SISTEMAS DE MONITOREO DE EVENTOS APLICANDO EL PROTOCOLO 6LOWPAN

CHRISTIAN FERNANDO CHAZATAR CUASTUMAL

UNIVERSIDAD DEL VALLE
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA
PROGRAMA ACADÉMICO DE INGENIERÍA ELECTRÓNICA
SANTIAGO DE CALI
2014

SISTEMAS DE MONTOREO DE EVENTOS APLICANDO EL PROTOCOLO 6LOWPAN

CHRISTIAN FERNANDO CHAZATAR CUASTUMAL

Trabajo presentado como requisito parcial para optar al título de
Ingeniero Electrónico

Director:
Fabio Germán Guerrero
Co-Director:
Asfur Barandica López

UNIVERSIDAD DEL VALLE
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA
PROGRAMA ACADÉMICO DE INGENIERÍA ELECTRÓNICA
SANTIAGO DE CALI
2

Agradecimientos

El autor les expresa sus más sinceros agradecimientos a los docentes Fabio Germán Guerrero y Asfur Barandica López que amablemente y desinteresadamente brindaron sus valiosos aportes y sugerencias durante la realización de este proyecto. De igual manera a la Universidad del Valle por darme la oportunidad de llevar a cabo este trabajo de grado, como también a los compañeros y amigos Andrés Felipe Grisales, Johannes Allen Sánchez y Gustavo Moreno Córdoba por ser el apoyo incondicional durante este proceso, finalmente agradecer a mis padres y a mi hermana ya que sin ellos no hubiera sido posible llegar a esta instancia de la vida.

Dedicado a mi familia, compañeros y demás seres queridos por su Incondicional apoyo durante todos estos años de formación.
A mis Padres:
Evelio Arnoldo Chazatar Bastidas y Dolores Elpidia Cuastumal
A mi Hermana:
Lizeth Omaira Chazatar Cuastumal
A mi tío:
Jaime Alberto Chazatar Bastidas (Q.E.P.D)

TABLA DE CONTENIDO

Pág.

INDICE DE FIGURAS

Pág.

INDICE DE TABLAS

Pág.

GOSARIO

ACK. Acknowledgement. Es un mensaje de confirmación generado por el destino que anuncia la recepción de un dato.

AES. Advanced Encryption Standard. Algoritmo de encriptación de información establecido para sistemas de alta seguridad.

CSMA-CA. Carrier Sense Multiple Access with Collision Avoidance. Protocolo de control de acceso a redes de bajo nivel que permite que múltiples estaciones utilicen un mismo medio de transmisión.

DSSS. Direct Sequence Spread Spectrum. Es una técnica de codificación de canal usando la técnica de modulación de espectro ensanchado para transmisión de señales digitales y por radio frecuencia definida por el IEEE

IEEE. Institute of Electrical and Electronics Engineers. Es una asociación mundial técnico-profesional dedicada a la estandarización de tecnológica Promueve la creatividad, el desarrollo y la integración, compartir y aplicar los avances en las tecnologías de la información, electrónica y ciencias en general para beneficio de la humanidad.

ICMP. Internet Control Message Protocol. Protocolo de control y notificación de errores para el protocolo IP.

IP. Internet Protocol. Protocolo no orientado a conexión que se utiliza para la comunicación de datos.

ISM. Industrial, Scientific and Medical. Son bandas reservadas internacionalmente para uso no comercial de radiofrecuencia electromagnética

RDC. Radio Duty Cycling. Mecanismo de ciclo de trabajo establecido en Contiki para gestionar el encendido y apagado de la radio.

RFC. Request for Comments. Establece una serie de documentos que contienen información oficial sobre los protocolos de Internet, publicados por la *Internet Engineering Task Force* (IETF)

UDP. User Datagram Protocol. Protocolo de nivel de transporte que se basa en el intercambio de datagramas.

PHY. Physical Layer. Es la interfaz entre la MAC y el medio inalámbrico.

WPAN. Wireless Personal Area Network. Redes de computadoras establecidas en un bajo rango para uso personal.

WSN. Wireless Sensor Network. Red inalámbrica de dispositivos con capacidades sensitivas que colaboran con una tarea en común.

RESUMEN

Actualmente en nuestra región el campo de las redes de sensores inalámbricas carece de exploración [1], lo que constituye una motivación por la investigación de este tipo de tecnologías.

El objetivo de este proyecto acercarse más al mundo de las redes de sensores, llevando siempre de la mano la filosofía y metodología del software y hardware libre, posibilitando así la apropiación de nuevas tecnologías.

Como primer punto se presenta la investigación acerca de los estándares y tecnologías que conforman las redes de sensores inalámbricas, posteriormente se procede a estudiar 6LoWPAN (IPv6 Over Low Power Wireless Personal Area Networks), estándar que permite el uso de IPv6 en redes de sensores.

Para la implementación de la red, se hace uso de las tarjetas de desarrollo AVR-RAVEN las cuales están disponibles en el grupo de investigación SISTEL-UV. Estos dispositivos se encuentran compuestos por los elementos necesarios para establecer una red inalámbrica. Los elementos de los dispositivos pueden ser fácilmente encontrados en el mercado, permitiendo así en trabajos futuros la construcción de nuevos equipos con la ayuda de software de libre distribución.

Las tarjetas están basadas en un microcontrolador de 8-bits que se encarga de gestionar los periféricos de entrada/salida y del dispositivo de radio transmisión (transceiver) el cual se encarga de enviar los paquetes de información bajo el estándar IEEE 802.15.4.

Para realizar una mejor estructura y gestión de la red se hace uso del sistema operativo Contiki con licencia BSD (Berkeley Software Distribution) el cual establece políticas de distribución de código abierto. El uso del sistema operativo permite la portabilidad y versatilidad en diferentes tipos de plataformas. Una de las características que primó en la elección del sistema operativo es su lenguaje de programación, que para el caso de Contiki es el lenguaje de programación C.

Finalmente se construye una aplicación prototipo para la validación y configuración del sistema, la cual se encarga de recolectar la información de los sensores.

SUMMARY

Today technological progress has evolved by leaps and bounds and thereby opening up new fields of research such as is the study of wireless sensor networks. Currently in our region the field of wireless sensor networks lacks exploration, this has been a motivation for investigating this type of technology.

The objective of this project is to get closer to the world of sensor networks, being always together hand the philosophy of free software and hardware, allowing the appropriation of new technologies

In the beginning was performed research on what are standards and technologies that make wireless sensor networks, then we proceed to study the 6LoWPAN (IPv6 over Low power Wireless Personal Area Networks) which is a standard that allows the use of IPv6 in sensor networks.

For the implementation of the network, it makes use of AVR development boards-RAVEN which is available in the research group SISTEL-UV, these devices are made with the components necessary to establish a wireless network. The components that comprise it can easily be found on the market, thus enabling future work building new devices with the help of free software.

The cards are made with a microcontroller 8-bit that manages peripheral input / output, as well as managing the radio transmission device (transceiver) which is responsible for sending data packets on the IEEE 802.15.4.

For better structuring and management of the network makes use of Contiki Operating Systems with licensed BSD (Berkeley Software Distribution) which establishes policies open source distribution, the use the operating system allows portability and versatility on different types of platforms, one of the characteristics that prevailed in the choice of operating systems is its programming language, in our case Contiki is designed in C programming language.

Finally, make a prototype application for validation and system configuration, which is responsible for collecting the information from sensors.

INTRODUCCIÓN

En la actualidad nos encontramos rodeados por miles de redes inalámbricas (wireless) que circundan nuestro entorno, conformadas por dispositivos capaces de establecer enlaces de conexión, como las de celular, tablet, smartphone, modems, etc, que cada vez se adicionan más a la vasta red inalámbrica de conexión global.

En los últimos años el creciente uso de las tecnologías wireless en las telecomunicaciones abre una brecha en los diversos campos de investigación incorporando al uso de las redes de sensores inalámbricas, solventado así necesidades asociadas al control, censado y monitoreo de variables para diferentes campos de aplicación. Anteriormente los enlaces de comunicación se establecían por medio de cables, esto implicaba que las redes no fueran lo suficientemente propagadas, por sus costos y limitantes asociados a la expansión de la red.

Tecnologías como Bluetooth y Zigbee. Han permitido la creación de las redes de sensores inalámbricas WSN por sus siglas en ingles "*Wireless Sensor Networks*" y con esto un sinnúmero de aplicaciones de ámbito social, como comercial. La evolución de las WSN ha llegado al punto de permitir enlaces de conexión a Internet y por consiguiente el uso del estándar IPv6, implicando así que cada nodo sensor tenga una dirección IP dentro de la red.

La necesidad de interacción del hombre con su entorno, ha llevado al punto de crear espacios omni-presenciales que permiten a las personas interactuar con los entornos físicos, permitiendo el acceso a la información que se presenta en el lugar.

El estudio de estas tecnologías está enfocado en buscar y adquirir conocimientos en los campos del hardware y software libre. También posibilitando que la electrónica preste un buen servicio a la sociedad y la conservación del planeta, buscando un ahorro de energía y colaborando en la construcción de sistemas de procesamiento más colectivos y expandibles.

En el primer capítulo de este documento se presentan los objetivos planteados para la realización de este proyecto.

El segundo capítulo presenta la información necesaria para conceptualizar y comprender acerca de los sistemas WSN, como también el protocolo IEEE 802.15.4, se explica cómo funciona 6LoWPAN bajo la estructura IPV6, mostrando el trabajo de la capa de adaptación para el estándar IEEE 802.15.4 que establece la IETF (The Internet Engineering Task Force).

El tercer capítulo presenta como está conformado el hardware necesario para establecer un enlace 6LoWPAN, como también el diseño del nodo sensor encargado de capturar las variables del entorno.

En el cuarto capítulo se muestra como está conformado el software y los componentes necesarios para permitir la configuración de la red 6LoWPAN, como también el diseño de la aplicación, esta aplicación permite monitorear una serie de sensores los cuales posibilitan obtener información acerca del entorno en el cual se encuentran.

El quinto capítulo explica la forma como se compone el software para la ejecución del proyecto, y la instalación del Sistemas Operativo Contiki encargado de comunicar los dispositivos y la adquisición de datos obtenidos, luego se presenta el hardware requerido para la ejecución del proyecto, presentando las características establecidas por el dispositivo para la validación del proyecto.

Finalmente se presenta los archivos necesarios de instalación, configuración y anexos referentes a este documento.

CAPÍTULO 1

OBJETIVOS Y ALCANCES

Con la aprobación a nivel mundial de IPv6 y el fuerte impacto que ello representa, se abre un vasto panorama de investigaciones relacionadas con la adopción e implementación de esta tecnología. Una de ellas se identifica en el campo de las redes de sensores inalámbricas basadas en IP, estableciendo un nuevo concepto de Internet llamado "Internet de las cosas" [2].
El uso de IPv6 en la región relativamente es bajo, lo cual motivó a investigar y diseñar un red de sensores inalámbrica 6LoWPAN, optando por el uso de herramientas de libre distribución, llevando consigo la filosofía del software libre, permitiendo así tener un punto de partida para futuros proyectos de implementación y diseño.

1.1 OBJETIVO GENERAL

Desarrollo de una aplicación de redes inalámbricas para el monitoreo de eventos y transmisión de datos usando un sistema embebido, que emplee la tecnología 6loWPAN permitiendo acceso remoto a la información a través de Internet.

1.2 OBJETIVOS ESPECÍFICOS

- Estudiar los protocolos que conforman 6loWPAN orientado a dispositivos con recursos limitados en redes de sensores.
- Implementar un sistema canónico de transmisión y recepción de datos a través de Internet usando 6LoWPAN.
- Desarrollar una aplicación para la interpretación de datos obtenidos del comportamiento de los sensores en la red 6LoWPAN.
- Validar el monitoreo de eventos y la transmisión de datos bajo la estructura de red 6LoWPAN en el contexto de una aplicación prototipo.

1.3 ALCANCE DE LOS OBJETIVOS

En la culminación del proyecto se presenta la documentación pertinente, que concierne a las especificaciones y funcionamiento de la red.
La finalidad del proyecto es presentar una red básica 6LoWPAN la cual se encarga de adquirir datos, Igual se implementa una aplicación que permita realizar consulta de datos de la red de sensores, desde un punto remoto, posteriormente se entrega los códigos de los prototipos del proyecto.
Por último se muestran los resultados obtenidos en el transcurso de las pruebas realizadas para el proyecto, con el fin de verificar el buen funcionamiento de la red.

CAPÍTULO 2

MARCO TEÓRICO

En este capítulo se explican los conceptos necesarios para el entendimiento de las redes de sensores WSN, se hace la introducción de algunas de las aplicaciones más comunes y se presenta la explicación del protocolo 6LoWPAN.

2.1 ESTADO DEL ARTE

Tradicionalmente los sensores han sido elementos indispensables en la industria, debido a la capacidad que proporcionan al monitorear y manipular las señales físicas involucradas en los diferentes procesos productivos. La conectividad entre los equipos de monitoreo y control con sensores se realizaba mediante redes cableadas tradicionales. Actualmente los constantes avances tecnológicos han permitido el desarrollo de dispositivos con capacidades de comunicación inalámbrica, dispersos en cualquier localización. Cada vez más pequeños más autónomos; de ahí las redes de sensores inalámbricas (WSN), que están compuestas por una serie de miles, incluso millones de sensores llamados nodos, los cuales poseen capacidad de almacenamiento y procesamiento, con energía limitada. El continuo desarrollo de estas particulares redes ha provocado su incorporación y uso en ámbitos muy diferentes. Desde la automatización ambiental (humedad, temperatura, luz, etc.) cualidad fundamental para el desarrollo de la domótica, pasando por las aplicaciones militares, industriales, médicas o comerciales [3].
Con el nacimiento de las WSN se presenta la necesidad de conectarse a la vasta red de interconexión mundial, surgiendo así 6LoWPAN, el cual es una adaptación del protocolo de comunicación IP (*Internet Protocol*), permitiendo así la interconexión con redes IPv6. Recientemente las redes de sensores inalámbricas WSN han impulsado un gran interés en diversas ramas de aplicación, por su facilidad y versatilidad de uso al no estar limitadas por condiciones topográficas, convirtiéndose en uno de los sistemas masivamente desplegables, en la recolección de datos para lugares que antes eran inaccesibles.

2.1.1 REDES DE SENSORES INALÁMBRICOS (WSN)

Una red de sensores se define como un tipo particular de red inalámbrica, compuesta por un conjunto de pequeños dispositivos o sensores con limitaciones en cuanto a memoria, ancho de banda, velocidad de procesamiento y almacenamiento de energía. Mucho más limitado que en el caso de los nodos establecidos con redes inalámbricas tradicionales.

Estas redes se caracterizan por ser auto configurables, de fácil despliegue con un alto número de dispositivos interconectados, con capacidades de hacer tareas de emisión y/o recepción de datos, caracterizándose por la miniaturización en la construcción de nodos desarrollando computadoras extremadamente pequeñas y baratas.

Las redes de sensores son un concepto relativamente nuevo concerniente a la adquisición y tratamiento de datos, que se pueden establecer por diversos estándares dependiendo del su enfoque y políticas de distribución.

***Tabla 2.1**: Tecnologías WSN [4]*

Estándar	**ZigBee**	**6LoWPAN**	**WirelessHART**
Aplicación principal	Monitoreo y control	Monitoreo y control	Monitoreo y control industrial
Memoria	4-32KB	4-32KB	-
Tiempo de vida (días)	100-1000+	100-365+	760+
Nodos de red	255	65536	200
Rendimiento	Hasta 250 Kbps	Hasta 250 Kbps	Hasta 250 Kbps
Rango (metros)	1-75	1-100	1-100
Características principales.	Confiabilidad, bajo costo, bajo consumo de energía	IPv6 sobre IEEE 802.15.4	Confiabilidad

2.2 APLICACIONES

Ciertas aplicaciones en las cuales se encuentran involucradas las (WSN), se han establecido como campo de investigación en el análisis de sistemas de monitoreo en la agricultura de precisión [5], como también se en el campo de la salud realizando monitoreo de pacientes y suministro de medicamentos [6]. Por otro lado en el campo medio ambiental se presenta el monitoreo de niveles de luz, humedad y temperatura, condiciones que influyen en la preservación de humedales de las cuencas hídricas y conservación de bosques, sistemas de prevención de desastres realizando seguimiento constante de lugares de actividad sísmica, detección de incendios [7]. En la parte comercial se encuentra la automatización de entornos, control y congestión vehicular, sistemas de distribución eléctrica “Smart Grids” o redes inteligentes [8].

2.2.1 APLICACIONES EN LA SALUD

Dentro de la rama de la salud es de vital importancia el uso de equipos encargados de capturar las señales fisiológicas y electromiografías [9], por lo cual se ha buscado la manera de monitorear los pacientes en tiempo real de forma inalámbrica. Esto ha contribuido a que los pacientes no tenga que cargar con equipos de gran tamaño, presentando un grado de comodidad en los pacientes al no estar atado a cables, permitiendo un enfoque de diagnóstico preventivo, como también el empleo en la

detección temprana de problemas cardiovasculares. En casos donde los pacientes con enfermedades crónicas, terminales o personas de la tercera edad no se encuentran dentro de las instalaciones hospitalarias empleado sistemas de monitoreo remoto con enfoque de tele asistencia preventiva.

Principalmente el área en el que se ha enfocado el uso de esta tecnología es en los sistemas de control de medicamentos [10], tal es el caso del control de insulina en la sangre o funciones concernientes a la tensión arterial, ya que se posibilitaría el control de las variables censadas y suministro del medicamento necesario para controlar dicha señal.

2.2.2 APLICACIONES EN EL SECTOR MEDIOAMBIENTAL

La preservación del medio ambiente siempre ha sido de gran interés en campos de protección y preservación, tal es el caso de la preservación de fuentes hídricas y protección de bosque y humedales, control de suministro de aguas, sistemas de prevención de desastres (incendios, desbordamiento de ríos, áreas de actividad símica)[6], control y uso racional de insumos químicos, control de emisión de gases tóxicos, sistemas de monitoreo de especies en vía de extinción, etc, lo cual las WSN se establecen como herramientas de censado con la cualidad que les permiten ser propagadas en terrenos de difícil acceso.

2.2.3 APLICACIONES EN EL SECTOR AGRÍCOLA

El sector agrícola es uno de los campos donde más se ha explorado el uso de la WSN, caracterizándose por permitir el acceso a variables físicas que presenta el medio[11], el enfoque que se le da a este tipo de aplicaciones es buscar el mejor rendimiento de producción, implementando sistemas de riego inteligente, sistemas de control de PH, tiempo de exposición a la radiación, temperatura, niveles de humedad; variables necesarias para examinar el buen rendimiento del crecimiento en los productos agrícolas.

2.3 IEEE 802.15.4

El estándar IEEE 802.15.4 especifica la capa física y de control de acceso al medio, diseñado para redes de acceso personal (WPANs) las cuales trabajan con baja transferencia de datos. El estándar establece que la conexión entre nodos se debe realizar de forma inalámbrica. Otra de las particularidades es que soporta diferentes topologías, tales como: redes malla, anillo, estrella [12]. Las topologías multi-hop están pensadas para dispositivos con una pila completa de protocolos en los cuales se programan diversos algoritmos para el enrutamiento de paquetes por diferentes caminos.
Su alcance oscila en rangos de 10 a 100 metros dependiendo del desempeño de la arquitectura; su direccionamiento IEEE 64-BIT, soporta 264^2 Nodos: su Seguridad AES 128, el control de acceso al medio se hace por CSMA-CA

2.3.1 CAPA FISICA

La capa física del estándar IEEE 802.15.4 se divide en las subcapas PHY *data servicie* y PHY *Management* que son las que se encargan de seleccionar el canal, recibir y transmitir mensajes a través de radio.

