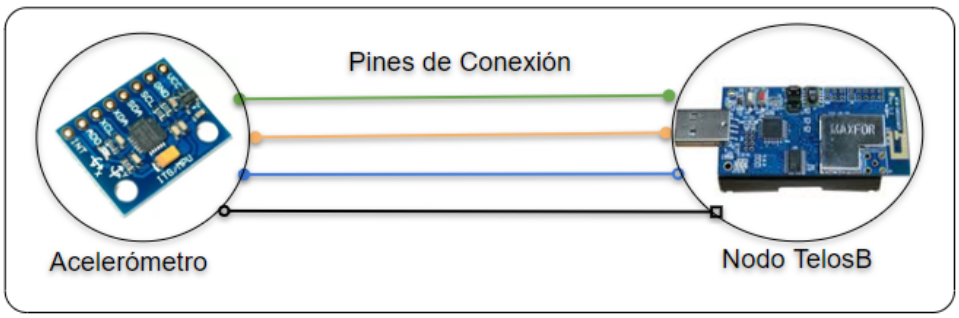

Figura 35 - Diagrama nodo Hoja

14.4. Construcción de Cliente.

Se requiere construir una aplicación que permita hacer funcionar la red de sensores, con los siguientes requerimientos:

- Recolección de información (en caso de nodo hoja)
- El reenvió de información (nodo router)

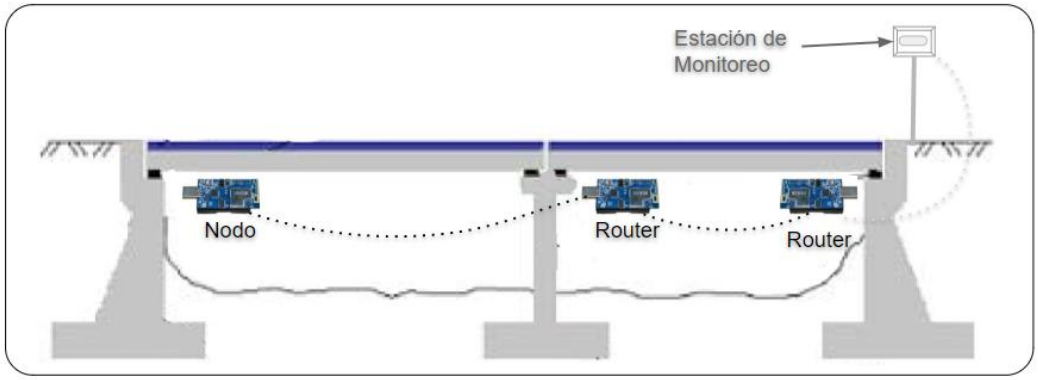

Figura 36 - Nodo Extremo

14.5. Codificación Cliente

```
#include "contiki.h"
#include "lib/random.h"
#include "sys/ctimer.h"
#include "net/uip.h"
#include "net/uip-ds6.h"
#include "net/uip-udp-packet.h"
#include "dev/light-sensor.h"
#include "dev/sht11-sensor.h"
#include "dev/sen637.h"
#include <stdio.h>
#include <stdint.h>
#define CLIENTE_PUERTO 8765
#define SERVER_PUERTO 5678

//CONFIGURACION RELOJ
#define PERIODO 10
#define INI_INTERVAL          (5 * CLOCK_SECOND)
#define ENVIO_INTERVAL        (PERIODO * CLOCK_SECOND)
#define ENVIO_TIEMPO          (random_rand() % (ENVIO_INTERVAL))
#define MAX_PAYLOAD_LEN       30

//ESTRUCTURAS DE CONEXION
static struct uip_udp_conn *client_conn;
static uip_ipaddr_t server_ipaddr;
```

Sección de código Cliente – Definiciones Generales.

```
//ESTRUCTURAS DE CONEXION
static struct uip_udp_conn *client_conn;
static uip_ipaddr_t server_ipaddr;

PROCESS(udp_client_process, "Cliente_demo");
AUTOSTART_PROCESSES(&udp_client_process);

static void send_packet(void *ptr)
{
      char buf[MAX_PAYLOAD_LEN];
      //LECTURA DE SENSORES
      SENSORS_ACTIVATE(sen637_sensor);
      int Valor_Fisico = sen637_sensor.value(SENSOR_VAL);
      SENSORS_DEACTIVATE(sen637_sensor);
      sprintf(buf, "Val_%d",Valor_Fisico);
      //ENVIO DE DATOS (CONEXION UDP, BUFER, TAMAÑO BUFER, DIR. SER-
VER, PUERTO SERVER)
      uip_udp_packet_sendto(client_conn, buf, strlen(buf),
&server_ipaddr, UIP_HTONS(SERVER_PUERTO));
}

PROCESS_THREAD(udp_client_process, ev, data)
{
      //RELOJES
      static struct etimer PERIODOic;
      static struct ctimer backoff_timer;
      PROCESS_BEGIN();
      uip_ip6addr(&server_ipaddr, 0xaaaa, 0, 0, 0, 0, 0x00ff, 0xfe00,
1);
      client_conn = udp_new(NULL, UIP_HTONS(SERVER_PUERTO), NULL);
      udp_bind(client_conn, UIP_HTONS(CLIENTE_PUERTO));
      etimer_set(&PERIODOic, ENVIO_INTERVAL);
      while(1) {
            PROCESS_YIELD();
            if(etimer_expired(&PERIODOic)) {
                  etimer_reset(&PERIODOic);
                  ctimer_set(&backoff_timer, ENVIO_TIEMPO,
send_packet, NULL);
            }
      }
      PROCESS_END();
}
```

Sección Código Cliente – Evento Principal.