Opera en uno de los siguientes rangos de frecuencia.

- 868-868.8 MHz esta franja se usa respecto a la normatividad europea, permite un canal de comunicación (2003), extendido a tres (2006).

- 902-928 MHz esta franja se usa respecto a la normatividad EE.UU, hasta diez canales (2003) extendidos a treinta (2006).

- 2400-2483.5MHz es la franja mundialmente aceptada para ISM, hasta dieciséis canales (2003, 2006).

La figura 2.1 muestra la versión original del estándar del año 2003, esta especifica dos niveles físicos basados en DSSS (Direct Sequence Spread Spectrum): uno en las bandas de 868/915 MHz con tasas de 20 y 40 kbps; y otra en la banda de 2450 MHz hasta 250 kbps.

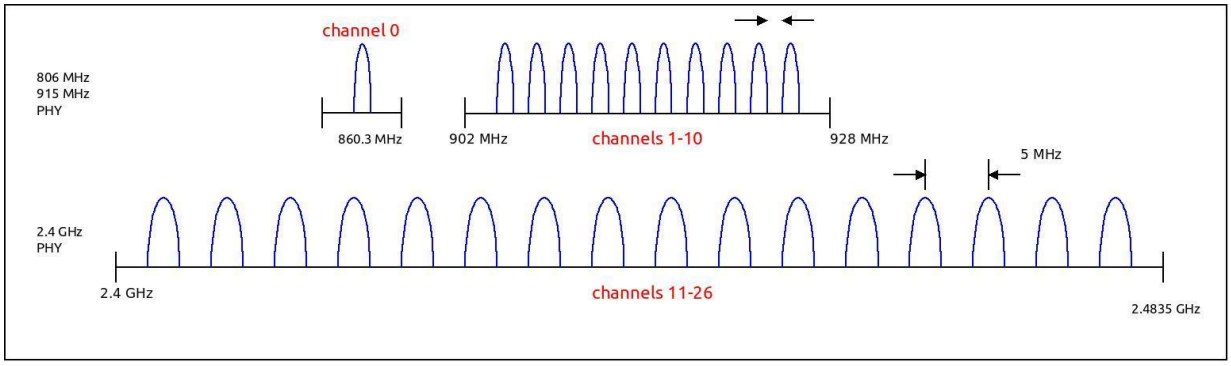

***Figura 2.1**: IEEE 802.15.4 distribución de canales (2003) [4]*

Existe la posibilidad de que trabajen varias redes inalámbricas en la misma banda de frecuencia, por lo cual se establece una asignación dinámica de canales.

- La capa MAC estable algoritmos de búsqueda para la selección del mejor canal.

- La capa física PHY establece algunas funciones para la detección de energía, considerando la calidad del canal y la comunicación del canal.

La gráfica 2.2 muestra cómo las redes de sensores pueden coexistir con redes WI-FI (802.11) sin crear interferencia. Los canales que se pueden usar para IEEE 802.15.4 en 2.4 GHz son: 15, 20, 25, 26

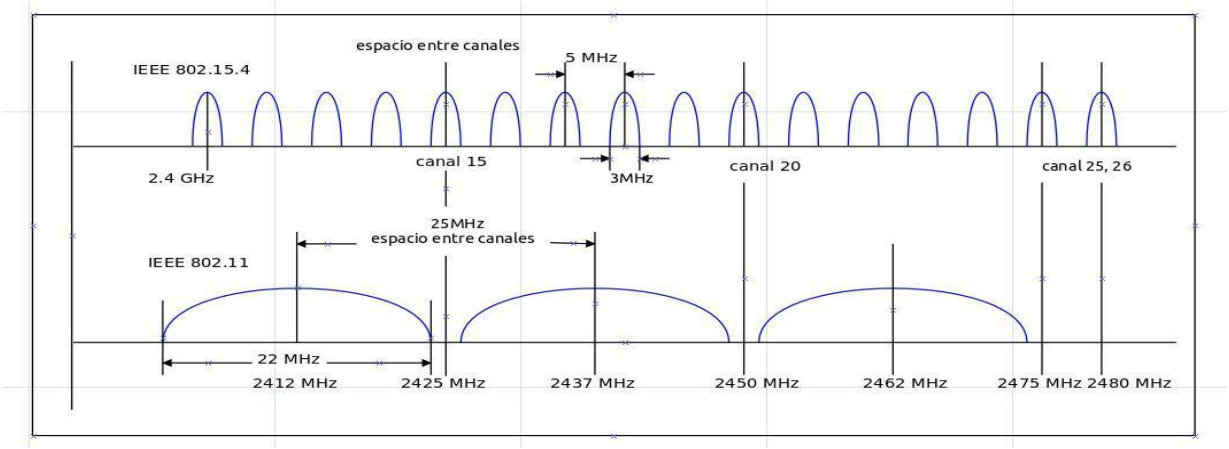

***Figura 2.2**: Canales IEEE 802.15.4 y IEEE 802.11. [4]*

2.3.1 CAPA MAC

El control de acceso al medio (MAC) transmite tramas MAC usando el canal físico. Además del servicio de datos, ofrece una interfaz de control que regula el acceso al canal físico y a la señalización de la red, también controla la validación de tramas y la asociación entre nodos, y garantiza slots de tiempo [11]. Por último, ofrece puntos de enganche para servicios seguros. El tamaño máximo de paquetes en 802.15.4 es de 127 bytes. Los paquetes son pequeños ya que IEEE 802.15.4 está diseñado para dispositivos que trabajan con velocidades reducidas, La capa MAC agrega un encabezado a cada paquete; la cantidad de datos disponibles para un protocolo de capa superior o de una aplicación está entre 96 y 116 bytes. Por lo tanto los protocolos de capas superiores a menudo añaden mecanismos para fragmentar grandes paquetes de datos en múltiples tramas 802.15.4.

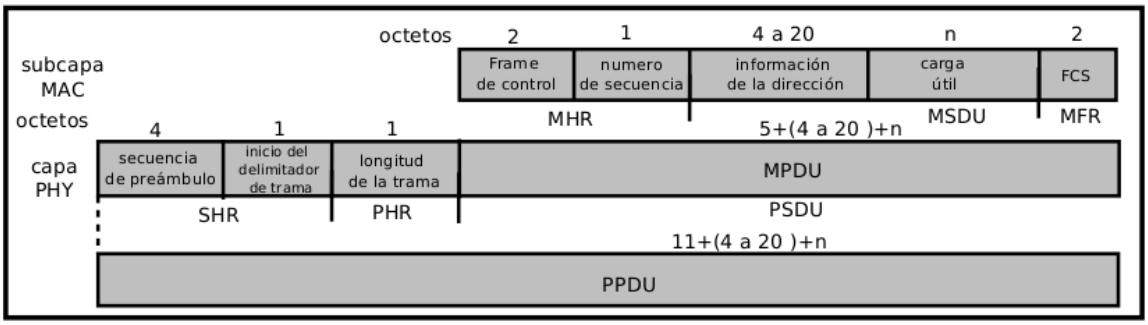

***Figura 2.3**: Capa física y subcapa MAC [4]*

La figura 2.3 muestra la capa PHY y MAC que comparten una única estructura de paquetes, la MPDU (Media access control protocol data Unit) que se compone de la cabecera MAC (MHR), la unidad de servicios de datos (MSDU) y el pie de la Mac (MFR). La MHR está compuesta por 2 bytes para el control de la trama que especifican el tipo de trama, el formato de dirección y el número de secuencia. Un byte para el número de secuencia que coincide con el ACK de la transmisión anterior. El MSDU incluye un campo de dirección de tamaño variable entre 4 y 20 bytes, permitiendo que se puedan ingresar las direcciones de origen y destino. La MSDU contiene

el campo de la carga útil que es de longitud variable dependiente de los datos de información que se establezcan, siempre y cuando la MPDU no sobrepase los 127 bytes. El MFR es un campo de secuencia de verificación de trama que es básicamente un código de detección de errores CRC de 16 bits.

La unidad de datos de la capa física (PPDU) está compuesta de un preámbulo o encabezado de sincronización de 32 bits, una cabecera de la capa física para la longitud del paquete y posteriormente la carga útil.

2.3.3 NODOS

Básicamente existen dos tipos de nodos Full Function Devices (FFD) y Reduced Function Devices (RFD) [13].

- FFD: puede servir como un coordinador PAN o como un nodo común. Implementa un modelo general que le permite comunicarse con cualquier otro dispositivo de la red, permitiéndole transmitir y recibir mensajes entre los dispositivos, trabajaría como coordinador de la red (coordinador PAN cuando está a cargo de toda la red). También se caracteriza por tener implementado todo el stack de protocolos.

- RFD: Destinados a ser dispositivos muy simples con muy limitados recursos de gestión y requisitos de comunicación. Sólo pueden comunicarse con FFD y nunca pueden actuar como coordinadores.

2.4 ZIGBEE

Es el nombre de la especificación de un conjunto de protocolos de alto nivel de comunicación inalámbrica [14], para la utilización de radio difusión digital de bajo consumo, basada en el estándar de comunicación IEEE 802.15.4. De redes inalámbricas de área personal. Su objetivo son las aplicaciones de sistemas embebidos, que requieren comunicaciones seguras con baja tasa de transferencia de envío de datos y ahorro de energía de las baterías.

2.5 6LoWPAN

6LoWPAN (IPv6 Over Low Power Wireless Personal Area Network)[15] es un estándar creado en el 2005, que permite el uso de IPv6 sobre redes basadas en el estándar IEEE 802.15.4. Está conformado por una colección de estándares definidos en el (RFC 4919) por The Internet Engineering Task Force (IETF), posibilita que los dispositivos de una red inalámbrica de área personal se puedan conectar por IP, lo cual lo hace compatible con modelo OSI [16] figura 2.4.

***Figura 2.4**: 6LoWPAN y modelo OSI*

2.5.1 CARACTERISTICAS

Las características más importantes de 6LoWPAN son:

- Bajos anchos de banda con tasas de transferencia de hasta 259Kbps, 40 Kbps y 20Kbps definida previamente por la capa física.
- Pequeño tamaño de paquetes: Estimando un PHY máximo de 127 bytes y en los casos de existir sobrecarga por las capas anteriores sobrará únicamente 81bytes para datos.
- Soporta direcciones MAC de 16 bits cortas o 64 bits extendidos.
- Posibilita la configuración de las topologías de malla o estrella.
- Estos dispositivos trabajan con bajo consumo de energía y su construcción es de bajo costo.
- Soporta un gran número de dispositivos conectados a la red.
- La localización de estos dispositivos no está predefinida en algunos casos debido a que pueden cambiar de posición dinámicamente.
- Los dispositivos en una red 6LoWPAN suelen estar largos periodos de tiempo en modo de hibernación (Sleep Mode) para ahorrar energía.

2.5.2 PILA DE PROTOCOLOS

La pila de protocolo 6LoWPAN es muy similar a la pila IPv6 figura 2.5, lo que corresponde a las 4 capas de modelo de Internet, con algunas diferencias en la capa física y de enlace. El protocolo 6LoWPAN es compatible con IPv6, para el cual se ha definido una pequeña capa de adaptación para permitir conexiones IPv6 sobre IEEE802.15.4 (RFC4944). Para implementaciones prácticas en dispositivos embebidos, se implementa la capa de adaptación junto con IPv6, de tal manera que puede ser mostrado como parte de la capa de red. El protocolo de transporte que regularmente se usa en 6LoWPAN es UDP (RFC 0768), que puede ser comprimido por LOWPAN. El protocolo de transporte TCP no es comúnmente usado con 6LoWPAN por razones de rendimiento, eficiencia y complejidad.
El protocolo de control de mensajes de Internet versión 6 ICMPv6 (RFC 4443), es usado para mensajes de control como eco ICMP, reconocimiento de vecinos y detección de destinos inalcanzables. La adaptación de IPv6 y el formato 6LoWPAN se lleva a cabo en los routers de

borde, los cuales se encuentran comunicando con las islas 6LoWPAN. Esta transformación es transparente para el protocolo IPv6.

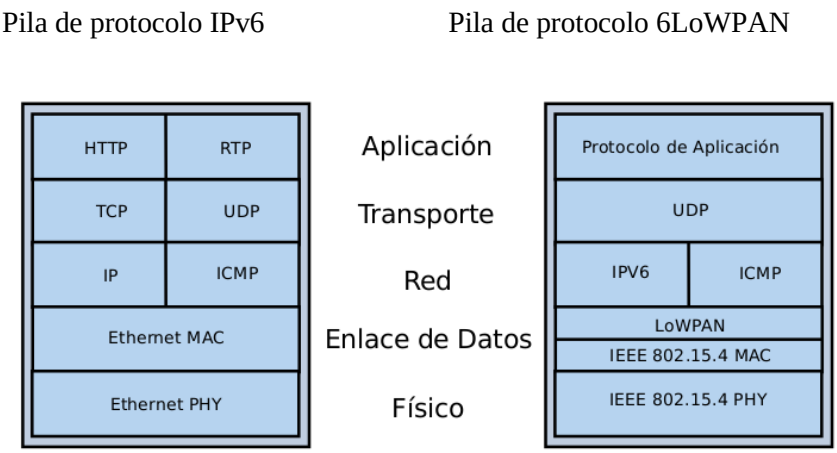

***Figura 2.5**: Pila de protocolo IPv6 y 6loWPAN*

2.5.2 TOPOLOGÍA DE LAS REDES

La figura 2.6 muestra las topologías soportadas por 6LoWPAN son anillo, malla y estrella. Para la construcción de topologías Multi-hop necesariamente los dispositivos deben estar configurados como FFD, los cuales permiten establecer conexiones usando algoritmos de enrutamiento posibilitando la transmisión de información por diferentes rutas.

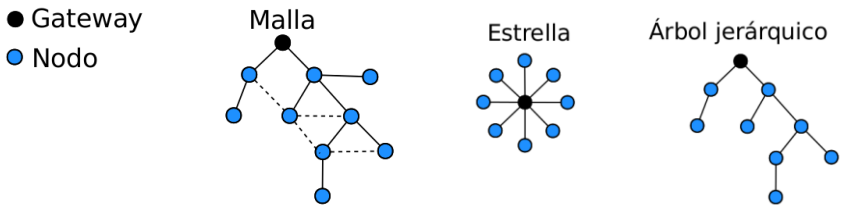

***Figura 2.6**: Topologías de red*

2.5.3 FUNCIONALIDAD

ADAPTACÍON DE PAQUETES El tamaño mínimo de paquetes de datos en IPv6 es de 1280 bytes en comparación con 6LoWPAN es de 81 bytes. Ello implica construir una capa de adaptación en la capa IP para fragmentación y re ensamble de paquetes.

COMPRESIÓN DE CABECERA Al con 81 bytes de encabezado y 40 bytes de IPv6, solo quedan 41 bytes para las capas superiores del modelo OSI como es UDP. UDP hace uso de 8 bytes, fragmentación y re ensamble de paquetes y algunos bytes más, lo cual deja pocos bytes para datos, de tal manera es necesario la compresión de cabecera.

CANTIDAD DE DIRECCIONES El espacio de direcciones es de 128 bytes (2^{128}) que se asigna de manera jerárquica, tiene un encabezado de tamaño fijo, como también no permite la fragmentación en el camino, establece sistemas de difusión por Unicast, Multicast y Anycast (no Broadcast) , permite la configuraciones de direcciones Stateless y Stateful.

2.5.4 SEGURIDAD

La red 6LoWPAN por lo general requiere de confidencialidad y protección de integridad, que pueden implementarse en la capa de aplicación, de red o de enlace. La elección puede influir por sus restricciones en tamaño de código, ancho de banda y baja potencia, Por lo general en la mayoría de los casos el uso de seguridad en capa de enlace se debe a que muchos de los equipos en IEEE 802.15.4 soportan seguridad AES-128[13]. Para realizar seguridad en la capa de red existen dos modelos; seguridad extremo a extremo o por limitación de red haciendo uso de firewall "cortafuegos". Para configuraciones con estrictas medidas de seguridad es recomendable implementar sistemas de seguridad extremo a extremo.

2.6 IPv6

IPv6 (Internet Protocol versión 6) por su siglas en ingles se define en el (RFC 2460). Diseñado por Steve Deering de Xerox PARC y Craig Mudge, se encarga de direccionar paquetes en redes de información como también en la Internet. La creación de IPv6 se realizó para sustituir la versión IPv4, que se encuentra limitada en el número de direcciones para posibles redes. IPv4 permite 4.294.967.296 (2^{32}) direcciones de host diferentes, un número inadecuado en la actualidad para dar una dirección a cada persona del planeta y mucho menos a cada vehículo, teléfono, PDA, etc. En cambio, IPv6 admite 340.282.366.920.938.463.463.374.607.431.768.211.456 (2^{128} o 340 sextillones de direcciones). Esta expansión proporciona flexibilidad en la asignación de direcciones y el enrutamiento del tráfico, eliminando la necesidad primaria de traducción de direcciones de red (NAT, Network Address Traslation), la figura 2.7 muestra el formato de una cabecera IPv6 estándar.

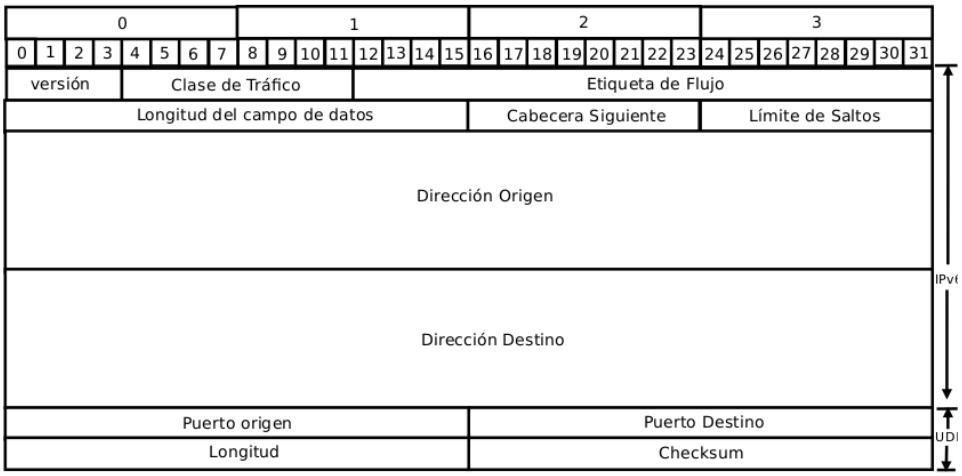

***Figura 2.7**: Encabezado Estándar IPv6/UDP (48 bytes)*

2.6.1 DIRECIONAMIENTO

Una dirección IPv6 tiene un tamaño de ocho campos de 16-bits cada uno de ellos unido por dos puntos. Cada campo debe contener un número hexadecimal, a diferencia de la notación decimal con puntos de las direcciones IPv4. En la figura 2.8 se representa el formato de una dirección IPv6.

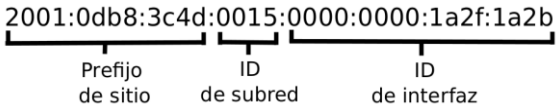

***Figura 2.8**: Dirección IPV6*

Los tres campos que están más a la izquierda (48 bits) contienen el prefijo del sitio. El prefijo describe la topología pública que el ISP o el RIR (Regional Internet Registry)

El campo siguiente lo ocupa el ID de subred de 16 bits que se asigna al sitio. El ID de subred describe la topología privada, denominada también topología del sitio.

Los cuatro campos situados más a la derecha (64 bits) contienen el ID de interfaz, también denominado token. El ID de interfaz se configura automáticamente desde la dirección MAC de interfaz o manualmente en formato EUI-64 (Global Identifier -IEEE).

Normalmente una dirección IPv6 no llega a alcanzar el tamaño máximo de 128-bits, lo cual implica ser rellenados con ceros. Los ceros a la izquierda de un grupo se pueden omitir, también IPv6 permite utilizar la notación de dos puntos consecutivos (::) para reemplazar campos consecutivos de 16-bits de ceros; esto es realizable solo una vez por dirección. La figura 2.6 muestra la dirección IPv6 aplicando la notación de reducción sintáctica.

2001:0db8:3c4d:15::1a2f:1a2b

***Figura 2.9**: Sintaxis reducida de una dirección IPV6*

2.7 6LoWPAN E IPV6

las figuras 2.10 representa el formato de una trama 6loWPAN, tal como lo representa la figura 2.11 el formato de trama IPv6, las cuales presentan ciertas con la particularidad de que la capa de adaptación 6LoWPAN de encarga comprimir el encabezado IP.

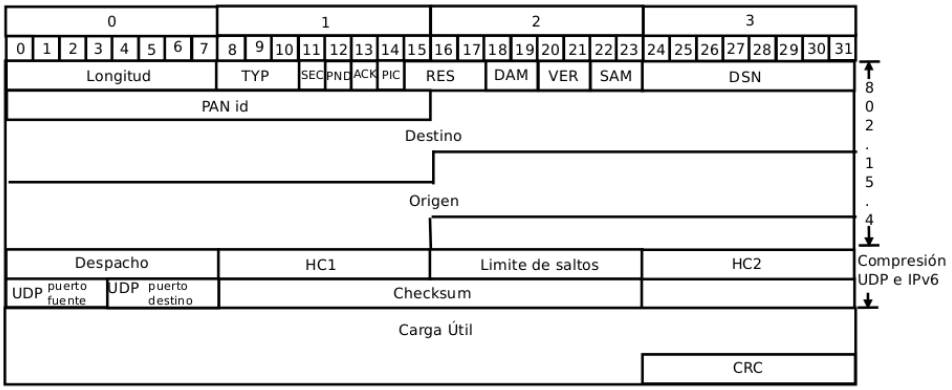

***Figura 2.10**: Trama 6LoWPAN [17]*

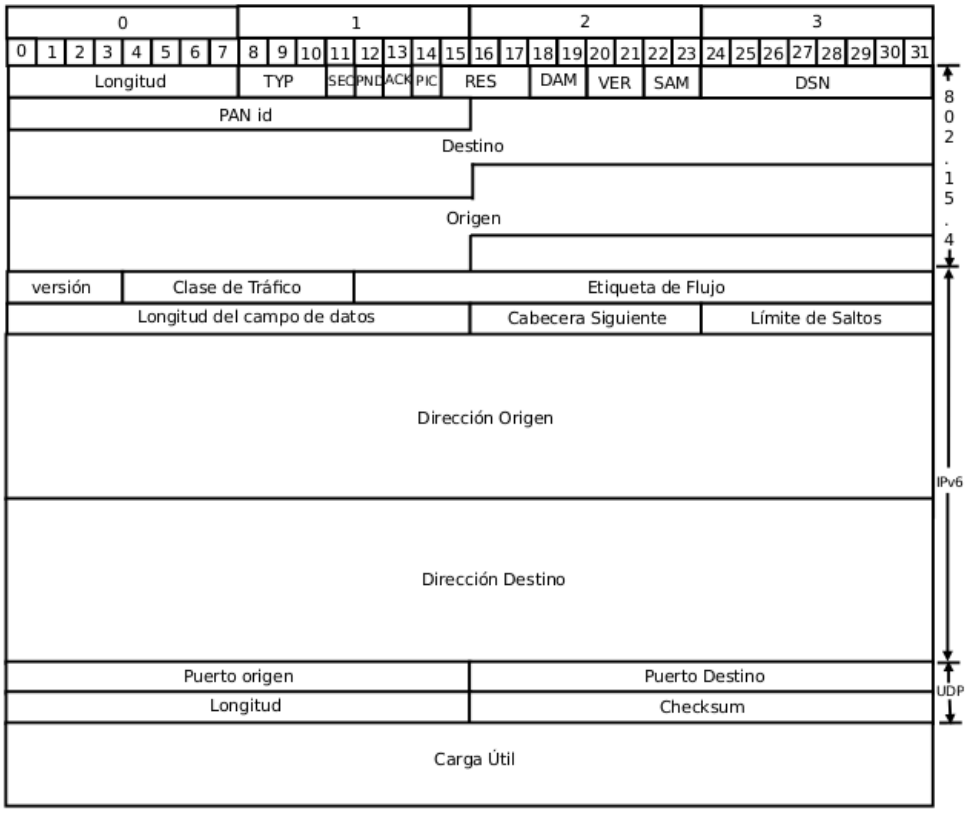

***Figura 2.11**: Trama IPV6 [17]*

2.8 CAPA DE ADAPTACIÓN

DIRERECCIONAMIENTO LOWPAN IEEE 802.15.4 define cuatro tipos de tramas: tramas de señalización, tramas comando MAC, tramas ACK y tramas de datos; también establece varios modos de direccionamiento como direcciones extendidas IEEE 64-bit y direcciones cortas de 16-bits. Los paquetes IPv6 se deben transportar en tramas de datos, y para su uso dentro de 6LoWPAN, se aceptan ambas direcciones. Una dirección corta de 16-bits básicamente es el número de identificación dentro del PAN; normalmente se asigna por el coordinador PAN durante eventos de asociación. Su validez y singularidad está determinada por el tiempo de vida de dicha asociación. La dirección de 64-bits es conocida como la dirección IEEE 802.15.4 o identificador de interfaz; se puede definir una dirección corta de 16-bits rellenada con ceros para ser vista como una de 64-bits.
IPv6 implementa tramas multicast (uno a muchos) pero no boadcast, en IEEE 802.15.4 no se permite la transmisión de tramas multicast de forma nativa, por lo cual los mensajes multicast deben ser transformados en tramas de difusión de capa de enlace en redes conformadas por IEEE 802.15.4.

Para la asignación de enlaces locales se usa el procedimiento de *Stateless Address Autoconfiguration* (SSA) para IPv6 conformado por un prefijo (64 bits) y un sufijo (64 bits).

El prefijo indica el alcance de una dirección ya sea tipo de enlace Local (prefijo fe80::/64) o enlace global (prefijo. Router Advertisement-Router Solicitation). El sufijo EUI64-bit (identificador Global IEEE) inserta FF:FE en el medio de la dirección, como por ejemplo: fe80::FF:FE:00:1.

TAMAÑO DE PAQUETES El tamaño de la unidad de transmisión máxima (MTU) de paquetes IPv6 es 1280 bytes, de tal manera un paquete completo IPv6 no cabe normalmente en una trama IEEE 802.15.4. Estimando que el máximo tamaño del paquete en la capa física es de 127 bytes y un máximo de 25 bytes de sobrecarga, se dice que:
127 bytes del tamaño del paquete menos 25 bytes resulta una trama de tamaño de 102 bytes, a esto la capa de enlace adiciona una sobrecarga de 21 bytes en el peor de los casos, lo que deja sólo 81 bytes disponibles. Además, la cabecera IPv6 es de 40 bytes de longitud, lo que implica que el tamaño del paquete de 81 bytes menos una sobrecarga de 40 bytes de IPv6 dejaría 41 bytes disponibles para los protocolos de capa superior como UDP y TCP. El protocolo UDP utiliza 8 bytes en la cabecera, lo que deja 33 bytes de datos para la aplicación en el peor de los casos. Esto señala la necesidad de crear una capa de adaptación para fragmentación y el re-ensamblaje, la creación de esta capa de adaptación incrementará la sobrecarga de bytes.

De lo mencionado anteriormente aparecen algunas consideraciones como: que la capa de adaptación debe cumplir con los requisitos mínimos para una MTU IPv6. Sin embargo, se espera que las pequeñas cargas útiles de aplicación con una adecuada compresión de cabecera produzcan paquetes que se ajusten a una sola trama IEEE 802.15.4. En el cálculo anteriormente realizado se presenta el escenario del peor de los casos, lo que implica que es casi inevitable la compresión de la cabecera, pero no necesariamente las aplicaciones IPv6 sobre IEEE 802.15.4 harán uso de la capa de adaptación, solo para el caso en que la carga útil sea superior a los 33 bytes.

ENCAPSULAMIENTO LoWPAN Las tramas encapsuladas es la carga útil en la MAC "Protocol data Unit" IEEE 802.15.4. La carga útil LOWPAN un paquete IPv6 presenta esta cabecera de encapsulación. Todos los datagramas encapsulados LOWPAN transportados por IEEE 802.15.4 se encuentran prefijados por una pila de cabecera de encapsulación, cada encabezado de la pila tiene un tipo de encabezado y campo de encabezado. Cuando se usa más de un encabezado LOWPAN en el mismo paquete se establece el orden de: encabezado de direccionamiento de red, Encabezado broadcast/multicast (dsp) y Compresión de cabecera (HC1), 6LoWPAN define HC1, como un esquema de compresión optimizado para comunicaciones IPv6. Con HC1 se puede reducir la cabecera IPv6 de 40 bytes a 2 byte. En cuanto a la capa de transporte, existe un bit en HC1 para indicar que se aplicará compresión en esta capa. La capa UDP también permite compresión, mediante la misma técnica de compresión de IPv6: no repitiendo información que se encuentra en otro lado. Su cabecera es HCUDP. El checksum es el único que debe cargarse completo. Permitiendo la reducción de UDP de 8 bytes a 4 bytes.

FRAGMENTACION si un datagrama con carga útil se ajusta dentro de un trama 802.15.4, este no será fragmentado y su encapsulación LOWPAN no contendrá una cabecera de fragmentación, por otro lado si un datagrama no encaja dentro de un trama 802.15.4 este se dividirá en fragmentos de enlace, figura 2.12. Como la fragmentación solo se puede expresar en múltiplos de 8 bytes, todos los fragmentos de enlace excepto el último deben ser múltiplos de 8 bytes de longitud.

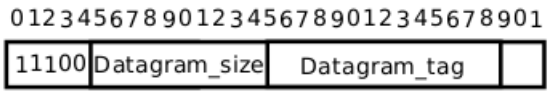

Figura 2.12*: Primer Fragmento*

Tanto los siguientes fragmentos como el último, contiene una cabecera de fragmentación que se ajustar al formato figura 2.13:

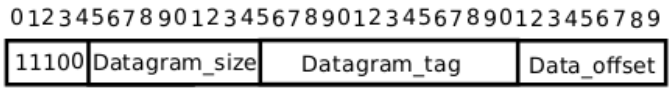

Figura 2.13*: Fragmento Subsecuente*

Datagram_size (11 bits): tamaño de todo el paquete IP antes de la fragmentación de capa de enlace. Este valor es el mismo para todos los fragmentos realizados.

Datagram_tag (16 bits): el emisor debe incrementar la etiqueta datagrama para datagramas fragmentados sucesivos. Este valor debe ser el mismo para todos los fragmentos de una carga útil.

Datagram_offset (8 bits): sólo en el segundo y siguientes fragmentos y en incrementos de 8 bytes.
La figura 11 es una vista gráfica de los diferentes encabezados que puede tener un paquete UDP enviado a través de una red 6LoWPAN.

La figura 2.14 muestra el diferente encabezado que puede tener un paquete UDP envidado sobre un enlace 6LoWPAN.

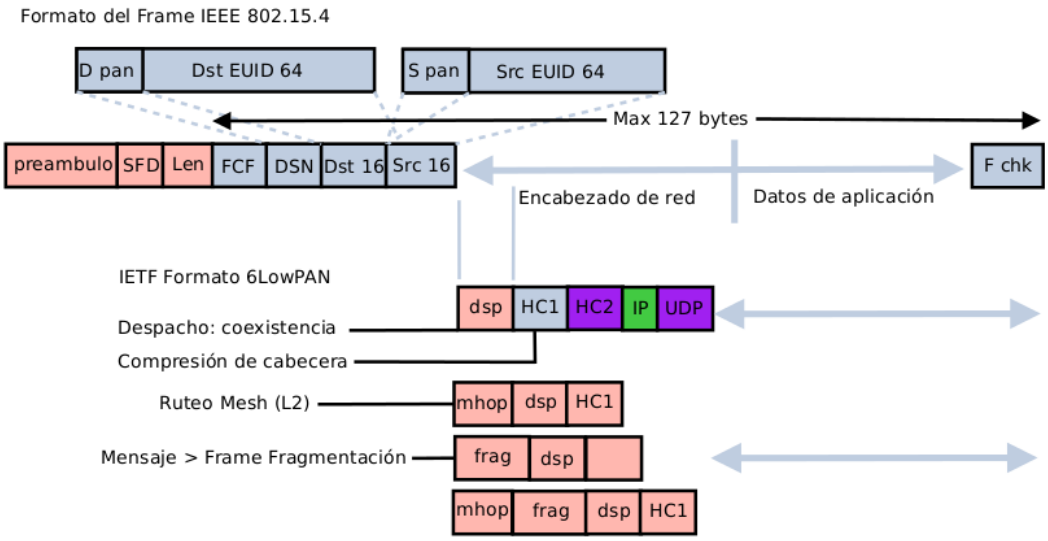

***Figura 2.14**: 6LoWPAN –Compresión UDP [18]*

CAPÍTULO 3

HARDWARE

La plataforma disponible durante el proyecto fue AVR RAVEN, desarrollada por la empresa ATMEL y se utiliza en aplicaciones de redes de sensores de bajo consumo. Lleva integrado un radio, antena y micro controlador. Los datos que se presentan a continuación son suministrados por el fabricante [19], figura 3.1:

Figura 3.1*: Nodo Sensores y Jakdaw (usbstick) [19]*

3.1 COMPONENTES DEL HARDWARE

El diagrama de bloque para AVR-RAVEN es representado a continuación por la figura 3.2.

AVR-RAVEN

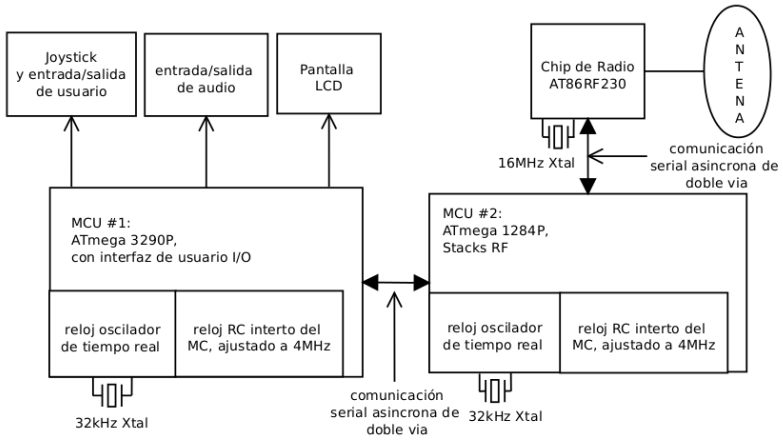

***Figura 3.2**: Diagrama de bloques tarjeta AVR-RAVEN*

3.1.1 MICROCONTROLADOR 1284P

Es un microcontrolador perteneciente a la empresa ATMEL de la familia AVR de 8 bits que combina una memoria flash de 128KB de programa con capacidad de lectura y escritura, una EEPROM de 4 KB y SRAM de 16 KB, 32 registros de propósito general, un contador de tiempo real, tres temporizadores que funcionan en modo de comparación y como PWM, 2 USART, Comunicación serial con Baud Rate de hasta 57600 bps, 8 canales A/D diferenciales con 10 bits de resolución, Watchdog programable con oscilador interno, puerto serie SPI con interfaz JTAG para realizar la programación y depuración del chip y seis modos de programación para condiciones de ahorro de energía[19]. El dispositivo funciona con voltajes entre 1.8 y 5 volts y está soportado para el Contiki OS [20].

3.1.2 MICROCONTROLADOR 3290P

Es un microcontrolador AVR con tecnología RISC encargado de hacer la gestión del LCD, manejo del joystick, y algunos periféricos de entrada y salida; consta de una memoria Flash 32 KB, 2 KB de SRAM y 1 KB EEPROM, 8 canales diferenciales con A/D de 10 bits de resolución [19].

3.1.3 USBSTICK (Jackdaw)

La USBstick figura 3.1 permite que el computador y las redes externas puedan comunicarse con los nodos integrados que trabajan con bajos recursos de energía (motas). Este dispositivo es creado por la empresa ATMEL el cual se encarga de percibir tráfico presente de la red inalámbrica. Este

dispositivo está diseñado para que sea compatible con múltiples sistemas operativos, el objetivo de esta interfaz es servir como medio de comunicación entre la red de sensores y el servidor emulando una conexión Ethernet [19].
Para la instalación de los controladores la cantidad de almacenamiento es muy limitada en torno a 59 Kbytes, suficiente para almacenar los archivos del controlador.

El diagrama de bloque para el USBstick Jackdaw se presenta a continuación en la figura 3.3.

USBSTICK-JACKDAW

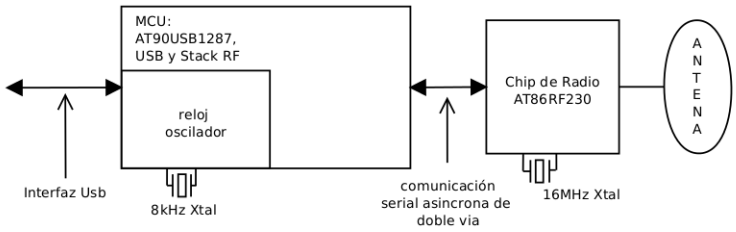

***Figura 3.3**: Diagrama de bloques tarjeta USBSTCK JACKDAW*

3.1.4 TRANSCEIVER AT86RF230

El AT86RF230 es un transceptor de radio de 2,4 GHz que se adapta a una amplia gama de aplicaciones inalámbricas. La comunicación se establece bajo IEEE.802.15.4. Trabaja con limitados recursos de energía; su consumo en modo Sleep es 20nA. En modo RX 15,5mA, y en modo TX 16,5mA. Se encuentra conectado vía SPI con el micro controlador 1284p en el cual se encuentra implementado Contiki OS, también la USBSTICK hace uso de este componente [19]. El voltaje de alimentación establecido es de 3.3v.

3.1.5 ALIMENTACIÓN – BATERÍAS

Para la alimentación de las tarjetas (motas) utiliza dos baterías LR44, consta de un valor nominal de 1.55V y 150mAh, adicional a esto presenta la posibilidad de alimentar el circuito mediante una fuente de alimentación externa, permitiendo el uso de fuentes de alimentación alternativas [21]. Ya sea el uso de pequeños paneles solares u otro tipo de sistemas de micro generación, siempre llevando un enfoque para que los sistemas sean energéticamente sostenibles, para el caso se utiliza una fuente de 6V para realizar pruebas.

3.1.6 SENSORES

El diagrama de bloque para NODO SENSOR se presenta a continuación en la figura 3.4.

NODO SENSOR

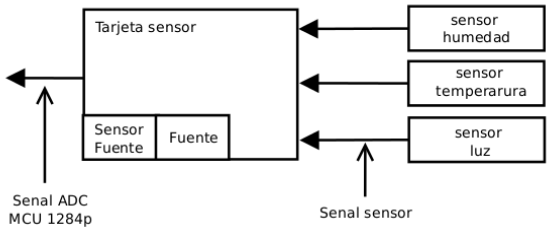

***Figura 3.4**: Diagrama de bloques tarjeta nodo sensor*

Cuatro sensores son usados en cada nodo. Estos se encargan de medir el voltaje de las baterías, intensidad de luz, temperatura y humedad. Los sensores son acondicionados con circuitos seguidores de voltaje hechos con amplificadores operacionales, esta configuración de circuito se usa como un buffer el cual permite mitigar los efectos de carga y adaptar las impedancias de entrada y salida, lo cual hace que el circuito sea ideal para sensores con bajas intensidades de medición.