14.6. Construcción de Servidor

Los requerimientos para el o los nodos servidores es la recolección de los datos enviados desde los nodos ubicados en el Puente:

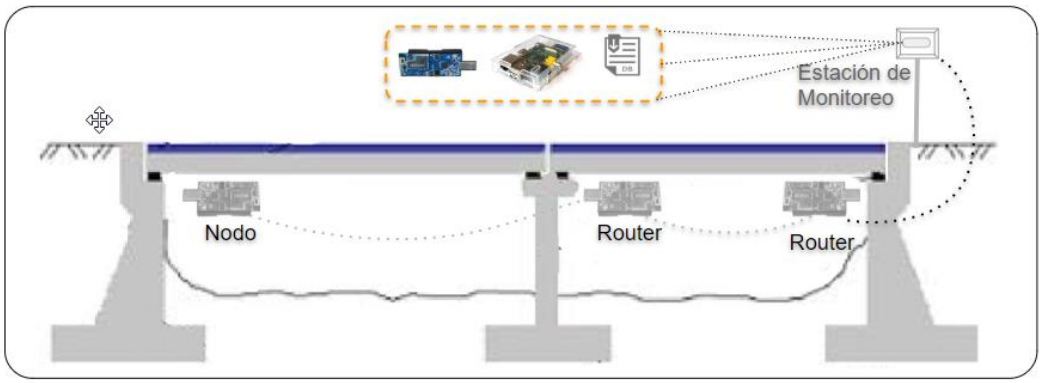

Figura 37 - Arquitectura de Solución para la Estación de Monitoreo

14.7. Codificación Servidor

```
#include <stdio.h>
#include "contiki.h"
#include "contiki-net.h"

#include "erbium.h"
#include "er-coap-13.h"

#include "node-id.h"
#include "dev/sht11-sensor.h"
```

Sección Código Servidor – Definiciones Generales

```
void readings_handler(void* request, void* response, uint8_t *buf-
fer, uint16_t preferred_size, int32_t *offset)
{
      snprintf((char *)buffer, REST_MAX_CHUNK_SIZE, "%u;%i", node_id,
((sht11_sensor.value(SHT11_SENSOR_TEMP) / 10) - 396) / 10);
    REST.set_response_payload(response, (uint8_t *)buffer, str-
len((char *)buffer));
}

/* ---------------------------------------------------------------- */

PROCESS(rest_server, "Erbium Server");
AUTOSTART_PROCESSES(&rest_server);

PROCESS_THREAD(rest_server, ev, data)
{
      PROCESS_BEGIN();

      SENSORS_ACTIVATE(sht11_sensor);

      rest_init_engine();
```

```
        rest_activate_resource(&resource_readings);

        PROCESS_END();
}
```

Sección Código Servidor – Evento Principal

14.8. Base de Datos de Recolección

En esta sección se presentará las secciones de codificación mas importantes de la aplicación de recolección en el dispositivo Raspberry.

```
require './subproc'
require 'open-uri'
require 'nokogiri'
require 'coap'
require 'mysql2'
require 'yaml'
```

Recolector - Encabezados

Codificación que define las credenciales para la conexión al base de datos.

```
retries = 3
client = nil
while retries != 0 and client.nil?
      tsputs "Conectando a base de datos..."
      begin
            client = Mysql2::Client.new(
                  :host          => db_host,
                  :username => db_user,
                  :password => db_password,
                  :database => db_name)
      rescue Mysql2::Error => e
            tsputs e
            tsputs "Reintentando..."
      end
      retries -= 1
      sleep 1
end
if client.nil?
      tsputs "No se pudo establecer la conexión a la base de datos "+
      "por favor revisa la configuración y ejecuta esto de nuevo, sa-
liendo..."
      shut_down
      exit 1
end
```

Recolector - Sección de conexión a base de datos

En la siguiente sección se configura el tunel de conexión entre internet y la red 6lowpan del puente, con la ayuda de la aplicación tunsli6 que fue descrita en apartados anteriores.

```
tsputs "Configurando adaptador tunel..."
# "sudo /home/pi/Argiot/contiki/tools/tunslip6 -a 127.0.0.1
aaaa::1/64"
# "sudo /home/pi/Argiot/contiki/tools/tunslip6 aaaa::1/64"
cmd = "sudo /home/pi/Argiot/contiki/tools/tunslip6 bbbb::1/64"
$tun_error = false
$tunnel_thread = Thread.new do
      Utils::Subprocess.new cmd do |stdout, stderr, thread|
        puts stdout
        puts stderr if !stderr.nil?
        #tsputs "pid: #{thread.pid}" # => 12345

        if $catch_ip
            $router_ip = stdout
            $catch_ip = false
        end
        $catch_ip = true if stdout == "Server IPv6 addresses:\n"
        if !stderr.nil?
            $tun_error = true if stderr.include? "Device or resource
busy"
        end
      end
end
```

Recolector - Configuración del Tunes a la red 6LowPan

```
while true
      tsputs "#{prefix}Esperando #{read_wait} segundos antes de
leer..."
      sleep read_wait

      # Leer HTML
      begin
            tsprint "#{prefix}Conectando a enrutador de borde...
\r"
            sleep 1
            html_data = open("http://[#{$router_ip}]/").read
            tsprint html_data
            tsprint "#{prefix}Haciendo web crawl a enrutador de
borde...                \r"
            sleep 1
            nokogiri_object = Nokogiri::HTML(html_data)
            li_elements = nokogiri_object.xpath("//li")

            # identificación de IP
            if li_elements.count > 0
                  tsprint "#{prefix}Armando lista de servidores
CoAP...                  \r"
                  sleep 1
                  coap_nodes_ips = Array.new
                  li_elements.each do |li_element|
                        coap_nodes_ips.push(li_ele-
ment.text.split('/').first)
                  end
```

Recolector - Lectura de los IP de los dispositivos presentes en la Red 6lowpan

```
            tsprint "#{prefix}Ejecutando consultas CoAP...
\r"
                  sleep 1
                  results = Array.new

                  coap_nodes_ips.each do |coap_node|
                        begin
                              pre_hash_result
=CoAP::Client.new.get_by_uri(

      "coap://[#{coap_node}]:5683/readings").payload
                              node_id = pre_hash_re-
sult.split(';').first
                              value = pre_hash_result.split(';').last
                              hashed_result = {'node_id' => node_id,
'value' => value}
                              results.push hashed_result
                        rescue RuntimeError => e
                              tsputs "#{e}\n#{prefix}CoAP Timeout en
servidor: #{coap_node}"
                        end
                  end
                  result_count_pre_filter = results.count
```

Recolector - Lectura de Datos mediaste CoAP desde Servidores (nodos 6lowPan)

```
#Guardar resultados en la DB
                  if results.count > 0
                        tsprint "#{prefix}Insertando #{results.count}
lecturas en base de datos '#{db_name}'...   \r"
                  else
                        tsprint "#{prefix}No hay datos para insertar
\r"
                  end
                  sleep 1
                  results.each do |result|
                        begin
                              client.query("INSERT INTO readings
(node_id, value, date)"+
                              " VALUES ('#{result['node_id']}',
'#{result['value']}', "+
                              "NOW());")
                        rescue Mysql2::Error => mysql_exception
                              tsputs "SQL Error: #{mysql_exception}"
                        end
                  end
```

Recolector - Implementación del almacenamiento en la base de datos.

14.9. Implementación de CoAP

Para la implementación del protocolo CoAP se utiliza la implementación del Motor Erbium REST disponible en el conjunto de aplicaciones de Contiki.

```
const struct rest_implementation coap_rest_implementation = {
 coap_init_engine,
 coap_set_service_callback,
 coap_get_header_uri_path,

}
```

Recolector - Implementación de principal estructura

Se requiere declarar un recurso periódico, por ejemplo, sondear un sensor y publicar un cambio en los valores en los clientes suscritos, deberíamos usar:

```
#define PERIODIC_RESOURCE(name, attributes, get_handler, post_han-
dler,
put_handler, delete_handler, period, periodic_handler) \
 periodic_resource_t periodic_##name; \
 resource_t name = { NULL, NULL, IS_OBSERVABLE | IS_PERIODIC, at-
tributes,
get_handler, post_handler, put_handler, delete_handler, { .periodic
=
&periodic_##name } }; \
 periodic_resource_t periodic_##name = { NULL, &name, period, { { 0
} },
periodic_handler };
```

Recolector - Implementación de principal estructura

14.10. Ejemplos de Trafico

Cuando el cliente realiza una petición, mediante una opción que indica el método a utilizar para solicitar un recurso (identificado por una URI), el servidor envía una respuesta con un código de respuesta que puede incluir una representación de dicho recurso, esta comunicación es asíncrona a través de transporte UDP. Esto se realiza utilizando una capa de mensajes que soporta una fiabilidad opcional, los mensajes pueden ser: Confirmable (CON), No-Confirmable (NON), sentimientos o reconocimiento (ACK) y Reset (RST) , estos mensajes se encuentran en la cabecera CoAP (esto se presentó con mayor detalle en capítulos anteriores).

Para hacer las pruebas de campo de los algoritmos generados, tanto para el servidor como para el cliente, se cargaron en el simulador COOJA tres nodos simulando los sensores presentes sobre el puente y un cuarto nodo que hace de estación recolectora, como se muestra a continuación:

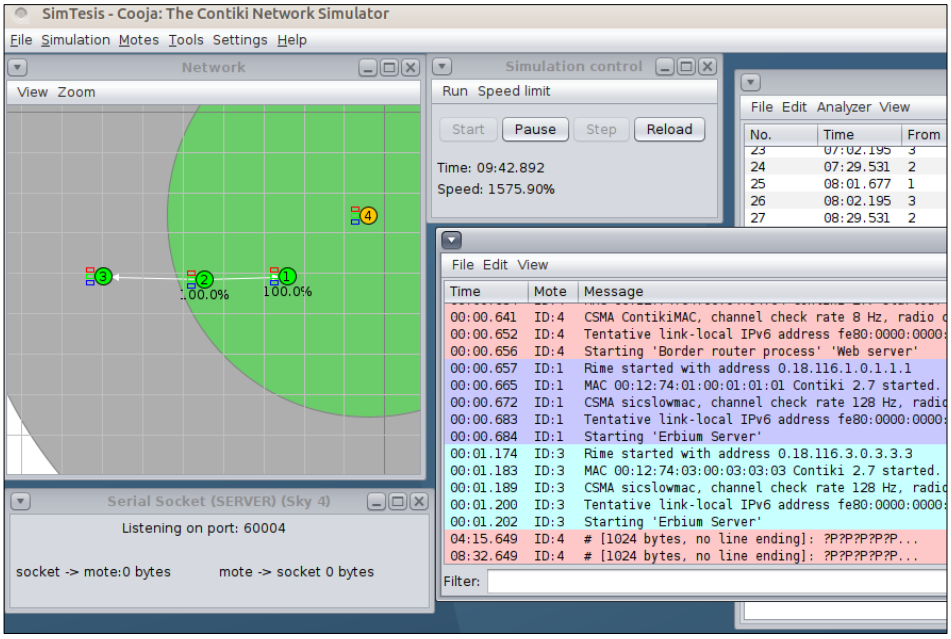

Figura 38 - Implementación de principal estructura

Se analizará el conjunto de frames que se generan al momento de efectuar una comunicación atreves de la red 6lowpan, en este caso, entre un nodo sensor sobre el puente y la estación recolectora. En la imagen siguiente se muestra seis (desde el 47 al 52) frames CoAP que viajan sobre una red 6lowpan.

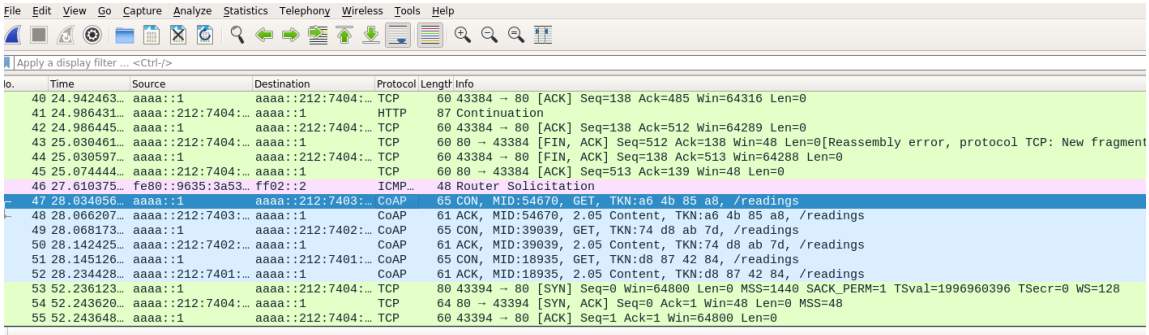

Figura 39 - Listado de Frames CoAP

El modelo de mensajería CoAP se basa en el intercambio de mensajes a través de UDP entre puntos finales, en este caso los sensores, usa un encabezado binario de 4 bytes que puede ser seguido de opciones binarias y una carga útil (payload), este tipo de formato utilizado en los mensajes de solicitud / respuesta. Para evitar la duplicación de mensajes y la fiabilidad cada mensaje tiene un ID de 16 bits de tamaño.

El primer frame que se analiza es el 47, que corresponde a la solicitud GET por parte de la estación recolectora

```
▸ Frame 47: 65 bytes on wire (520 bits), 65 bytes captured (520 bits) on interface 0
  Raw packet data
▾ Internet Protocol Version 6, Src: aaaa::1, Dst: aaaa::212:7403:3:303
    0110 .... = Version: 6
  ▸ .... 0000 0000 .... .... .... .... .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
    .... .... .... 1011 0101 1101 0000 1001 = Flow Label: 0xb5d09
    Payload Length: 25
    Next Header: UDP (17)
    Hop Limit: 64
    Source: aaaa::1
    Destination: aaaa::212:7403:3:303
▸ User Datagram Protocol, Src Port: 32884, Dst Port: 5683
▾ Constrained Application Protocol, Confirmable, GET, MID:54670
    01.. .... = Version: 1
    ..00 .... = Type: Confirmable (0)
    .... 0100 = Token Length: 4
    Code: GET (1)
    Message ID: 54670
    Token: a64b85a8
  ▾ Opt Name: #1: Uri-Path: readings
      Opt Desc: Type 11, Critical, Unsafe
      1011 .... = Opt Delta: 11
      .... 1000 = Opt Length: 8
      Uri-Path: readings
    [Response In: 48]
    [Uri-Path: /readings]
```