MEDICIÓN DEL VOLTAJE. En la Figura 3.5 se presenta el diagrama utilizado para medir el voltaje de la fuente. La capturar de información en la tarjeta se implementa un circuito divisor de tensión para medir el voltaje de la fuente entre 0 y 6v. Se conoce que las resistencias son componentes lineales lo cual asegura la captura real de la medición.

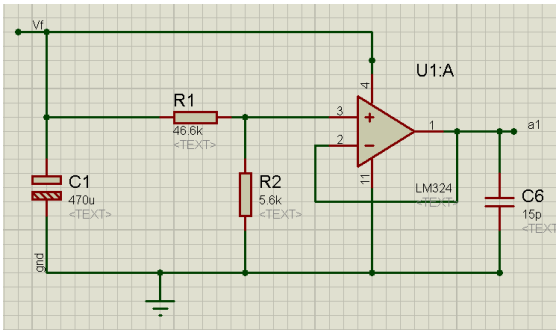

***Figura 3.5**: Acondicionamiento se señal para captura del voltaje de la fuente*

Voltaje de salida del amplificador no inversor:

$$Vo = \frac{R1}{R1+R2} V_f \qquad (3.1)$$

Para un voltaje de 6.3V presenta:

Voltaje de Salida = ((3.3*ADC[1]/1023)*(9.67)) (V) (3.2)

SENSOR DE LUZ. La figura 3.6 presenta el acondicionamiento de señal para el sensor de luz, este utiliza una fotorresistencia dependiente de la luz (LDR)[22], la cual se comporta como una resistencia de 2,5 kΩ a luz brillante y 500 KΩ en la oscuridad, su acondicionamiento de señal se implementó con un circuito divisor de voltaje el cual mide el voltaje de salida que se presenta en el circuito.

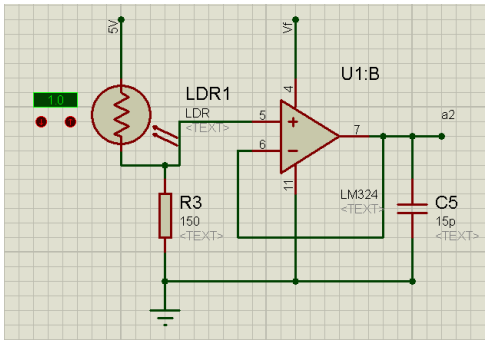

***Figura 3.6**: Acondicionamiento de circuito LDR*

Para la LDR se usa una configuración de divisor de tensión, el voltaje a través de la 150 Ω generados por la fotorresistencia da la lectura de la intensidad de luz. El buffer es usado para evitar cualquier carga y así conseguir el valor mostrado por el divisor de tensión.

La función de transferencia es:

$$Vo = \frac{R3}{R3+R_{LDR}} V_{5V+} \qquad (3.3)$$

Donde ADC[2] = *Vo*,

Voltaje de Salida =((3.3*ADC[2]/1023)*1000) (mV) (3.4)

SENSOR DE TEMPERATURA. La figura 3.7 presenta el acondicionamiento de señal para la del sensor de temperatura, este sensor captura la información del entorno. Para el sensor de temperatura se hace uso de un sensor LM35[23] el cual presenta un voltaje de salida reaccionando a los cambios de temperatura presentados en el ambiente. Se implementa un circuito acondicionador de baja señal para medir el voltaje de salida que presenta el circuito.

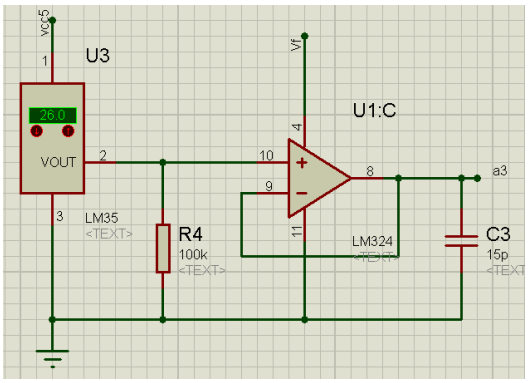

***Figura 3.7**: Acondicionamiento del circuito LM35*

$$Vout = V_o \qquad (3.5)$$

Donde ADC[3] = *Vo*,

Dato de Temperatura = ((3.3*ADC[3]/1023)*100) (Cº) (3.6)

SENSOR DE HUMEDAD. La figura 3.8 presenta el acondicionamiento de señal para el sensor de Humedad, se utiliza el sensor de Humedad HIH-4000-002[24]. La salida lineal del sensor permite establecer conexión directa con el microcontrolador, pero el rango máximo del ADC es de 1.6 V por lo que el sensor necesita una amplificador operacional para ser reducido antes de conectar al ADC, la escala se reduce mediante un divisor de tensión y luego un buffer para separar el circuito del microcontrolador. El gráfico de entrada y salida del sensor muestra una relación lineal. El voltaje de polarización para el sensor es de 5V.

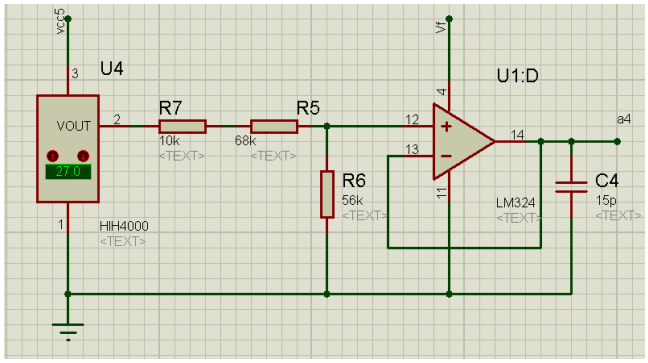

***Figura 3.8**: Acondicionamiento del circuito para el sensor HIH 4000-002*

La relación de la salida del sensor y la entrada del ADC es:

$$Vo = \frac{R3}{R3+R_{LDR}} V_{5V+} \qquad (3.7)$$

Ecuación de Linealización [24],

$$f(x) = 0.032X + 0.7439 \text{ , donde } ADC[4] = Vo \qquad (3.8)$$

Dato de salida = (((3.3*(1.1*ADC[4])/(0.5822*1023)-0.7439))/0.032) (RH) (3.9)

FUENTE DE PODER. La figura 3.9 presenta el acondicionamiento de señal para la fuente de poder. Para que funcione el nodo sensor y el microcontrolador de forma independiente, se realiza una arreglo de 4 baterías AA de +1.5V para generar 6V, el amplificador operacional y la AVR_RAVEN son alimentados directamente con 6 V y con el regulador de voltaje se convierte a 4,5V para alimentar a los sensores de luz, temperatura y humedad.

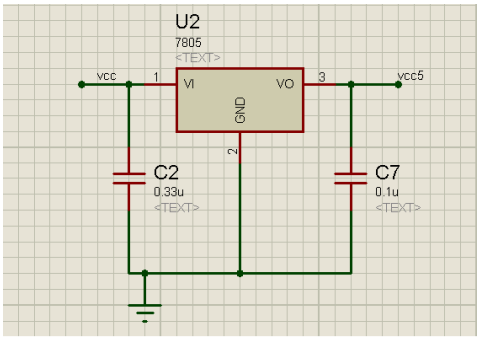

***Figura 3.9**: Fuente de Poder*

CAPÍTULO 4

SOFTWARE

En este capítulo se presentan algunos sistemas operativos que en la actualidad se encuentran diseñados para redes de sensores, luego se explica las características y funcionamiento del sistema operativo de elección con sus herramientas de programación, la captura de los datos obtenidos por los sensores y el software necesario para la configuración e inserción de los datos a un servidor web.

4.1 ELECCIÓN DEL SISTEMA OPERATIVO

Los nodos sensores son mecanismos de bajos recursos de procesamiento, memoria y baja demanda de energía, lo que los convierte en sistemas con característica particulares comparados con los PC, portátiles, dispositivos móviles (Celulares, SmartPhone, Tablets), etc. De tal manera se selecciona un sistema operativo (SO) como mecanismo de gestión de recursos que se ajuste a las exigencias que los nodos sensores presentan respecto a gastos de energía. Estos son algunos OS que en la actualidad realizan este tipo de tareas:

TinyOS[25]: Es un OS de código abierto orientado a las WSN. Está escrito en lenguaje de programación nesC compuesto de tareas y procesos que colaboran entre sí. Está diseñado para que funcione con limitados recursos de memoria, este sistema operativo contiene algoritmos que posibilitan enrutamientos y aplicaciones preestablecidas para ciertas plataformas específicas.

FReeRTOS[26]: Es un OS de tiempo real con licencia de código abierto " General Public License" (GPL) [27] con la cualidad de que funciona en sistemas embebidos, escrito en lenguaje de programación C. Diseñado para ocupar poca memoria con un funcionamiento apropiable y cooperativo con posibilidad de asignación de prioridades a las tareas de una forma flexible, utiliza el manejo de colas para el paso de mensajes entre tareas, usa semáforos binarios recursivos, de conteo y de exclusión mutua, posee una pila uIP TCP/IP ligera para aplicaciones en WSN.

ARCH ROCK [28]: Es una empresa privada de CISCO encargada de construir tecnología basada en el protocolo de Internet, uno de sus productos es un OS diseñado para aplicaciones de WSN escrito en lenguaje de programación C, se utiliza en aplicaciones de vigilancia medio ambiental, seguimiento, logística, automatización industrial de control.

CONTIKI [29] **:** Es un OS de código abierto multitarea, desarrollado para plataformas con recursos limitados de procesamiento, energía y memoria. Escrito en lenguaje de programación C, desarrollado por el grupo de investigación "Swedish Institute of Computer Science" liderado por Adam Dunkels [30]. Contiki consiste en un núcleo orientado a eventos el cual realiza peticiones por medio de protothread [31], estos programas son cargados y descargados dinámicamente del stack de tareas. El sistema operativo está pensado para el "Internet of things" que establece como objetivo principal la conexión de objetos de forma inalámbrica.

eCos[32]: "Embedded Configurable Operating System" es un sistema operativo de tiempo real de código abierto diseñado para sistemas embebidos que solo necesitan ejecutar un proceso, puede ser personalizado para que ejecute cualquier tarea. Ofreciendo tareas de ejecución en tiempo real, con propiedades de minimizar los requerimientos del hardware.

4.2 CONTIKI

La elección del Contiki OS se establece a criterios relacionados como lenguaje de programación, documentación, herramientas de simulación, estilo de programación, compatibilidad con la arquitectura que se dispone y políticas de distribución.

El OS se encuentra disponible para las plataformas AVR-Raven de la empresa ATMEL, además posee un paquete de herramientas de simulación que facilita al programador la visualización del comportamiento de la red, compilación y depuración de los programas.

El sistema operativo dispone de dos stack o pilas de comunicación, una versión TCP/IP adaptada para redes de sensores inalámbrica y la stack Rime [33] que es un stack modular, estructurado en capas muy simples, de manera que los encabezados sean lo más pequeños posibles.

La pila de protocolo de Contiki contiene el driver de radio encargado de enviar y recibir datos por aire. En un nivel superior se encuentra la capa RDC (Radio Duty Cycle) [34] la cual se encarga de prender y apagar la radio para ahorrar energía. Sobre esta se encuentra la de acceso al medio (MAC), esta realiza el trabajo de evitar las colisiones entre las transmisiones hecha por los nodos. Luego se encuentra la capa de red, que es la encargada del enrutamiento de los paquetes. Por último se encuentra la capa de aplicación, que es donde se interpretan y procesan los datos capturados.

El OS suministra una serie de librerías diseñadas para diversas arquitecturas, entre ellas la AVR-Raven. El driver de radio se encarga de administrar el AT86RF230 de ATMEL, este driver permite gestionar los recursos de encendido, apagado, transmisión y recepción de datos, el mecanismo que utiliza para la capa RDC es ContikiMAC [34] este permite apagar la radio la mayor cantidad de tiempo posible ya que la radio consume mucha energía, por lo que este es un recurso limitado.

El mecanismo que utiliza la capa MAC es CSMA-CA (Carrier Sense Multiple Access-Collision Avoidance)[35]. Antes de transmitir se escucha el canal para saber si hay alguien transmitiendo y de esa manera evitar colisiones para luego realizar la transmisión del dato, finalmente se espera el dato de confirmación del receptor.

El mecanismo que se utiliza para la capa de red es SICSlowpan [36], es una implementación realizada por los creadores de Contiki, la capa de adaptación 6LoWPAN se encarga de transmitir tramas IPv6 sobre IEEE 802.15.4 en redes de sensores inalámbricos.

CARACTERISTICAS

- Portabilidad: el OS puede ser portado a diferentes plataformas y CPU's. Su estructura le permite sacar el sistema base y multiplexarlo para manejar diversos sensores, servicios y aplicaciones para distintas estructuras.

- Configuración por IP: es un protocolo abierto que le da versatilidad al sistema, es escalable y estable, se usa a nivel mundial por miles de servidores, computadoras y dispositivos móviles.

- Programación por eventos: el sistema de gestión es administrado por eventos lo que conforma una sola pila para todos los procesos, la administración y comunicación de los procesos conocidos como protohilos (protothreads), permitiendo reducir el consumo de memoria RAM.

- Sistema ligero: Contiki por ser un sistema liviano puede ser portado en procesadores y máquinas obsoletas con éxito. Igualmente puede ser portado en microcontroladores de 8-bits, lo que permite conformar conexiones en red con bajos requerimientos de procesamiento, posibilitando establecer sistemas de comunicación más complejos.

- Conexión a Internet: al estar usando IPv6 los nodos pueden conectarse entre sí. De la misma manera permite que los nodos por medio de un router de borde puedan conectase a la red de Internet.

4.2.2 ESTRUCTURA DE CONTIKI

Dentro de directorio de Contiki se encuentran alojadas ciertas carpetas necesarias para el buen para la comprensión y el buen funcionamiento del sistema operativo como loe es herramientas, ejemplos, plataformas que soporta el sistema operativo, documentación, núcleo y demás archivos adicionales que brinda Contiki OS.

APPS: las aplicaciones son programas auxiliares que se pueden utilizar en el programa principal. Se tiene en cuenta que no es posible usar cualquier aplicación con cualquier protocolo. Tal es el caso de muchas aplicaciones que están diseñadas para el stack Rime y no para uIP.

CORE: Es el directorio en el cual se encuentra el núcleo del OS. En este directorio se puede encontrar la implementación de los protothreads, los etimers, uIP, Rime, RPL y todo lo que refiere al sistema operativo.

CPU: En este directorio se encuentra el código necesario del microcontrolador. Como es la configuración de los timers, módulo serial, memoria flash, instrucciones para que el microcontrolador entre y salga del low-power mode. Para la arquitectura Tamegua 1284p se utiliza la carpeta disponible para AVR, llegado el caso de migrar los programas a otra arquitectura simplemente se carga los drives de la carpeta de microcontrolador para nuevamente ser compilado.

DOC: Esta carpeta contiene todo lo que se refiere a la documentación del código.

EXAMPLE: En esta carpeta se encuentran los programas de los ejemplos que utilizan el sistema operativo Contiki.

PLATFORM: Esta carpeta contiene el código específico de ciertas plataformas presentes en el mercado. Como lo es caso de la plataforma AVR_RAVEN o la SKY, como también la

configuración de los botones, sensores y demás periféricos, además en esta carpeta se encuentran la rutina main de la arquitectura.

TOOLS: Esta carpeta contiene varias herramientas extras que no hacen parte del sistema operativo Contiki, pero son de vital importancia para la construcción y visualización de proyectos relacionados con las WSN como lo es el simulador COOJA [37].

4.3 FUNCIONAMIENTO DE CONTIKI

Contiki está conformado por un kernel, librerías y procesos, donde cada proceso puede estar constituido como un servicio o un aplicativo, Los procesos son definidos como funciones controladoras de eventos, todos los procesos comparten el mismo espacio de pila de dirección lo que no les permite correr en distintos dominios.

4.3.1 PROCESOS

Los procesos creados por los usuarios generalmente se desarrollan como tareas específicas. Los procesos se guardan en una lista encadenada, esto quiere decir solo es necesario conocer la dirección del primero, luego el primer proceso conoce la dirección del segundo, el segundo del tercero, y así sucesivamente, de esta manera se enlazan los procesos. Cada proceso tiene un protothread asociado, el cual se ejecuta cuando el proceso es llamado. Los estados de los procesos pueden ser desactivados, activados y llamados, un proceso está en estado llamado cuando se está corriendo un protothread. Esta activado cuando este puede ser llamado por un evento y esta desactivado cuando ningún evento puede despertar al protothread.

4.3.2 EVENTOS

Contiki funciona por eventos que se almacenan en una cola circular y son procesados por orden cronológico. Cada evento tiene configurado el proceso que va a despertar, puede enviarle datos, aparte de comunicarle cual fue el evento que lo despertó, los eventos pueden ser generados tanto por protothreads como en cualquier otra petición del programa, como es el caso de las interrupciones.

4.3.3 PROTOTHREADS

Un protothread son funciones que tiene la propiedad de ser capaz de esperar por eventos. Cada proceso tiene asociado un único protothread.

Existe un evento que inicializa los procesos. Cuando este evento llama a un proceso, se corre su protothread asociado desde el comienzo. El proceso pasa del estado desactivado al llamado. Cuando el protothread necesite esperar por un evento, retorna, y su proceso pasa del estado llamado a un estado activo. Luego cuando otro evento llame al proceso, se correrá el protothread desde donde se había retornado anteriormente. En donde se chequeará si el evento recibido es el esperado, en caso negativo vuelve a retornar en el mismo punto, en caso positivo se seguirá corriendo el protothread.

4.3.4 TIMERS

ETIMERS: Es un módulo ya implementado por Contiki para el manejo de temporizadores. Se basa en una lista encadenada de estructuras etimers. Estas estructuras guardan una marca de tiempo inicial, un intervalo de tiempo después del cual el temporizador expirará, un puntero al proceso que se va a despertar cuando el etimer expire, y un puntero al próximo etimer de la lista. También se tiene un proceso etimer, cuyo protothread se encargará de procesar las listas de estructuras etimers, este protothread lo que hace es recorrer la lista de etimers y fijarse si alguno expiró, y en cuyo caso se quita la estructura de la lista y se envía un evento.

CTIMERS: Los callback timers sirven para llamar a una función con un parámetro dado cuando el etimer expira. Cuando se reinicia el ctimer se elige el intervalo de tiempo. En este momento se llama a la función con el parámetro. Solo se podrán usar funciones de un solo parámetro con el ctimer.

RTIMER: Esta estructura representa una tarea en tiempo real y llama a una función en un momento preciso. Su funcionamiento se basa en la cuenta de ticks producidos por el oscilador de 32 kHz. Su utilización típica es la de medir intervalos de tiempo con precisión.

4.4 HERRAMIENTA DE SIMULACIÓN COOJA

Es una herramienta de simulación creada en java para Contiki aunque se puede correr otros sistemas operativos, diseñado para simular redes de sensores, la figura 4.1 muestra en los tres niveles en los cuales él trabaja como es código de máquina, sistema operativo y de red. La importancia de este simulador es poder estudiar diversas topologías de red, de una forma que se asemeje a la realidad con el fin de optimizar el ancho de banda y administrar el consumo de energía para casos reales. Para la construcción e interpretación de un proyecto con Cooja se presenta en el Anexo 1.