Figura 40 - Solicitud desde el cliente al nodo numero 3

La confirmación de cada mensaje es comprobada por un tiempo hasta que el destinatario envía un mensaje de asentimiento (ACK) con el mismo ID, esto se refleja en el Frame 48.

```
   47 28.034056… aaaa::1              aaaa::212:7403:3:303 CoAP    65 CON, MID:54670, GET, TKN:a6 4b 85 a8, /readings
   48 28.066207… aaaa::212:7403:3:303 aaaa::1              CoAP    61 ACK, MID:54670, 2.05 Content, TKN:a6 4b 85 a8, /readings
   49 28.068173… aaaa::1              aaaa::212:7402:2:202 CoAP    65 CON, MID:39039, GET, TKN:74 d8 ab 7d, /readings
   50 28.142425… aaaa::212:7402:2:202 aaaa::1              CoAP    61 ACK, MID:39039, 2.05 Content, TKN:74 d8 ab 7d, /readings
   51 28.145126… aaaa::1              aaaa::212:7401:1:101 CoAP    65 CON, MID:18935, GET, TKN:d8 87 42 84, /readings
   52 28.234428… aaaa::212:7401:1:101 aaaa::1              CoAP    61 ACK, MID:18935, 2.05 Content, TKN:d8 87 42 84, /readings
   53 52.236123… aaaa::1              aaaa::212:7404:4:404 TCP     80 43394 → 80 [SYN] Seq=0 Win=64800 Len=0 MSS=1440 SACK_PERM
   54 52.243620… aaaa::212:7404:4:404 aaaa::1              TCP     64 80 → 43394 [SYN, ACK] Seq=0 Ack=1 Win=48 Len=0 MSS=48
   55 52.243648… aaaa::1              aaaa::212:7404:4:404 TCP     60 43394 → 80 [ACK] Seq=1 Ack=1 Win=64800 Len=0

▸ Frame 48: 61 bytes on wire (488 bits), 61 bytes captured (488 bits) on interface 0
  Raw packet data
▾ Internet Protocol Version 6, Src: aaaa::212:7403:3:303, Dst: aaaa::1
    0110 .... = Version: 6
  ▸ .... 0000 0000 .... .... .... .... .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
    .... .... .... 0000 0000 0000 0000 0000 = Flow Label: 0x00000
    Payload Length: 21
    Next Header: UDP (17)
    Hop Limit: 63
    Source: aaaa::212:7403:3:303
    Destination: aaaa::1
▸ User Datagram Protocol, Src Port: 5683, Dst Port: 32884
▾ Constrained Application Protocol, Acknowledgement, 2.05 Content, MID:54670
    01.. .... = Version: 1
    ..10 .... = Type: Acknowledgement (2)
    .... 0100 = Token Length: 4
    Code: 2.05 Content (69)
    Message ID: 54670
    Token: a64b85a8
    End of options marker: 255
    [Request In: 47]
    [Response Time: 0.032150747 seconds]
    [Uri-Path: /readings]
  ▸ Payload: Payload Content-Format: application/octet-stream (no Content-Format), Length: 4
▾ Data (4 bytes)
    Data: 333b3234
    [Length: 4]
```

Figura 41 - Captura del Frame 48 con la repuesta del nodo tres

El contenido del mensaje es recolectado por el nodo cliente, en este caso la estación recolectora para ser almacenado en la base de datos (nodo 3; valor 24).

```
▾ Data (4 bytes)
    Data: 333b3234
    [Length: 4]

0000  60 00 00 00 00 15 11 3e  aa aa 00 00 00 00 00 00   `·····> ········
0010  02 12 74 03 00 03 03 03  aa aa 00 00 00 00 00 00   ··t····· ········
0020  00 00 00 00 00 00 00 01  16 33 ec 36 00 15 9c 79   ········ ·3·6···y
0030  64 45 9e 97 df c6 41 65  ff 33 3b 32 34           dE····Ae ·3;24
```

Figura 42 - Captura del Frame 48 Datos

15. Conclusiones

Conclusiones y recomendaciones a las que se ha llegado después de meses de trabajo con los dos sistemas operativos.

Comenzando con puntos de comparación entre dos Sistemas Operativos que son bastante populares, ambos presentan un gran número de aplicaciones y referencias. En un principio ContikiOS aparece como un sistema operativo atractivo para la implementación de soluciones y/o para tareas de investigación, porque se muestra como un sistema versátil y simple, con muchos elementos que se deben desarrollar y depurar, pero al poco andar en la construcción de aplicaciones se comienza a encontrar con problemas de documentación formal que permitan resolver problemas de mediana complejidad, es en esta etapa donde el único punto en que se pude encontrar una solución es el directorio de código fuente de ContikiOS, esto ocurre porque la documentación oficial disponible solo resuelve problemas básicos, los foros y listas de correo oficiales que se muestran en este análisis resuelven problemas avanzados y por los general las respuestas a los problemas de mediana complejidad son derivados a que se efectué una consulta a los ejemplo disponibles junto al código fuente de ContikiOS. En el caso de TinyOS esto no sucede y se debe principalmente a su sistema de documentación (TEPs) que están bastante bien estructurado, permitiendo especializarse bastante rápido en la estructura de sus aplicaciones, al igual que ContikiOS cuanta con un directorio de ejemplo que también representan una buena fuente de información, teniendo estos antecedentes podemos afirmar que en este punto la documentación es TinyOS quien lleva la delantera y permite tener una mejor curva de aprendizaje.

Un segundo elemento muy importante y que tiene una enorme relevancia para la Construcción de Aplicaciones es el Entorno de Desarrollo Integrado (IDE), hoy en día los entornos de desarrollo integrados son una herramienta imprescindible para la construcción y depuración de aplicaciones siendo decisivo en la velocidad con que se construyen las aplicaciones, un IDE puede tener una infinidad de funcionalidades y en este trabajo de tesis también se analizaron los sistemas operativos desde esta perspectiva. Se instalaron, configuraron y analizaron entornos de desarrollo para los dos Sistemas Operativos en ambos casos las mejores características las ofrece Eclipse, pero con una enorme diferencia que es impuesta por los plugins que son instalados. Por un lado, ContikiOS instalar una extensión que sólo permite el reconocimiento de palabras reservadas y con algo más de trabajo en la configuración, se logra que la compilación de las aplicaciones se pueda hacer desde los menús dinámicos de Eclipse. En el caso TinyOS este cuenta con una extensión que permite el autocompletado de código, funcionalidad que es de mucha utilidad al momento de hacer las conexiones entre los eventos, comando e interfaces de una aplicación, esto representa una gran ayuda para el programador liberándolo de aquellas tareas que requieren un gran cuidado, además permite que el programador no tenga que recurrir a la documentación para averiguar por ejemplo cuales son las interfaces de un módulo del Sistema Operativo.

La forma en que se construyen las aplicaciones en cada entorno analizado es distinta, si bien podría pensarse que es una afirmación obvia por el simple motivo que utilizan lenguajes de programación distintos, pero la diferencia que se intenta poner de manifiesto apunta principalmente a la reutilización de código, algo importante a la hora de construcción de aplicaciones modulares, en ese sentido es TinyOS quien saca muy buen partido de técnicas de encapsulado que permiten construir pequeños elementos (componentes) que pueden ser reutilizados por

otros componentes, la dificultad aquí radica en la planificación y diseño de las aplicación. ContikiOS no se preocupa de la reutilización, sino que más bien en la optimización de algoritmos que permitan hacer un uso más eficiente de los escasos recursos con que cuentan las motas, esto no quiere decir que sus algoritmos y/o aplicaciones no se puedan reutilizar.

Un elemento importante y no muy discutido a la hora de utilizar y/o configurar redes de sensores con alguno de estos sistemas operativos es la interacción con el usuario final, ambos sistemas operativos no ofrecen una interface amigable que permita que un usuario común con conocimiento básicos pueda instalar y configurar una red de sensores, para poder configurar una red básica se requieren conocimientos avanzados de sistemas operativos y redes, en este sentido la tarea para hacer más accesible estas tecnologías está todavía en una etapa incipiente y puede ser motivo para algún aporte con algún trabajo de postgrado, por ejemplo la implementación de una aplicación o interface (Frameworks) que permita seleccionar las funcionalidades que se desean cargar en los dispositivos dentro de las funcionalidades que se requieren están:

- Seleccionar los sensores que se desean activar en el caso de nodos sensores y en caso de ser un nodo recolector por ejemplo como se desea almacenar la información (base de Datos y/o texto plano).
- Seleccionar las funcionalidades que se requieren en los nodos, en concretos seleccionar los protocolos que se requieren en cada nodo (http, ftp, email)
- Incorporar una interface que permita cargar nuevas aplicaciones creadas por usuarios avanzados algo similar a como funcionan los plugin.

Las motas TelosB tienen capacidades de extensión en su hardware de fácil conexionado, se podría escribir la inserción de algún tipo de conector para algún otro sensor por ejemplo toda la línea de desarrollo de Arduino cuenta con una enormidad de sensores se podía hacer una extensión de hardware para hacer compatible TelosB con los sensores de Arduino

Concluido este trabajo que pone a prueba varios elementos que forman parte de esta nueva Internet y para analizar si este estudio estaba en la dirección correcta se efectúa una revisión del estado del Arte, como por ejemplo el Open IoT Challenge (OpenIoT, 2018) que publica una recopilación de los principales hallazgos en su estudio sobre tendencias en el desarrollo del Internet de las Cosas, y en ella se verifica que varios de los elementos tratados en este Trabajo de Tesis representan hoy una tendencia en lo que a Internet de las Cosas se refiere.

16. Bibliografía

En esta sección se detalla la bibliografía referenciada y consultada para el desarrollo de este proyecto, copias de los documentos están disponibles en https://tinyurl.com/BibliografiaUNLP. Todos los documentos tienen una etiqueta permite identificar el documento en el repositorio con respecto a esta bibliografía.

Alliance, Z. (2014). ZigBee.org. Obtenido de http://www.zigbee.org/wp-content/uploads/2014/11/docs-07-5356-19-0zse-zigbee-smart-energy-profile-specification.pdf [A1]

Ayyaz Mahmood, S. I. (2016). https://uia.brage.unit.no/uia-xmlui/handle/11250/2433522 Obtenido de Web Site University of Agder. [A2]

Correa, F. (2017). Simulación de Algoritmos de Movimientos Tácticos para Equipos de Rescate. Obtenido de http://bibing.us.es/proyectos/abreproy/70445

Free Software Foundation. (2018). *El sistema operativo GNU*. Obtenido de: https://www.gnu.org/licenses/license-list.en.html

García Serrano, A. (2018). Sistemas digitales e Internet de las cosas. Obtenido de https://www.digilogic.es/modos-de-bajo-consumo-en-msp430/ [G1]

Lincoln Laboratory. (1985). Lincoln Laboratory. Obtenido de https://people.eecs.berkeley.edu/~pister/290Q/Papers/History%20and%20Context/Lacoss%20DSN%201986.pdf [L1]

Ludovici, A. (2013). Journal of Sensor and Actuator Networks. Obtenido de http://www.mdpi.com/2224-2708/2/2/288 [L2]

Moore, S. (2013). Blog Wireless Sensor Networks. Obtenido de https://wirelesssensornetworks.weebly.com/blog [M1]

Nikoi, N. (2000). ieee. Obtenido de https://standards.ieee.org/standard/1451_2-1997.html [N1]

OpenIoT. (2018). Open IoT Challenge 4.0. Obtenido de https://www.eclipse.org/org/press-release/20180404_iotchallenge_winners2018.php [O1]

Pat Kinney, P. J. (2003). EEE 802.15 WPAN™ Task Group 4 (TG4). Obtenido de http://www.ieee802.org/15/pub/TG4.html [P1]

Philip Levis, E. B. (2008). The Emergence of a Networking Primitive in Wireless Sensor Networks. Communications of the ACM, 99.

Pol Moreno, a. A. (2013). TinyCoAP: A Novel Constrained Application Protocol (CoAP). Journal of Sensor and Actuator Networks.

Shelby, Z. (2014). IETF. Obtenido de https://tools.ietf.org/html/rfc7252 [S1]

Waher, P. (2015). Learning Internet of Things. UK: Packt Publishing.