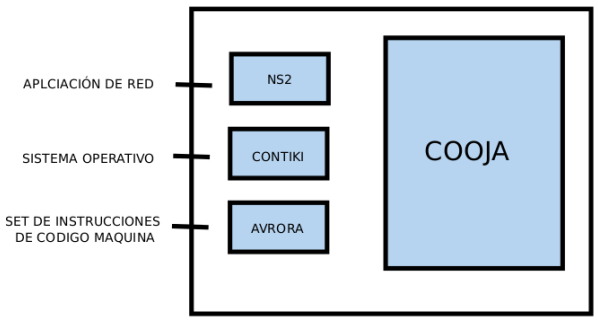

***Figura 4.1**: Niveles de simulación de COOJA*

4.5 ESTRUCTURA BÁSICA DE UN PROGRAMA EN CONTIKI

Los protothread son generados como macros de lenguaje C. Estos macros son sustituidos por el procesador con código C. Dependiendo del compilador se implementan los protothreads utilizando sentencias *goto* o *switch cases*. Por la estructura de gestión los creadores no recomiendan usar las sentencias *switch cases* dentro de un prototheads.

Hay que tener cuidado con el uso de variables locales en los protothreads debido a que Contiki no guarda el estado de las variables cuando se retorne al protothread, por lo tanto es recomendable usar variables globales o estáticas para cada función propia.

```
#include "contiki.h"
#include <stdio.h>
/*---------------------------------------------------------------------------*/
PROCESS(hello_world_process, "Hello world process");
AUTOSTART_PROCESSES(&hello_world_process);
/*---------------------------------------------------------------------------*/
PROCESS_THREAD(hello_world_process, ev, data)
{
  PROCESS_BEGIN();
  printf("Hello, world\n");
  PROCESS_END();
}
```

PROCESS(hello_world_process, "Hello world process") Este macro declara la función del protothread y define un proceso asociándole el protothread declarado. Se le debe ingresar como primer parámetro el nombre del proceso que se usa en el código C y de segundo parámetro una cadena de información que contenga un nombre con el cual el usuario identifique el proceso.

AUTOSTART_PROCESSES(&hello_world_process) este macro permite arrancar el proceso.
PROCESS_THREAD(hello_world_process, ev, data) este macro define el cuerpo del proceso(protothread). El proceso llama cada vez que se produce un evento en el sistema. Este macro incluye los campos *hello_world_process:* nombre del proceso, ev: contiene el número de evento y data: apuntador a los datos adicionales que se publican con el evento. Un proceso siempre comienza con el macro *PROCESS_BEGIN()* y termina con el macro *PROCESS_END()*.

4.6 PROGRAMAS CLIENTE Y SERVIDOR

Para establecer conexión entre dos dispositivos se ejecuta los programas cliente y servidor con el fin de establecer comunicación, estos archivos se encuentran alojados en el directorio de ejemplos, estos se modifican para cumplir las funciones de captura y empaquetamiento de la información. Se ejecuta en el entorno de simulación Cooja, el cual permite visualizar el comportamiento de la conexión entre los dos dispositivos. Esta tarea se realiza debido a que el simulador Cooja no tiene los paquetes necesarios para simular el USBstick jackdaw como un servidor. Este dispositivo está diseñado para que se comporte como un router de borde el cual se conecta con el equipo encargado de almacenar en la base de datos la información obtenida por el nodo sensor.

Para la conexión básica de un nodo sensor con un router de borde se muestra en de la siguiente figura 4.2.

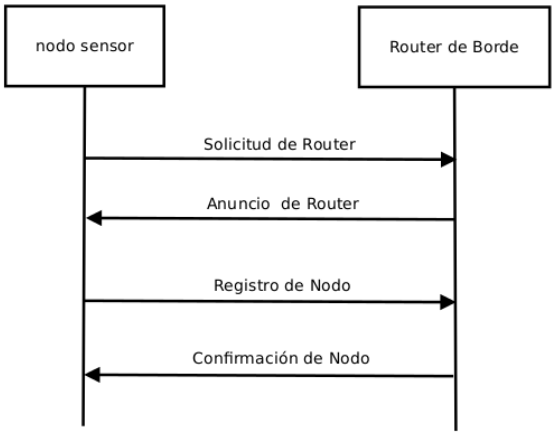

***Figura 4.2**: Proceso Básico de solicitud y registro del nodo con un router de borde*

El funcionamiento tanto para la transmisión del cliente como la recepción en el servidor se representan en el diagrama de flujo de la figura 4.3

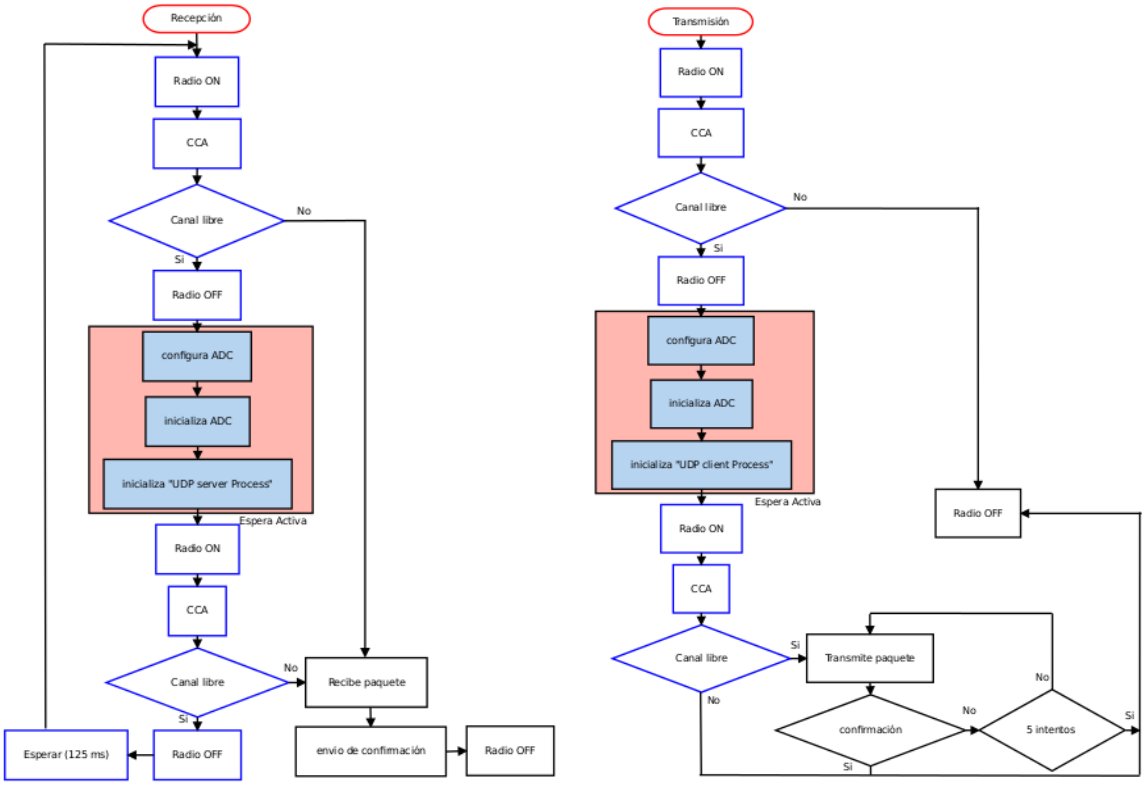

***Figura 4.3**: Diagrama de flujo de transmisión y recepción de ContikiMAC*

El sistema operativo contiene el código ContikiMAC[34] el cual es implementado en los archivos cliente y servidor con su correspondiente cabecera. ContikiMAC prende la radio 2 veces periódicamente para escuchar el canal con el fin de saber si existe una transmisión. Al encendido de radio para el chequeo del medio se le llama CCA (Clear Channel Assesment). La espera activa relaciona el tiempo que toma al hardware estabilizase y realizar el proceso de habilitar el ADC y realizar la captura de datos.

#define CCA_COUNT_MAX 2	ContikiMAC realiza la comprobación del canal de forma periódica. Cada comprobación del canal consiste en dos o más chequeos de CCA. CCA_COUNT_MAX es el número de las ACC a realizar por cada chequeo periódico del canal. Su valor predeterminado es dos
#define CCA_CHECK_TIME RTIMER_ARCH_SECOND/8192	CCA_CHECK_TIME es el tiempo que se necesita para chequear un CCA 1/8192
#define CCA_SLEEP_TIME RTIME_ARCH_SECOND/2000	CCA_SLEEP_TIME es el tiempo entre dos CCA sucesivas 1/2000= 0,0005 s

El macro *RTIMER_ARCH_SECOND* presenta un segundo en la arquitectura en la que se esté trabajando y es la referencia temporal para definir todas las demás constantes involucradas. El

número de CCA's (CCA_COUNT_MAX) definido por defecto es dos. Esto quiere decir que se harán dos escuchas consecutivas por cada ciclo. El valor *CCA_CHECK_TIME* (122us) es el tiempo de duración de una CCA y el valor CCA_SLEEP_TIME es el tiempo entre los dos CCA´s. El valor de CCA_SLEEP_TIME es el tiempo entre los dos CCA´s. El valor CCA_SLEEP_TIME se ve que está definido de manera que la separación entre CCA's sea de 0.5 ms. Este par de CCA's se realiza cada 125 ms en la configuración por defecto y está definida en el macro CYCLE_TIME [39].

4.7 COMPILACIÓN, GENERACIÓN DE ARCHIVOS EXE Y MAKEFILES

Contiki tiene un makefiles implementado en la carpeta raíz llamado Makefiles.include. Cuando se quiera implementar un programa nuevo se debe crear un archivo makefiles el cual incluye al archivo Makefiles.include para que se incluyan todos los archivos necesarios del sistema operativo

La estructura del archivo Makefiles para los programas udp-client-adc y udp-server-adc se establecen de la siguente manera:

Makefiles

```
// programas incluidos para la construcción del maque TARGET=avr-raven
All:udp-client-adc udp-server-adc
// habilitando esta bandera indica que Makefiles.include incluirá los paquetes de configuración IPv6 en su //compilación
UIP_CONF_IPV6=1
//se indica donde se encuentra la carpeta raíz de Contiki en donde se encuentra alojado el //Makefiles.include
CONTIKI = ../..
// se incluye el archivo Makefiles.include
include $(CONTIKI)/Makefile.include
```

4.8 ALMACENAMIENTO DE INFORMACIÓN EN LA BASE DE DATOS

Para esta tarea se instala el servidor apache 2.0 en Ubuntu 2.4 el cual albergará la información capturada por los nodos sensores. Adicional a esto se instala los paquetes de gestión como el módulo de php, MySQL, y un gestor de base de datos como PHPMyadmin[40].

La instalación del servidor, paquetes de gestión y creación de base de datos se muestra en anexo dos y tres. PhpMyAdmin es una herramienta que se basa en la administración de base de datos y tablas MySQL, permitiendo así la creación de bases de datos de una manera fácil e intuitiva.

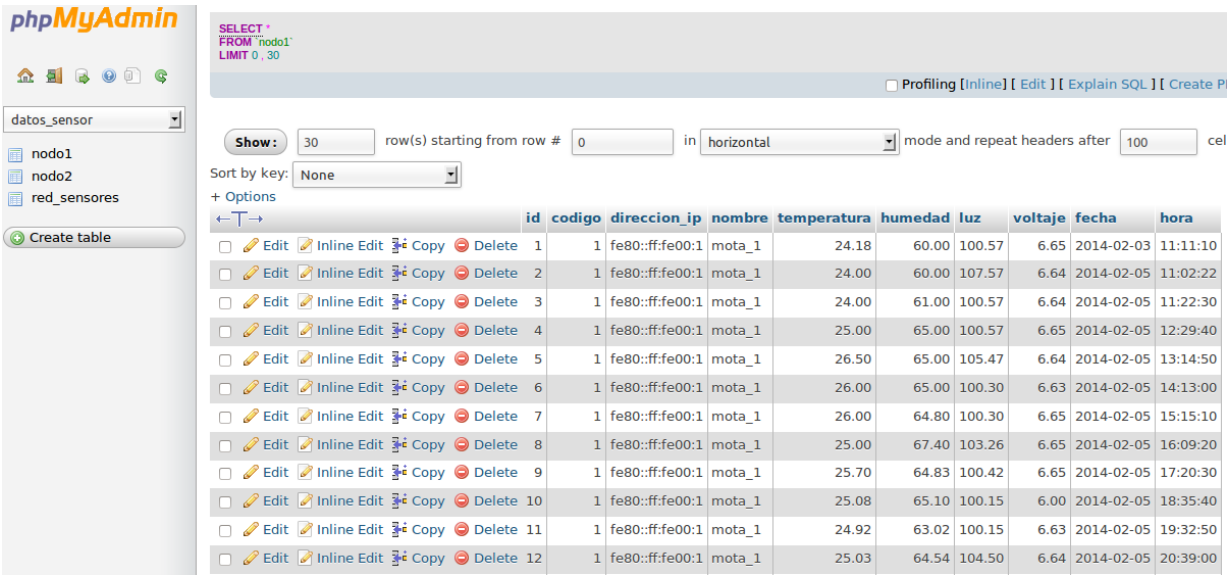

***Figura 4.4**: Base de datos*

4.9 COMPONENTES DE LOS DATOS

El aplicativo contiene en su estructura los siguientes componentes:
Se crea una base de datos llamada “datos_sensor” que contiene 4 tablas las cuales son:

***Tabla 4.1**: Nodo*

Nombre Tabla	Descripción
motas	Contiene el nombre de los dispositivos conectados
Nodo1	Contiene la información del nodo1 o mota 1
Nodo2	Contiene la información del nodo2 o mota 2
Red_sensores	Contiene la información del último dato ingresado

A continuación se describen cada una de las tablas del modelo

***Tabla 4.2:** motas*

Nombre tabla: motas		
Nombre_campo	**Tipo de dato**	**Descripción**
Id_mota	Entero	Identificador de la tabla
Nombre	cadena	Código de la mota

Descripción de cada una de las tablas del modelo.

Tabla 4.3: *Nodo 1*

Nombre tabla:Nodo1		
Nombre campo	**Tipo de dato**	**Descripción**
Id	Entero	Identificador de la tabla
Código	Cadena	Código de la mota
Dirección_ip	Cadena	Dirección de la mota
Nombre	Cadena	Nombre de la mota
Temperatura	Decimal (00,00)	Valor de temperatura
humedad	Decimal (00,00)	Valor de humedad
Luz	Decimal (00,00)	Valor de luz
Voltaje	Decimal (00,00)	Valor de voltaje
Fecha	Fecha	Fecha en que obtuvo el dato
hora	Hora	Hora en que obtuvo el dato

Tabla 4.4: *Nodo 2*

Nombre tabla:Nodo2		
Nombre campo	**Tipo de dato**	**Descripción**
Id	Entero	Identificador de la tabla
Código	Cadena	Código de la mota
Dirección_ip	Cadena	Dirección de la mota
Nombre	Cadena	Nombre de la mota
Temperatura	Decimal (00,00)	Valor de temperatura
humedad	Decimal (00,00)	Valor de humedad
Luz	Decimal (00,00)	Valor de luz
Voltaje	Decimal (00,00)	Valor de voltaje
Fecha	Fecha	Fecha en que obtuvo el dato
hora	Hora	Hora en que obtuvo el dato

Tabla 4.5: *Red de Sensores*

Nombre Tabla: red_sensores		
Nombre campo	**Tipo de dato**	**Descripción**
Id	Entero	Identificador de la tabla
Código	Cadena	Código de la mota
Dirección_ip	Cadena	Dirección de la mota
Nombre	Cadena	Nombre de la mota
Temperatura	Decimal (00,00)	Valor de temperatura
humedad	Decimal (00,00)	Valor de humedad
Luz	Decimal (00,00)	Valor de luz
Voltaje	Decimal (00,00)	Valor de voltaje
Fecha	Fecha	Fecha en que obtuvo el dato
hora	Hora	Hora en que obtuvo el dato

4.10 MODELO DE DATOS

Para llevar un buen manejo de la información es necesario aplicar un modelo relacional utilizado para la gestión de los datos, la figura 4.8 muestra un modelo relacional sencillo uno a muchos que consta de tres tablas, las cuales son: mota, nodo1 y nodo2.

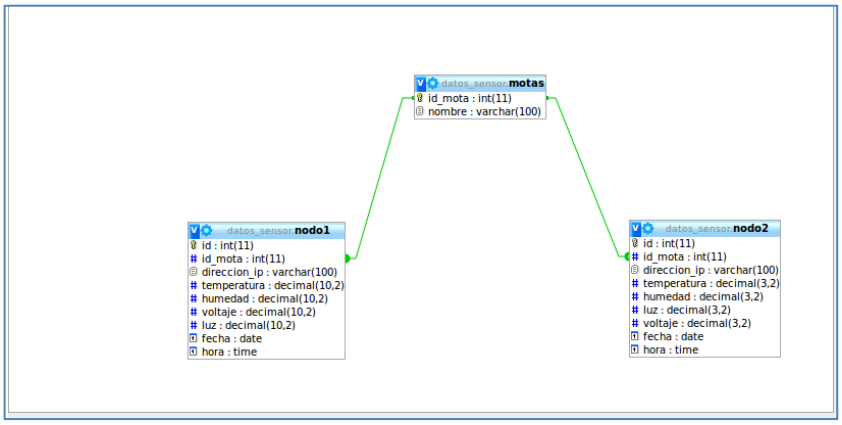

***Figura 4.5**: Modelo Relacional*

La tabla motas contiene el nombre y el código de los diferentes dispositivos conectados al servidor principal, los cuales para efectos prácticos en este trabajo se adicionan de forma manual, se espera que en un estudio más profundo esta información sea adicionada de forma automática por la aplicación.

El contenido de la tabla motas será el siguiente.

***Tabla 4.1**: Tabla motas*

Id_mota	Nombre
1	Mota1
2	Mota2
3	Mota3
4	Mota4
.	.
.	.
.	.
n	Mota n

Donde el campo id_mota es un numero entero creado automáticamente por el gestor de base de datos que indica el código de la mota y el campo nombre que corresponden a una cadena de caracteres que describen el nombre de la mota.

La tabla nodo1 contiene la información que recibe de una mota o nodo sensor por parte del socket, el cual se encarga de recibir los paquetes capturados por el USBstick jackdaw ya que este dispositivo se comporta como una interfaz de red.

Para la adición de la información de las tablas se realiza una serie de consultas en PHP el cual se encarga de consultar a una archivo de texto, este es generado por el programa server6.c [41] quien se encarga de escuchar el puerto por el cual se intercambia la información de los nodo sensores.

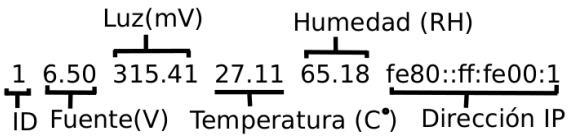

***Figura 4.6**: Estructura de información*

Cada paquete contiene una serie de variables donde cada variable se separa de las demás con un espacio en blanco. El código realizado en PHP lee cada línea y ejecuta una consulta a la base de datos específica, una vez leído y actualizado en la base de datos, el archivo de texto es borrado.

El código PHP para insertar la información presenta el siguiente esquema.

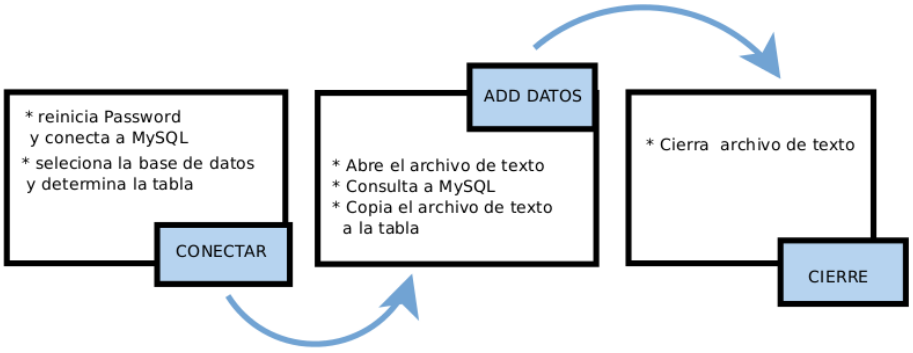

***Figura 4.7**: Diagrama de carga de la base de datos*

El archivo de texto server6.c está ajustado para que inserte la información de 20 paquetes en el archivo de texto y luego el archivo es borrado para nuevamente ser creado en la llegada de otro paquete, esto se realiza con el objetivo de limpiar la hoja de texto y actualizar los datos de llegada.

4.11 INTERFAZ WEB

Los valores de medición de los sensores son enviados vía inalámbrica de un nodo a otro para luego ser recibidos por el USBSTICK, este realiza una conexión con el servidor para permitir el almacenamiento de los datos, cuando la información ha sido almacenada, es necesario elegir un sistema adecuado para mostrar los datos solicitados por los usuarios conectados al servidor se establecen requerimientos de la visualización como:

- La aplicación permite la visualización de gráficos estadísticos basándose en los datos arrojados por el hardware.
- La aplicación muestra los gráficos por lapsos de tiempo (segundo, minutos u horas).

4.12 HERRAMIENTAS WEB

Para el desarrollo del aplicativo se utilizaron las siguientes tecnologías en su desarrollo:

Symfony 1.4.20v: es un Framework completo diseñado para optimizar el desarrollo de aplicaciones web a través de varias características clave. Para empezar, separa las reglas de una aplicación web de negocios, la lógica del servidor, y las vistas de presentación. Contiene numerosas herramientas y clases encaminadas a acortar el tiempo de desarrollo de una aplicación web compleja. Además, automatiza las tareas comunes para que el desarrollador pueda centrarse exclusivamente en los aspectos específicos de una aplicación. Synfony está escrito íntegramente en PHP. Es compatible con la mayoría de los motores de bases de datos disponibles, incluyendo MySQL, PostgreSQL, Oracle y Microsoft SQL Server, es ejecutable en Unix y plataformas Windows.

PHP[42]**:** (acrónimo recursivo de PHP: Hypertext Preprocessor) es un lenguaje de código abierto muy popular especialmente adecuado para el desarrollo web, que puede ser incrustado en HTML, el cual se ejecuta en el servidor. Actualmente PHP se encuentra en su **versión 4.**

Jquery[43]: Es considerado un marco de trabajo de Javascript. Que posee un conjunto de funciones que permite interactuar con los documentos HTML de una manera más fácil y simplificada, por ejemplo: galerías de fotos dinámicas y elegantes, validación de formularios, calendarios, etc. Es compatible con los navegadores Mozilla Firefox, Internet Explorer, Safari, Opera y Google Chrome.

Google Chart [44]: Es una biblioteca que permite la creación de diferentes tipos de gráficos. Proporcionando una manera perfecta para visualizar los datos en el sitio web, la galería gráfica ofrece desde gráficos de líneas simples hasta complejos mapas de árboles jerárquicos, presentando así un gran número de opciones de representación de información.

Para los archivos de aplicación el marco de trabajo Symfony 1.4.20v presenta la siguiente estructura de archivos correspondientes a las carpetas.

***Tabla 4.5**: Estructura Symfony 1.4.20v*

Directorio	Descripción
apps/	Hospeda todas las aplicaciones del proyecto
cache/	Los archivos en caché
config/	Los archivos de configuración del proyecto
lib/	Las bibliotecas y clases del proyecto
log/	Los archivos de registro
plugins/	Los plugins instalados
test/	Los archivos de pruebas unitarias y funcionales
web/	El directorio raíz web

De las cuales se usaron apps, lib, y web.

Apps: se encuentra toda la información de las tablas, cada una con su respectivo modulo. Así pues existe un módulo llamado nodo1 que corresponde a la tabla nodo1, modulo nodo2 para la tabla nodo2 y así sucesivamente. En estos se encuentran todos los archivos.

Lib: Se encuentran los archivos pertenecientes al modelo de datos de las clases asociadas.

Web: Se encuentra los archivos Css, Js e imágenes que contiene el proyecto.

La forma de guardar los datos se realiza por inserción de tablas, y la forma de mostrarlo se realiza por gráficos.

4.13 FUNCIONAMIENTO DEL LA INTERFAZ WEB

La aplicación web permite visualizar los datos obtenidos por los sensores encargados de capturar ciertas variables del entorno como humedad relativa, temperatura, detección de luz y medición de la fuente de energía. Como primera medida se abre el navegador y se coloca la siguiente dirección: http://sensor.localhost/sensores_app_dev.php, mostrando la siguiente imagen

***Figura 4.8**: Pagina de Presentación*

Esta permite visualizar la página de presentación, para continuar se debe hacer click en el nombre y continuar hacia el menú principal de la aplicación figura 4.9 en la cual se encuentran tres iconos:

Modulos: contiene iconos a cada uno de los módulos del aplicativo en este caso son los dispositivos con sus respectivas opciones.

Graficos: contiene iconos a cada uno de los módulos de visualización de gráficos correspondientes a cada nodo sensor.

Acerca de: contiene la información de las personas involucradas que hacen parte del proyecto

***Figura 4.9**: Menú principal*

Continuando con el manejo del aplicativo se procede a ingresar a cada uno de los iconos antes mencionados, en su orden se explicará el contenido del icono módulos figura 4.10. Al ingresar al icono se encuentra otro menú el cual presenta la siguiente información

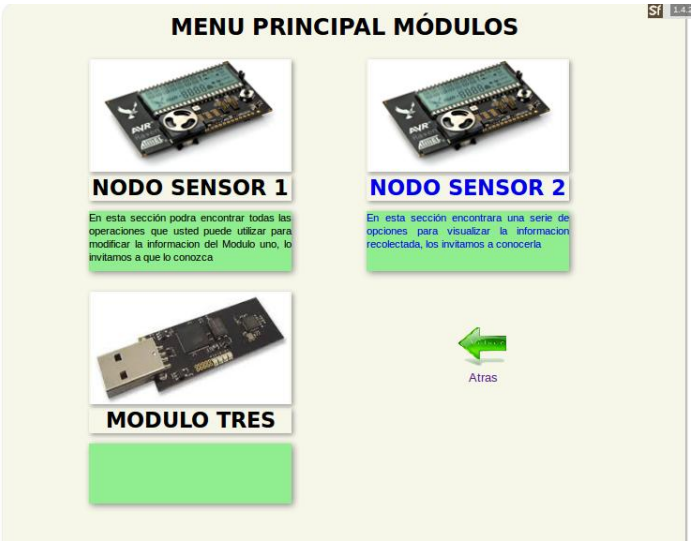

***Figura 4.10**: Menú Módulos*

Nodo sensor1: contiene las operaciones necesarias para capturar la información obtenida del nodo sensor.

Modulo tres: contiene información acerca del dispositivo RZUSBSTICK el cual hace el trabajo de router de borde.

Continuando se ingresa a cada uno de los tres iconos donde se hace su descripción. Esta sección describe cada uno de los iconos que contiene el ítem NODO SENSOR 1, al adicionar otro nodo análogamente es similar al primero como lo es el NODO SENSOR 2.

ADICIONAR: al ingresar permite asignar un valor numérico mayor a 0 para iniciar la carga de datos que el hardware ha enviado.

HISTORIAL: permite visualizar los datos que han ingresados a la base datos en un instante determinado.

ELIMINAR: permite borrar la información correspondiente a la mota o nodo 1 y que se encuentra consignada en la base de datos.

Para ilustrar lo antes expuesto se presenta la figura 4.11:

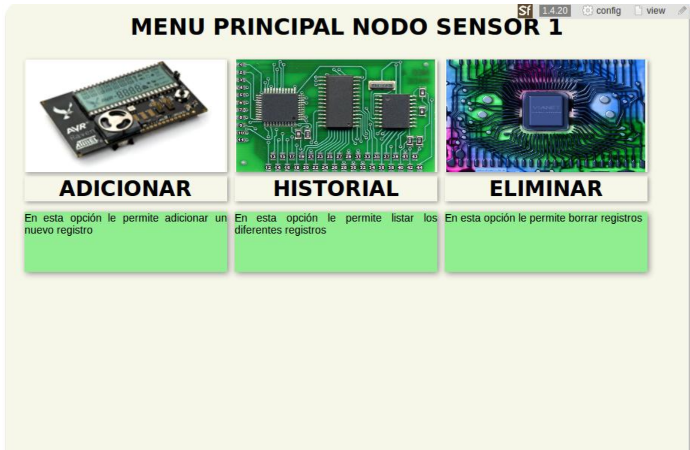

***Figura 4.11**: Menú Principal nodo Sensor1*

A continuación se explica cada uno de los iconos.

ADICIONAR: esta opción permite cargar los datos que el hardware a enviado y escrito en un archivo de texto plano figura 4.11, el proceso para la carga es muy sencillo, basta con ingresar un valor numérico superior a 0 en la caja de texto “valor de refresco” y hacer clic en el botón “cargar datos” para iniciar el proceso de carga.
Cuando inicia los datos que han cargado se muestran en el área de texto que se encuentra en la parte inferior de la pantalla.
Los datos se insertan automáticamente a la base de datos y el proceso es continuo hasta que se cierre el navegador.

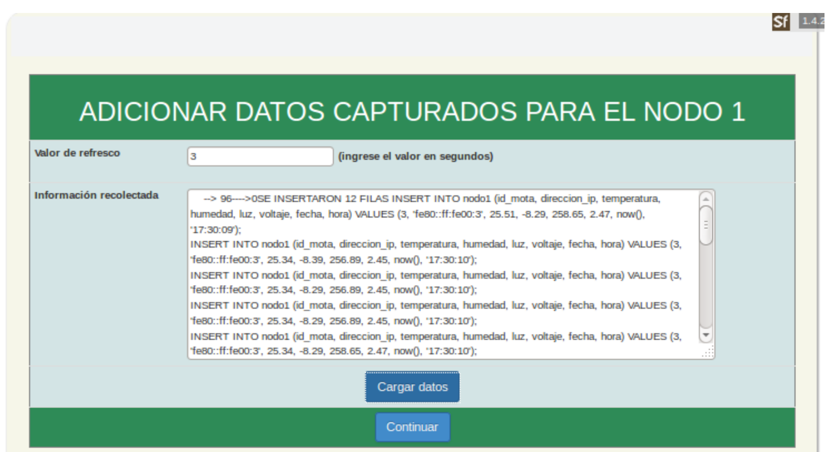

***Figura 4.12**: Adicionar tiempo de ajuste para consultas*

El proceso de refresco de los datos manera automática utilizando JavaScript y Ajax con un tiempo definido por el usuario, el uso de esta tecnología permite reducir el gasto computacional y hace más

ágil el proceso y evitando crear ciclos infinitos que puedan afectar el desempeño del servidor apache.
Cuando se hace clic en el botón continuar no llevara directamente a los gráficos estadísticos sin afectar la carga, la cual seguirá continuamente y solo terminará cuando se cierre el navegador.

HISTORIAL: en la parte superior se encuentra el menú que lleva directamente a cada sección del proyecto.

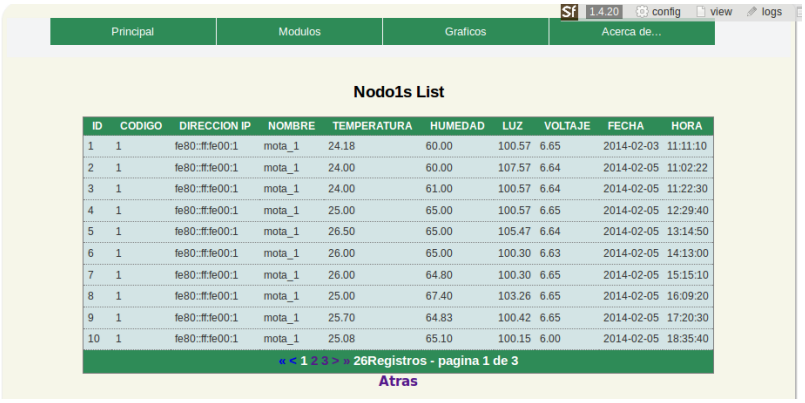

***Figura. 4.13**: Tabla con el historial de medición*

En esta opción permite mostrar los datos que han sido ingresados a la base de datos, la información se muestra en un listado y en la parte inferior de enuncia la cantidad de registros y el número de páginas de registros.
En la parte inferior aparece un enlace “atras” el cual permite devolverse al menú anterior.

***Figura. 4.14**: Eliminar Tabla*

ELIMINAR: esta opción permite eliminar todos los datos que se encuentran en la base datos; se debe seleccionar al menos una de las casillas de verificación y presionar el botón borrar contenido. En la parte inferior aparece un enlace “atras” el cual permite devolverse al menú anterior.

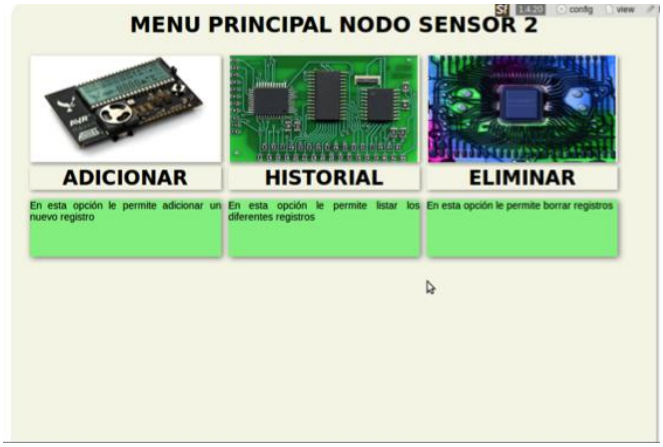

***Figura. 4.15**: Eliminar Tabla*

En la figura 4.14 muestra los iconos correspondientes al nodo sensor 2, cada uno de los iconos tiene el mismo funcionamiento que tiene el nodo sensor 1, por lo tanto no se hará descripción de estos.

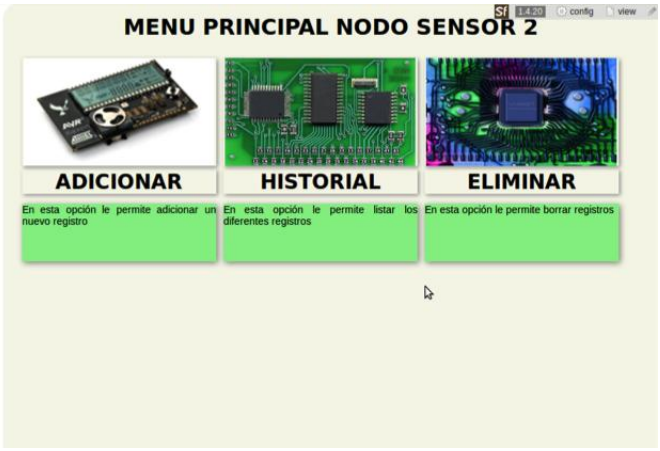

***Figura. 4.16**: Gráficos nodo Sensor 2*

En la sección se hace la descripción de el modulo gráficos. En la figura 4.15 se muestra dos iconos, los cuales se describen a continuación:

MONITOR: en esta sección se muestra los dos últimos datos ingresados en el sistema.

HISTORIAL: en esta sección se muestran los diferentes valores graficados de forma independiente.

Al hacer clic en el icono HISTORIAL se despliega el siguiente menú:

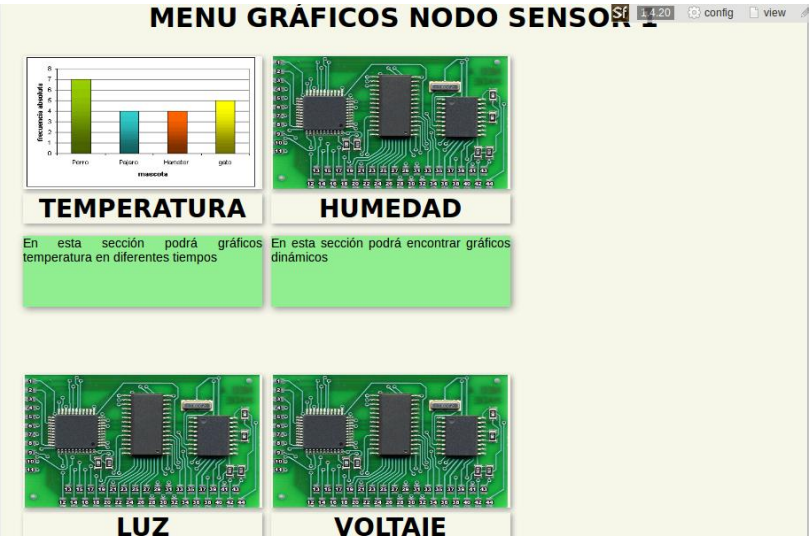

***Figura. 4.17**: Menú de gráficos de variables*

Este contiene la interfaz para graficar los datos recolectados por los nodos sensor con respecto al tiempo que fue ingresado, dentro de estos iconos se encuentran:

TEMPERATURA: en este icono se muestran los gráficos para los nodos 1 y 2, del valor de temperatura con respecto al tiempo.

HUMEDAD: en este icono se muestran los gráficos para los nodos 1 y 2, del valor de humedad con respecto al tiempo.

LUZ: en este icono se muestran los gráficos para los nodos 1 y 2, basándose en el valor de luz frente al voltaje (mV).

VOLTAJE: en este icono se muestran los gráficos para los nodos 1 y 2, basándose en el valor de voltaje.

CAPITULO 5

PRUEBAS Y RESULTADOS

En este capítulo se muestra los resultados relacionado al estudio del protocolo 6loWPAN, se decide establece un sistema de comunicación que permita establecer conexión entre dispositivos aplicando la tecnología 6LoWPAN. Para el cual se establece una configuración de prueba que se encarga de recolectar los datos capturados del entorno por los sensores para luego ser almacenado y visualizados desde un servidor web.

PROCEDIMIENTO

Para que el sistema completo realice el proceso de almacenamiento de datos capturados por el nodo sensor en el servidor es necesario que el servidor web, la aplicación y el archivo ejecutable creado por server6.c estén corriendo, también debe estar asignada la dirección IP a la interfaz creada por la USBStick Jackdaw que para el caso se asigna la dirección fe80::ff:fe00:2 y para la plataforma AVR-RAVEN se asigna la dirección IP fe80::ff:fe00:1, se enciende la AVR-RAVEN la cual arranca cargando el sistema operativo por un breve lapso de tiempo, luego comienza a establecer conexión con el router de borde USBStick Jackdaw, este es escuchado por el archivo ejecutable creado por server6.c el cual se encarga de capturar la carga útil entrante en un archivo de texto, este archivo de texto es consultado por la base de datos para almacenar la información en el servidor para luego ser visualizada por el aplicativo web, la figura 5.1 muestra el diagrama de red que se configura para realizar las pruebas.

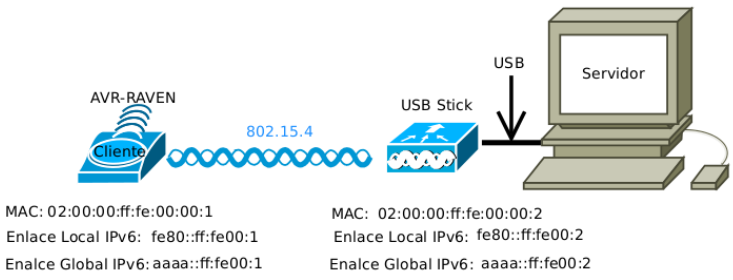

***Figura 5.1**: Configuración de prueba*

5.1 CLIENTE-SERVIDOR

Se establece una aplicación cliente – servidor, en el cual se crea dos programas que se implementan junto al Sistema Operativo, el programa cliente se encarga de capturar los datos de los sensores y establece conexión con el servidor web, para así suministrarle los datos capturados, el programa cliente es implementado en la arquitectura ARV-RAVEN. Mientras que el servidor está del lado del USBstick.