Whitman, E. C. (2005). Obtenido de Undersea Warfare Magazine: http://www.public.navy.mil/subfor/underseawarfaremagazine/Issues/Archives/issue_25/sosus.htm [W1]

Zach Shelby, C. B. (2009). 6LoWPAN : the wireless embedded internet. Chippenham, Great Britain: John Wiley & Sons Ltd.

Adam Dunkels, B. G. (2004). Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors. Obtenido de http://www.dunkels.com/adam/dunkels04contiki.pdf [A4]

Advanticsys. (2015). Obtenido de http://www.advanticsys.com/shop/documents/1320249906_Temperature_Humidity_Sensirion_SHT1x.pdf [A5]

Computer Science Department - Cornell University. (2010). The MagnetOS Operating System. Obtenido de http://www.cs.cornell.edu/people/egs/magnetos/

David E. Culler, G. T. (2009). Compact Application Protocol (CAP): Uniting the Best of IP and ZigBee. Obtenido de http://intelligentsystemssource.com/arch-rock/ [D1]

Derpsch, S. C. (2012). Biblioteca Universidad Catolica de Chile. Obtenido de https://repositorio.uc.cl/bitstream/handle/11534/1723/592209.pdf?sequence=1&isAllowed=y [D2]

Dunkels, A. (s.f.). Dunkels. Obtenido de http://dunkels.com/adam/pt/expansion.html [D3]

Castellani, A., Loreto, S., Rahman. (2016). Guidelines for HTTP-to-CoAP Mapping Implementations. Obtenido de ietf.org: https://datatracker.ietf.org/doc/rfc8075/ [C1]

Dunkels, A. (s.f.). SODA, the Software institutes' Online Digital Archive. Obtenido de http://soda.swedish-ict.se/2372/1/SICS-T--2005-05--SE.pdf [D3]

Free Software Foundation. (2014). GNU Make. Obtenido de http://www.gnu.org/software/make/ [F1]

Gay, D. L. (2003). The nesC language: A holistic approach to networked embedded systems. Proceedings of the acm sigplan 2003 conference on programming language design and implementation, 1-11.

Hector Abrach, J. C. (2003). MANTIS: System Support For MultimodAl NeTworks of In-situ Sensors. Obtenido de http://www.cs.colorado.edu/~rhan/MANTIS_tech_report_CU_CS_950_03.pdf

Hill, J. S. (2000). architecture directions for networked sensors. ACM SIGARCH Computer Architec-.

John Yannakopoulos, A. B. (s.f.). CORMOS: A Communication-Oriented Runtime System for Sensor Networks. Obtenido de http://users.ics.forth.gr/~bilas/pdffiles/cormos-ewsn05-cr.pdf [J1]

Joshua Lifton, D. S. (2002). Pushpin Computing System Overview:A Platform for Distributed, Embedded,Ubiquitous Sensor Networks. Obtenido de http://resenv.media.mit.edu/pushpin/publications/PushpinPervasive2002.pdf [J2]

Levis, P. (2006). TinyOS Programming. Obtenido de http://csl.stanford.edu/~pal/pubs/tinyos-programming.pdf [L1]

Levis, P. M. (2004). The emergence of networking abstractions and techniques in TinyOS. Berkeley, USA. Obtenido de https://people.eecs.berkeley.edu/~culler/papers/tinyos-nsdi03.pdf [L2]

Lin Gu, J. A. (2009). t-kernel: Providing Reliable OS Support to Wireless Sensor Networks. Obtenido de http://www.cs.virginia.edu/~stankovic/psfiles/fp01-gu.pdf [L3]

Masaaki Takahashi, B. H. (2009). Demo Abstract: Design and Implementation of a Web Service for LiteOS-based Sensor Networks. Obtenido de http://www.cs.wichita.edu/~bintang/papers/webservice.pdf [M1]

P. Levis, S. M. (2001). TinyOS: An Operating System for Sensor Networks. Obtenido de http://www.cs.berkeley.edu/~culler/papers/ai-tinyos.pdf [P1]

Photonics, H. (2014). Specification Sheet S1087-S1133 Series. Obtenido de https://www.hamamatsu.com/resources/pdf/ssd/s1087_etc_kspd1039e.pdf [P2]

Proconx. (2008). Programming the Nut/OS httpd example application. Obtenido de http://www.proconx.com/assets/files/appnotes/AN106-0801.pdf [P3]

Ruiz, M. l. (1999). Sensores Inteligentes: La Revolución Tecnológica de la Instrumentación. Departamento de Ing. Electrónica. UPC. Obtenido de http://upcommons.upc.edu/revistes/bitstream/2099/9745/1/Article015.pdf [R1]

Tanenbaum, A. S. (2009). Sistemas Operativos Modernos. Naucalpan de Juárez, Estado de México: Pearson Education, Inc.

TEPS. (2013). Documentación. Obtenido de http://tinyos.stanford.edu/tinyos-wiki/index.php/TEPs [T1]

Villanueva, F. J. (2013). Blog TinyOS. Obtenido de https://sites.google.com/site/felixjesusvillanueva/testimonials-1/tinyos [V1]

Welton, D. N. (2004). Il Sistema Operativo eCos. Obtenido de http://www.welton.it/articles/ecos.html [W1]

S. Namal (2016) Implementation of OpenFlow based Cognitive Radio Network Architecture: SDN&R, Wireless Networks 22(2): 663-677 2016.

Manveer Joshi, B. P. (2015). CoAP Protocol for Constrained Networks. Obtenido de http://www.mecs-press.org/ijwmt/ijwmt-v5-n6/IJWMT-V5-N6-1.pdf [M1]

P. Porambage, A. B. (2015). Secure End-to-End Communication for Constrained Devices in IoT-enabled Ambient Assisted Living Systems. Obtenido de https://www.researchgate.net/publication/282671713_Secure_End-to-End_Communication_for_Constrained_Devices_in_IoT-enabled_Ambient_Assisted_Living_Systems [P1]

Chun, S.-M., & Park, J.-T. (2015). Mobile CoAP for IoT mobility management. Obtenido de http://ieeexplore.ieee.org/document/7157990/ [C1]

Ashish Patro, S. B. (2015). Outsourcing Coordination and Management of Home Wireless Access Points through an Open API. Obtenido de IEEE INFOCOM 2015: http://pages.cs.wisc.edu/~patro/papers/coap_infocom15.pdf [A3]

Shelby, Z., Bormann, C. (2009). 6LoWPAN: The Wireless Embedded Internet, Alemania, Wiley

Dunkels, A., Gronvall,B., Voigt T. (2004), Contiki - a lightweight and flexible operating system for tiny networked sensors. In Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I), Tampa, Florida, USA.

Dunkels, A. (2012). Contiki: The Open Source OS for the Internet of Things. extraido de http://contiki.sourceforge.net/docs/2.6/ [D1]

García, E. (2009). Implementación de Protocolos de Transporte en Redes de Sensores Extraído de https://upcommons.upc.edu/bitstream/handle/2099.1/7480/PFM-ERNESTOGARCIADAVIS.pdf. Universidad Politécnica de Cataluña, España

17. Anexo A

17.1. Codificación Archivo de Módulos Aplicación para TinyOS

```
1 #include<UserButton.h>
2 #include"MoteToMote.h"
3
4 module MoteToMoteC
5 {
6     // Definir Interfaces
7     uses
8     {
9         interface Boot;
10        interface Leds;
11        interface Get<button_state_t>;
12        interface Notify<button_state_t>;
13        interface Packet;
14        interface AMPacket;
15        interface AMSend;
16        interface SplitControl as AmControl;
17        interface Receive;
18
19    }
20 }
21
22 implementation
23 {
24    bool _radioBusy = FALSE;
25    message_t _packet;
26    event void Boot.booted()
27    {
28        call Notify.enable();
29        call AmControl.start();
30
31    }
32    event void Notify.notify(button_state_t val)
33    {
34        if (_radioBusy == FALSE)
35        {
36            //Creando paquete
37            MoteToMoteMsg_t* msg = call Packet.getPayload(&_packet,
38                                                    sizeof(MoteToMoteMsg_t));
39            msg->NodeID=TOS_NODE_ID;
40            msg->Data=(nx_uint8_t)val;
41
42            //Enviando paquete
43            if ((call AMSend.send(AM_BROADCAST_ADDR, & _packet,
44                                  sizeof(MoteToMoteMsg_t))) == SUCCESS)
45                _radioBusy = TRUE;
46        }
47    }
48
```

Figura A - 1 Codificación Archivos de módulos Parte 1/2

```
49
50      event void AMSend.sendDone(message_t *msg, error_t error)
51      {
52          if (msg==&_packet)
53              _radioBusy=FALSE;
54      }
55      event void AmControl.startDone(error_t error)
56      {
57          if (error==SUCCESS)
58                  call Leds.led0On();
59              else
60                  call AmControl.start();
61      }
62      event message_t * Receive.receive(message_t *msg, void *payload,
63                                                          uint8_t len)
64      {
65          if (len==sizeof(MoteToMoteMsg_t))
66          {
67              MoteToMoteMsg_t * incomingPacket =(MoteToMoteMsg_t*)payload;
68              //incomingPacket->Data==2;
69              uint8_t data = incomingPacket->Data;
70              if (data==1)
71                  call Leds.led2On();
72              if (data==0)
73                  call Leds.led2Off();
74          }
75          return msg;
76      }
77
78      event void AmControl.stopDone(error_t error){}
79  }
```

Figura A - 2 Codificación Archivos de Módulos Parte 2/2

17.2. Codificación Archivo de Configuración Aplicación para TinyOS

```
configuration MoteToMoteAppC
{

}
implementation{
    components MoteToMoteC as App;
    components MainC;
    components LedsC;

    App.Boot -> MainC;
    App.Leds -> LedsC;

    //configuracion botones
    components UserButtonC;

    App.Get ->  UserButtonC;
    App.Notify -> UserButtonC;

    //Comunicaciones Radios
    components ActiveMessageC;
    components new AMSenderC(AM_RADIO);
    components new AMReceiverC(AM_RADIO);
    App.Packet -> AMSenderC;
    App.AMPacket -> AMSenderC;
    App.AMSend -> AMSenderC;
    App.AmControl -> ActiveMessageC;
    App.Receive -> AMReceiverC;

}
```

Figura A - 3 Codificación de Archivos de configuraciones

```
#ifndef MOTE_TO_MOTE_H
#define MOTE_TO_MOTE_H

typedef nx_struct MoteToMoteMsg
{
    nx_uint16_t NodeID;
    nx_uint8_t Data;
} MoteToMoteMsg_t;

enum
{
    AM_RADIO = 6
};

#endif /* MOTE_TO_MOTE_H */

```

Figura A – 4 Codificación de Archivos de configuraciones