PROGRAMA CLIENTE Antes de modificar el código del cliente, la dirección de la MAC debe ser modificada en *contiki-main*, debido a que las direcciones tanto del cliente como la del servidor deben ser diferentes.
Después de cambiar la dirección MAC, el programa cliente debe ser cambiado para incluir la dirección del ATmega1284p del servidor, esto se puede visualizar con el simulador COOJA, de manera que el cliente puede enviar un mensaje al servidor.

El código de la función ***set-connection_address*** en udp-client-adc se debe establecer los siguientes cambios para realizar la modificación de la dirección IP del nodo sensor de forma manual:

contiki-main modificar dirección IP

```
static void
  set_connection_address(uip_ipaddr_t *ipaddr)
  {
    uip_ip6addr(ipaddr,0xfe80,0,0,0,0xff,0xfe00,0x1);
  }
```

5.1.1 NODO CLIENTE UDP SENSORES

Se prueba la adquisición de los datos y se logra establecer comunicación entre el nodo y el servidor. Una vez probados los procesos relacionados se establece el nodo sensor validado para el resto del proyecto.

La prueba consiste en dejar correr el programa hecho para la plataforma AVR-RAVEN junto con la tarjeta de adquición de datos de los sensore, el cual se encarga de capturar los datos obtenidos de los sensores y luego son enviados cada determinado tiempo, posteriormente se muestra en pantalla los datos generados.

Se verifica que el dato UDP enviado es recibido correctamente, pero en ciertos lapsos de tiempo se presentan problemas para establecer conexión, por lo que se percibe que frecuentemente el mensaje transmitido por el cliente no ha sido recibido por el servidor.

EJECUCIÓN DEL CÓDIGO CLIENTE

En el anexo A se visualiza figura A.1 el arranque del sistema operativo Contiki, la inicialización y verificación de la radio rf230, muestra la versión e ID, también la dirección MAC y el canal por el cual se comunica siendo este último con frecuencia de 2.4 GHz. Se inicia el proceso de comunicación UDP y con una función que usa la MAC se establece la dirección IPv6 del nodo, también se define el puerto de comunicación. Finalmente según el tiempo establecido el nodo cliente transfiere el estado del proceso desde el estado del proceso UDP al estado de lectura del ADC.

5.1.2 NODO SERVIDOR UDP SENSORES

De la misma manera para la prueba en el simulador Cooja, al servidor se asigna el canal MAC y puerto, luego se realiza el arranque del sistema operativo para verificar su correcto funcionamiento. La tarea del servidor es esperar datos suministrados por el cliente, el cual responde con una respuesta de confirmación. Este programa solo se ejecuta en el simulador ya que para la práctica se utiliza el ejecutable suministrado por socker6.c.

5.2 LECTRURA DE SERVER6.C

La información suministrada por el nodo cliente es escuchada por medio de un Socket UDP versión 6, como muestra la imagen de la consola de comandos en la figura 5.2, siendo el resultado del programa ejecutable de server6.c (socket), este imprime el resultado de los valores obtenidos por los nodos. El código se encuentra disponible en el anexo E.

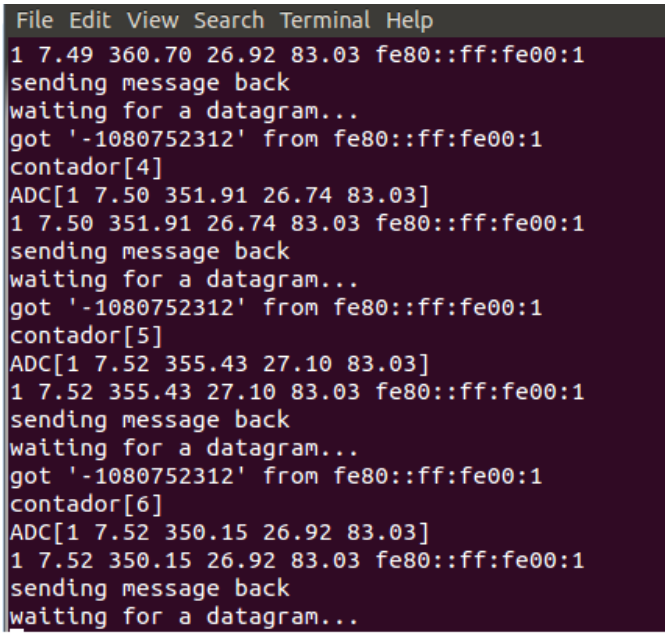

***Figura 5.2**: Datos obtenidos de los sensores*

Para la comprobación de que se está aplicando IPv6 para redes de sensores se usa el analizador de protocolos Wireshark, ya que permite clasificar protocolos de comunicación, visualizar el flujo y el contenido de los paquetes.

5.3. WIRESHARK

Wireshark [45] es una herramienta que se encarga de analizar protocolos, la cual permite el monitoreo de datos de una red en tiempo real, además brinda una gran variedad de filtros para organizar la información. Se encuentra distribuido bajo licencia GLP de GNU. Este software se utiliza junto con la AVR Raven USBstick para realizar capturas del tráfico que se establece entre los nodos y el servidor montado en el pc.

Se utiliza las siguientes instrucciones desde consola para la asignación de la IP al USBstick Jackdaw figura 5.3:

```
sudo ifconfig usb0 up
add ip -6 address add fe80::ff:fe00:2/64 dev usb0
sudo wireshark
```

***Figura 5.3**: Instrucción por consola para escuchar trafico usb0*

Como se puede observar el resultado de la captura realizada con Wireshark Figura 5.4 el proceso de handshaking entre uno de los nodos y el router de borde se reconocen mutuamente, para finalmente establecer la conexión y comenzar la transmisión de paquetes UDP.

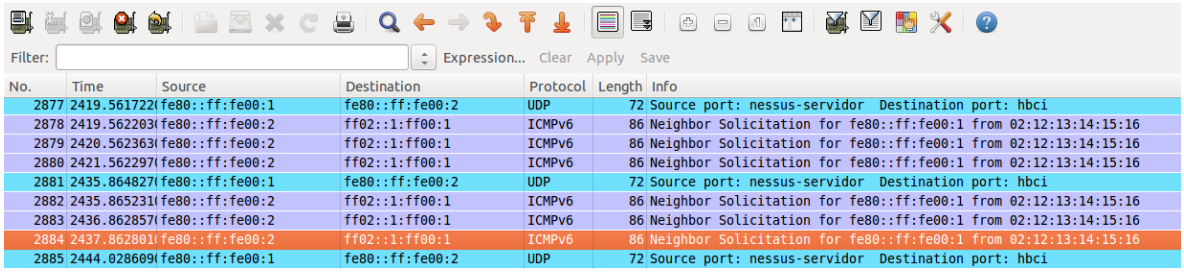

***Figura 5.4**: Comunicación UDP Capturas Wireshark*

En la figura 5.5 muestra la captura de un mensaje de control de solicitud de vecino (Neighbor Solicitation). ICMPv6 del USBstick que trabaja como router de borde al cliente AVR-RAVEN, como se aprecia en el grafico en la cuarta y la quinta línea se observa el origen/destino del paquete y la versión del protocolo.

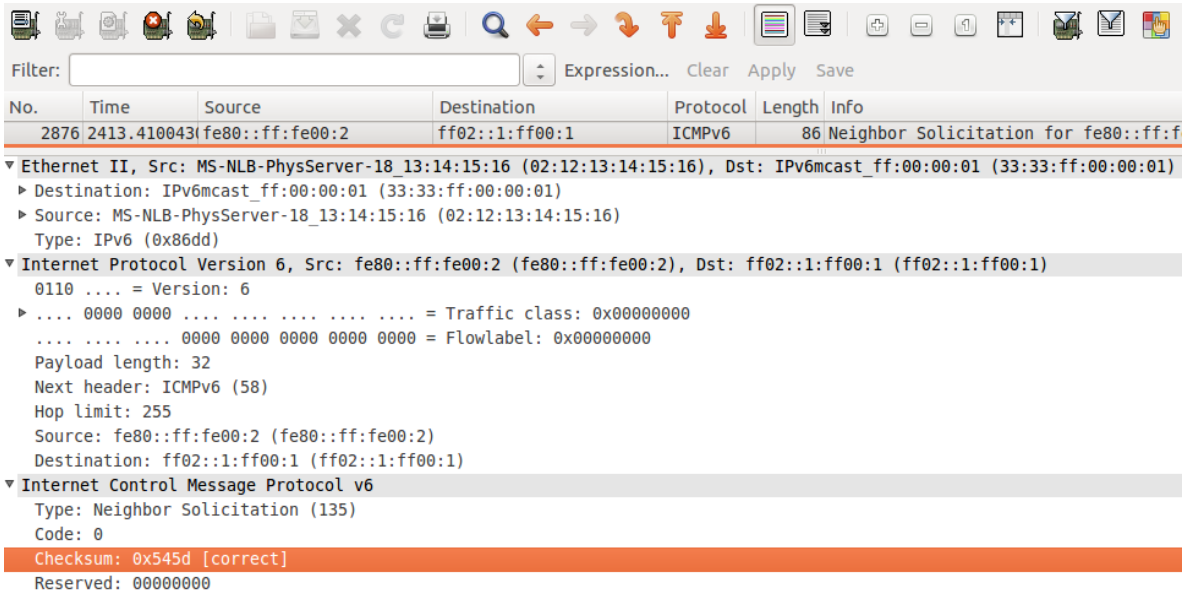

***Figura 5.5**: Capturas Wireshark de Mensaje de control* Neighbor Solicitation *ICMPv6*

En la figura 5.6 muestra como el cliente envía el paquete de información UDP capturada por los sensores al Router de borde, como se aprecia en el grafico en la línea catorce se observa el origen y destino del paquete, también en la línea cinco se aprecia el tipo del protocolo y la versión, en la línea quince se observa los puertos utilizados para establecer la comunicación.

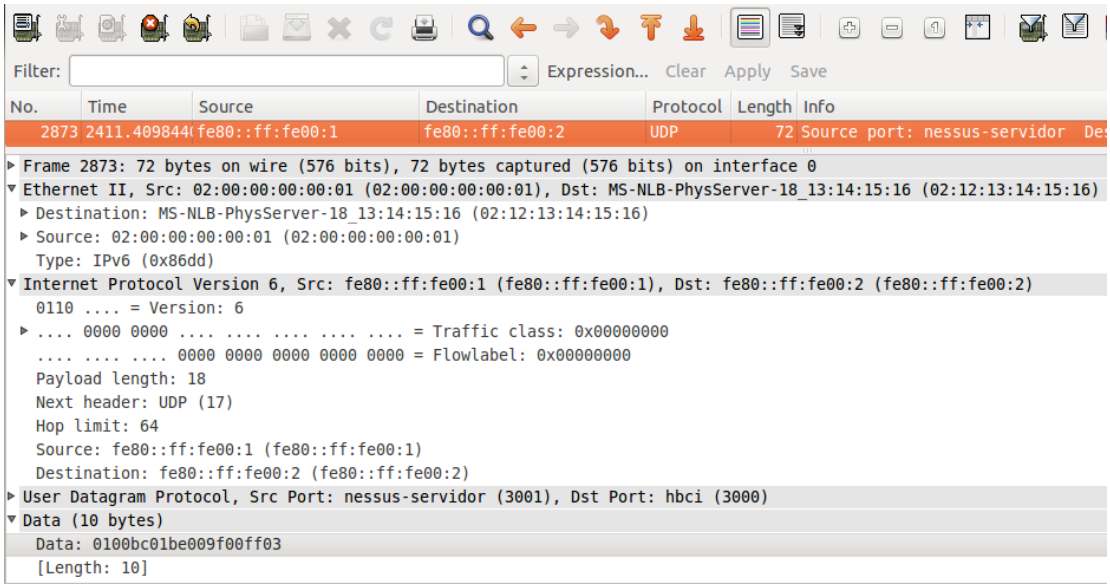

***Figura 5.6**: Capturas Wireshark de envío de paquete UDP*

5.3.1 RESPUESTA DEL ROUTER DE BORDE

En la figura 5.7 muestra como le envía un paquete del router de borde al cliente confirmado que ha recibido el paquete UDP con el mensaje “ok cliente!!”

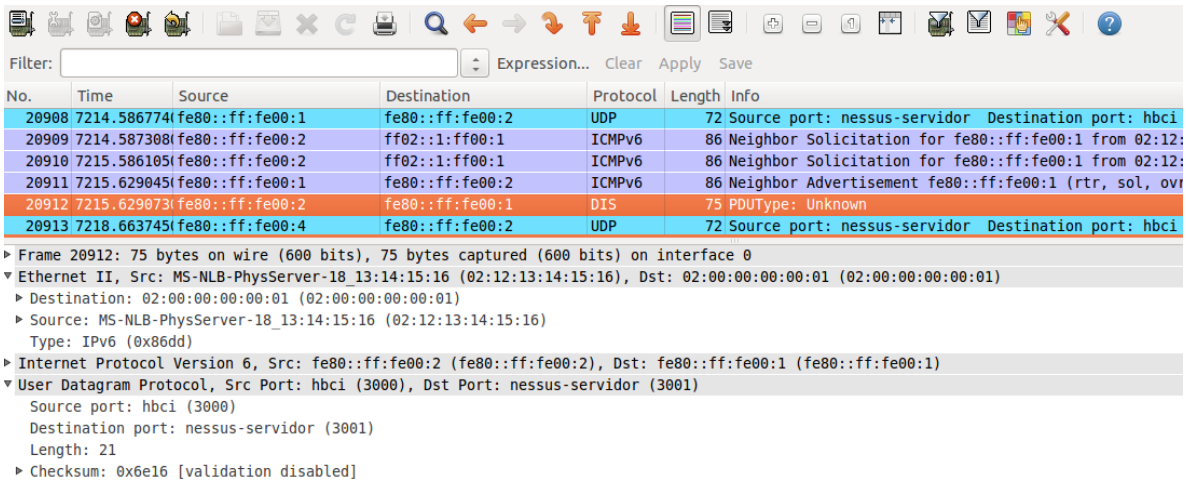

***Figura 5.7**: Datos obtenidos pantalla principal Wireshark*

5.4. INTERFAZ DE INFORMACIÓN

Para la visualización de la información se presenta una interfaz Web figura 5.8, en la parte superior muestra la barra de información la cual se encarga de realizar los saltos de las páginas. Dentro de la aplicación se crea un archivo HTML que permite a los usuarios consultar y visualizar los datos del servidor, consultando las mediciones del último dato presente ingresado del nodo1 y nodo2, el cual posibilita visualizar la diferencia del estado de las variables.

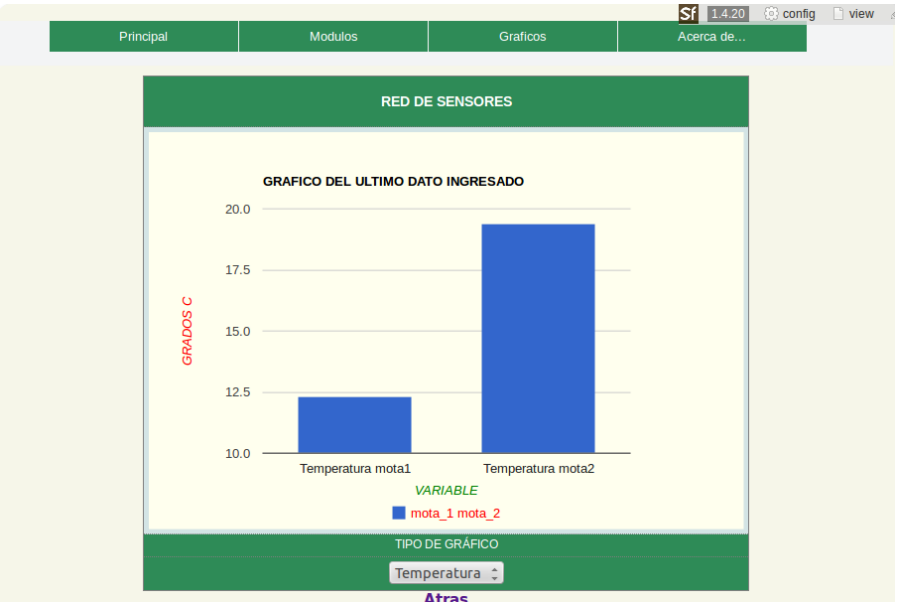

***Figura 5.8**: Monitor.html*

Se crea un archivo HTML figura 5.9 el cual se encarga de suministrar el historial de medición de los datos almacenados en la base de datos.

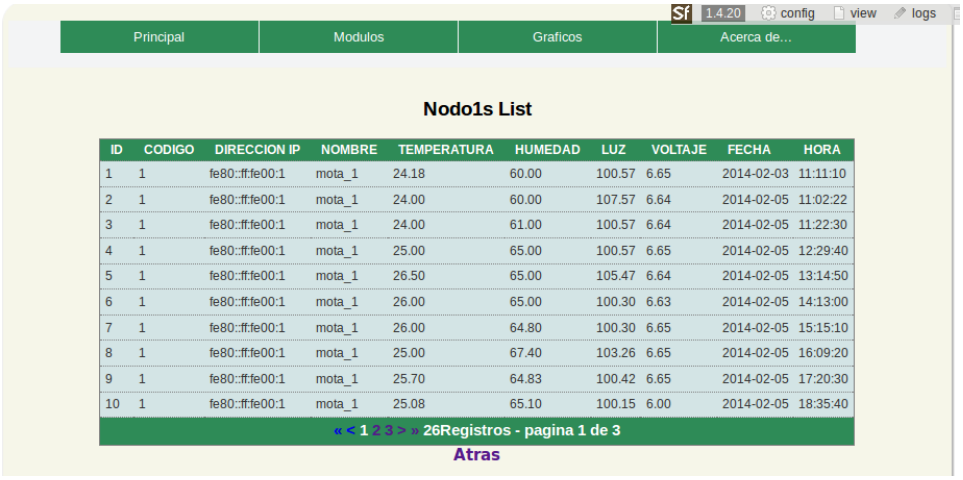

***Figura. 5.9**: Tabla con el historial de medición*

La figura 5.10 permite ver el gráfico que muestra de forma más clara las diferencias en las mediciones entre los nodo sensores.

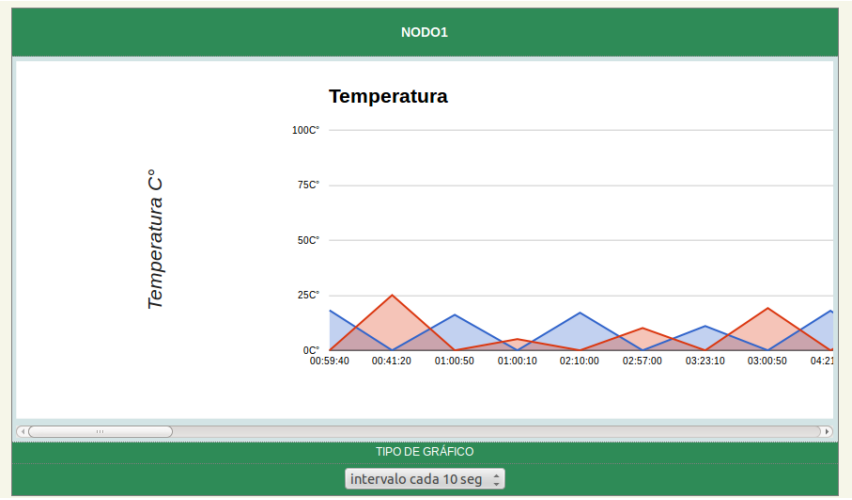

***Figura 5.10**: Historial.html*

El período de tiempo depende de la cantidad de resultados que se logre ingresar a la base de datos. Es posible modificar el período de tiempo 10 segundos, 10 minutos y una hora, el cual podría modificarse a un periodo más largo, por ejemplo, semanas, meses o incluso años, luego de elegir el periodo de tiempo de captura, la página permite visualizar los datos ingresados por nodos sensores, ya sea humedad, temperatura intensidad de luz o voltaje de la fuente.

5.5 NODO SENSOR – AVR-RAVEN

Se diseña una tarjeta para la adquisición de señal a cual es conectada con la plataforma AVR-RAVEN figura 5.11. Esta tarjeta está construida con elementos de electrónica activos y pasivos necesarios para acondicionar la señal capturada por los sensores, para cada variable censada es acondicionada con un seguidor de voltaje o también llamado buffer de voltaje encargado de realizar el acoplamiento de impedancias.

***Figura 5.11**: AVR-RAVEN en Nodo Sensor y USBstick Jackdaw*

INCONVENIENTES

La USBstick Jackdaw presenta problemas de conexión con el computador en el cual se instala el servidor web, al parecer presenta problemas de sincronización del hardware con el puerto usb, valga decir que esta interfaz se muestra como una terminal de red, de tal manera que si se establece desconexión entre los dispositivos es necesario asignarle nuevamente la dirección IP.

La interfaz web presenta una limitación en el momento de almacenar los datos ya que si por casualidad se sale del aplicativo las consultas de datos realizadas dejan de hacerse, por lo cual es necesario que el aplicativo se esté ejecutando.

El archivo server6.c presenta un inconveniente al momento de almacenar los datos ya que en el instante en que se borra y se crea nuevamente el archivo de texto se pierde un paquete en la transición de borrar y a crear.

Realizando pruebas con uno de los dispositivos AVR_RAVEN presento daños de manera que se busca reemplazarlo, pero la empresa ATMEL que suministra lo ha descontinuado. Al parecer dispositivo fue reemplazado por microcontroladores que ya viene incorporado el transductor inalámbrico, esto se debe a que presentaba mayores costos, en consumo de energía, tamaño y precio.

CONCLUSIONES

En el presente proyecto se realizó un estudio acerca de lo que involucra el protocolo 6LoWPAN y sus tecnologías asociadas, como también conocer su potencial en diversas aplicaciones.

La aplicación ha sido desarrollada con el protocolo 6LoWPAN, es decir, los nodos tienen su propia dirección IPv6 y se pueden conectar a una red local. Los dispositivos envían la información al servidor, el cual recoge los datos y los dispone para luego ser consultados a través de un sitio web.

En este sentido, se ha realizado una implementación real de este protocolo utilizando la plataforma AVR-RAVEN y se ha puesto a prueba su funcionamiento, el cual logra establecer conexión con el servidor conectado al rourer de borde. El flujo de información no es continuo por lo que en ciertos periodos de tiempo se pierde la conexión.

Para aplicar el protocolo 6loWPAN se usa un sistema operativo que permite la implementar la uIPv6, permitiendo una buena administración de los recursos del sistema embebido. El uso de sistemas operativos que trabajan con limitados recursos de procesamiento, abre la posibilidad a la implementación de redes capaces de interconectar con una gran cantidad de nodos. El sistema operativo Contiki es una buena opción para la implementación de redes de sensores, pues soporta una gran variedad de arquitecturas y facilita la programación gracias a sus herramientas de simulación, reduciendo así el trabajo al programador.

Se pueden hacer mejoras en lo que se refiere a la calidad de fabricación del PCB y también en el terminado en la soldadura de los componentes con las pistas con el fin de garantizar un buen funcionamiento, pues de ello depender la fiabilidad de adquisición de la información del entorno.

En un ámbito más general se percibe que uno de los problemas relacionados con Internet tiene que ver con el direccionamiento IP. Se sabe que las tradicionales direcciones IPv4 se encuentran en detrimento por lo cual la solución presentada por la IETF es darle paso a los nuevos espacios de direcciones IPv6, con el objetivo de que se despliegue IPv6 de forma permanente en los productos y servicios.

Específicamente el mundo de las redes de sensores inalámbricas es un campo nuevo por explorar en el cual no existe un protocolo dominante. El protocolo 6LoWPAN, en su versión uIP adaptada a este tipo de dispositivos, ha sido verificado como un sistema útil para establecer comunicaciones en redes de área local.

Se percibe que con el tiempo la tendencia de esta tecnología permitirá reducir el tamaño de los componentes el cual posibilita la construcción de dispositivos económicamente más rentables y ligeros.

Se han alcanzado los objetivos establecidos al comienzo del proyecto para el cual se añadieron otros objetivos durante el desarrollo, como es la construcción de la base de datos en el servidor web.

TRABAJO FUTURO

- Diseño de un nuevo nodo sensor genérico y estandarizado para ser aplicable a diversos entornos, con el fin de tener una plataforma estándar que permita un grado de flexibilidad para diferente tipo de aplicación.

- Implementación de algoritmos multi-hop en el SO Contiki para diferente topologías de conexión, con el fin de estudiar el comportamiento y el rendimiento de la red.

- Análisis del comportamiento de redes de sensores establecidas en entornos de procesamiento cooperativo, con el fin de estudiar el desempeño de diversos sistemas de aprendizaje.

REFERENCIAS

[1] Fabián A. Gonzáles, Willian A. Mosquera, Germán D. Useche "Diseño de una red inalámbrica de sensores para apoyar actividades de agricultura de precisión en el jardin botanico de Cali" [Online].Disponible: http://bdigital.uao.edu.co/bitstream/10614/5085/1/TEK01504.pdf

[2] White paper – AEI eMOV "*Redes Sensoriales Inalámbricas*" [Online]. Disponible: http://www.google.com.co/url?sa=t&rct=j&q=&esrc=s&source=web&cd=10&ved=0CGAQFjAJ&url=http%3A%2F%2Femov.imasdtic.es%2FDescargarDocumento.aspx%3Fidd%3D3204&ei=5G_qU6OVNNPlsASQ1oDYDA&usg=AFQjCNEa9Voz0GrC0IHU2jaD9QlVpM4BZg&sig2=moLMgi7of3YKy6FJ_usFhw

[3] CISCO "Internet of Things" [Online]. Disponible: http://www.cisco.com/web/solutions/trends/iot/overview.html

[4] Daniel Villón Valdivieso "Diseño de una Red de sensores inalámbrica para agricultura de precisión" [Online]. Disponible: http://tesis.pucp.edu.pe/repositorio/bitstream/handle/123456789/266/VILLON_VALDIVIEZO_DANIEL_DISEÑO_RED_SENSORES_AGRICULTURA_PRECISION.pdf?sequence=1

[5] Samuel Alberto Agudelo Quiroz "Rede de Sensores Inalámbricos utilizando Zigbee/802.15.4 " [Online]. Disponible: http://kosmos.upb.edu.co/web/uploads/articulos/(A)_Redes_de_Sensores_Inalambricos_Utilizando_ZIGBEE802154_a05yzX_.pdf

[6] Pascual Martínez Pérez"Sistema de Alarma de Incendios basado en una red de sensores" [Online]. Disponible: http://openaccess.uoc.edu/webapps/o2/bitstream/10609/14632/8/pmartinezpeTFC0612memoria.pdf

[7] Esteban Mauricio Inga Ortega "Redes de comunicación en Smart Grid"[Online]. Disponible: http://ingenius.ups.edu.ec/documents/2497096/2497487/Art5.pdf

[8] Melisa A. Frisoli, Carlos A. Cifuentes, Anselmo Frizera, Alfonso Santiago, Ariel A. Braidot "Sensor Portable para Registro Cinemático por Comunicación ZigBee" [Online]. Disponible: http://bioingenieria.edu.ar/grupos/geic/biblioteca/j2012/Documentos/Trabajos/T12TCAr40.pdf

[9] J. Francisco Merino Torres "Nuevas Tecnologías en el seguimiento y control del Paciente Diabético" [Online]. Disponible: http://www.sediabetes.org/gestor/upload/00011420archivo.pdf

[10] Daniel Villón Valdivieso "Diseño de una red de Sensores inalámbrica para agricultura de precisión" [Online]. Disponible: http://tesis.pucp.edu.pe/repositorio/handle/123456789/266

[11] "El Estándar IEEE802.15.4 [Online]. Disponible "http://catarina.udlap.mx/u_dl_a/tales/documentos/lem/archundia_p_fm/capitulo4.pdf

[12] Escolá Ana. Cantero Josep. "*Development of a wireless sensor network with 6LoWPAN support.*" Escolá Politécnica Superior de Castelldefels, U.P. Catalunya, España, 2009. [Online]. Disponible: https://upcommons.upc.edu/pfc/bitstream/2099.1/7806/1/memoria.pdf

[13] draft-ietf-6lowpan-format-23 "Transmission of IPv6 Packets over IEEE 802.15.4 Networks" [Online]. Disponible: http://tools.ietf.org/html/draft-ietf-6lowpan-format-13

[14] Zigbee [Online]. Disponible: http://www.digi.com/technology/rf-articles/wireless-zigbee

[15] RFC 4919 "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)" [Online]. Disponible: https://datatracker.ietf.org/doc/rfc4919/

[16] Modelo OSI [Online]. Disponible: http://www.exa.unicen.edu.ar/catedras/comdat1/material/ElmodeloOSI.pdf

[17] Trama 6LoWPAN [Online]. Disponible: https://openwsn.atlassian.net/wiki/display/OW/6LoWPAN

[18] Jonathan Hui, Pascal Thubert "Compression Format for IPv6 Datagrams in 6LoWPAN Networks" [Online].Disponible: http://www.ietf.org/proceedings/74/slides/6lowpan-2.pdf

[19] Atmel Corporation (Nasdaq: ATML). Avr2015: rzraven quick start guide. [Online].Disponible:http://www.atmel.com/dyn/products/tools_docs.asp?category_id=163&family_id=676&subfamily_id=1953&tool_id=4291

[20] "Sistema Operativo CONTIKI" [Online].Disponible: http://www.contiki-os.org/

[21] Fuentes de Alternativas de Energía utilizadas en la actualidad. [Online].Disponible: http://www.planetseed.com/es/relatedarticle/fuentes-alternativas-de-energia-utilizadas-en-la-actualidad

[22] Fotorresistencia datasheet LDR [Online].Disponible http://www.electan.com/datasheets/cebek/CE-C2795.pdf

[23] Sensor LM35 datasheet [Online].Disponible http://www.ti.com/lit/ds/symlink/lm35.pdf

[24] Sensor de humedad datasheet [Online].Disponible http://datasheet.octopart.com/HIH-4000-002-Honeywell-datasheet-62840.pdf

[25] Sistema Operativo TinyOS: [Online].Disponible http://www.tinyos.net

[26] Sistema Operativo FReeRTOS [Online].Disponible http://www.freertos.org

[27] GNU General Public License [Online]. Disponible https://www.gnu.org/copyleft/gpl.html

[28] ARCH ROCK [Online].Disponible http://www.cisco.com/web/about/ac49/ac0/ac1/ac259/archrock.html

[29] Sistema Operativo CONTIKI [Online].Disponible http://www.contiki-os.org

[30] Adam Dunkels [Online].Disponible http://dunkels.com/adam/

[31] Prototheads [Online].Disponible http://dunkels.com/adam/pt/

[32] Sistema Operativo eCos [Online].Disponible http://ecos.sourceware.org

[33] Stack Rime [Online].Disponible http://dak664.github.io/contiki-doxygen/a01678.html

[34] Adam Dunkels. "The ContikiMAC Radio Duty Cycling Protocol" [Online].Disponible http://dunkels.com/Adam/dunkels11/contikimac.pdf

[35] CSMA-CA [Online].Disponible: http://searchnetworking.techtarget.com/definition/CSMA-CA.

[36] SICSlowpan [Online].Disponible http://www.iis.se/docs/SICS_Lowpan-report.pdf

[37] COOJA [Online].Disponible https://github.com/contiki-os/contiki/wiki/An-Introduction-to-Cooja.

[38] Adam Dunkels "The ContikiMAC Radio Duty Cycling Protocol" [Online]. Disponible: http://dunkels.com/adam/dunkels11contikimac.pdf

[39] Ignacio de Mula, German Ferrari, Gabriel Firme "ContikiWSN" [Online].Disponible: http://iie.fing.edu.uy/publicaciones/2011/DFF11/DFF11.pdf.pdf

[40] PhpMyAdmin [Online].Disponible: http://www.phpmyadmin.net/home_page/index.php

[41] Proteus 7 [Online].Disponible: http://www.labcenter.com/index.cfm

BIBLIOGRAFIA

- Jean-Philippe Vasseur, Adam Dunkels "Interconnecting Smart Objects With IP" Morgan Kaufmann Publishers Inc. – 2010.

- Catalina Aranzazu Suescún, Gustavo Alberto Moreno López "Revisión del Estado del Arte de Redes de Sensores Inalámbricos" Politécnico Colombiano - Revista Politécnica ISSN 1900-2351, Año 5, Número 8, 2009.

- Juan Camilo Monguí D, William Eduardo Calderón C. "Diseño de una red de sensores inalámbricos usando protocolo 6LoWPAN" Universidad Nacional de Colombia – 2012.

- Zach Shelby, Carster Bormann. "6LoWPAN: The Wireless Embedded Internet" A John Wiley and Sons, Ltd, Publication – 2009

- Ernesto J. García Davis "Implementación de Protocolos de Transporte en Redes de Sensores" Universidad Politécnica de Catalunya -2009.

- Siarhei Kuryla "Implementation and Evaluation of the Simple Network Management Protocol over IEEE 802.15.4 Radios under the Contiki Operating System" School of Engineering and Science Jacobs University Bremen gGmbH, Germany – 2010.

ANEXO A SIMULACIÓN EN COOJA DE LOS CÓDIGO CLIENTE Y SERVIDOR

Los códigos *udp-client-adc.c* y *udp-server-adc.c* son archivos que se crearon para poder establecer una conexión entre dos dispositivos, suministrando información acerca de la captura de datos del ADC.

udp-client-adc.c

```
#include "contiki.h"
#include "contiki-lib.h"
#include "contiki-net.h"
#include "util/delay.h"
#include <string.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <avr/wdt.h>
#include <util/delay.h>
#include <stdbool.h>
#include "timer.h"
#include <stdio.h>
#include <stdlib.h>

#define DEBUG DEBUG_PRINT
#include "net/uip-debug.h"
#define nop() asm volatile ("nop\n\t" ::);

#define SEND_INTERVAL   1*CLOCK_SECOND
#define MAX_PAYLOAD_LEN   120 // 3
#define ARRAY_SIZE  3
static process_event_t event_data_ready;
```

Los programas realizados con Contiki incluyen librerías del sistema operativo, gestión de periféricos y librerías de la arquitectura que se haya seleccionado, para el caso se selecciona la arquitectura AVR para configurar el microcontrolador ATMEGA 1284p. En esta configuración se establece el tiempo de envió de datos y la máxima longitud del paquete, como también se define el proceso ***event_data_ready*** que es usado exclusivamente para la comunicación de procesos.

Inicialización del ADC

```
void adc_setup(void);
unsigned int adc_read(char channel);

//se definen las funciones
/*-------------------------------------------------------------------------*/
// Configurar el conversor ADC
```

```
//*****************************************************************************
void adc_setup(void)
{
   // Vref+ y Vref-            = VCC y GND
   // Justificación de resultado = Derecha
   ADMUX = 0x40; // 0x40 referencia 3.3v // 0x80 referencia 1.1 v
   // Reloj de ADC   0x86         = F_CPU/64 = 125kHz
   // Estado del conversor       = Habilitado
   // Modo de conversión         = Manual
   ADCSRA = 0x87; //0x87 para una F_CPU/128 = 62.5kHz
   ADCSRB = 0x00;
}

//*****************************************************************************
// Lee el canal 'channel' del conversor ADC
//*****************************************************************************
unsigned int adc_read(char channel)
{
   ADMUX &= 0xF8;              //
   ADMUX |= channel;           // Seleccionar canal
   ADCSRA |= (1<<ADSC);        // Iniciar conversión
   while(ADCSRA & (1<<ADSC));  // Esperar a que termine la conversión
   return ADC;                 // Retornar resultado de conversión
}
```

Se definen las funciones ***adc_setup***, ***adc_read*** y se inicializa el ADC del micro controlador luego se realiza la lectura del mismo y se retorna el valor.

Inicialización del proceso

```
static struct uip_udp_conn *client_conn;
/*---------------------------------------------------------------------------*/
PROCESS(udp_client_process, "UDP client process");
PROCESS(sensores,"sensores");
AUTOSTART_PROCESSES(&udp_client_process,&sensores);
```

Se inicializan los procesos que interactuaran en la aplicación, UDP y lectura del ADC.

Uip_newdata

```
static void tcpip_handler(void) {
  char *str;
  if(uip_newdata()) {
    str = uip_appdata;
    str[uip_datalen()] = '\0';
```

```
    printf("Response from the server: '%s'\n",str);
     PRINTF("\n");
  }
}
```

En el momento en que se envía un dato capturado por el sensor, este realiza una espera de tiempo para verificar si el servidor lo ha recibido. Con la función ***tcpip_handler*** se espera un nuevo dato en el Transceiver con formato UDP, ***uip_newdata*** imprime un dato con un mensaje con la respuesta dada en el servidor, ***uip_newdata*** responde con un dato booleano 0 o 1 para verificar la existencia de un dato o no.

Envía los datos por el Transceiver

```
static char buf[MAX_PAYLOAD_LEN];
static int16_t buf1[MAX_PAYLOAD_LEN];

static void timeout_handler(void)
{
  printf("Client sending to: ");
  PRINT6ADDR(&client_conn->ripaddr);
  printf("  dato %d %d %d %d %d",2,adc_res[0],adc_res[1],adc_res[2],adc_res[3]);
  buf1[0]=(int16_t)2;
  buf1[1]=adc_res[0];
  buf1[2]=adc_res[1];
  buf1[3]=adc_res[2];
  buf1[4]=adc_res[3];
 sprintf(buf,"%d %d %d %d %d",2,adc_res[0],adc_res[1],adc_res[2],adc_res[3]);
  printf(" ADC:[%s]\n", buf);

uip_udp_packet_send(client_conn, buf1,6);
}
```

La función ***timeout_handler*** está diseñada para enviar los datos por el Transceiver tras un determinado tiempo. La función ***timeout_handler*** fue adecuada para que recibiera datos en un buffer para luego ser empaquetados y enviados en un paquete UDP. Las funciones importantes dentro del ***timeout handler*** son los ***sprinf*** que se encargan de guardar en un buffer el texto, ***ADC[0000 0000 0000 0000].*** La función ***uip_udp_paquet_send*** se encarga de tomar la información UDP del ***client_conn***, agrega al buffer con el valor del ADC y la longitud de valor del paquete del buffer ***buf1***, para posteriormente ser transmitida vía inalámbrica.

Etimer_set

```
static void print_local_addresses(void) {
  int i;
  uint8_t state;
```

```
  PRINTF("Client IPv6 addresses: ");
  for(i = 0; i < UIP_DS6_ADDR_NB; i++) {
    state = uip_ds6_if.addr_list[i].state;
    if(uip_ds6_if.addr_list[i].isused &&
       (state == ADDR_TENTATIVE || state == ADDR_PREFERRED)) {
      PRINT6ADDR(&uip_ds6_if.addr_list[i].ipaddr);
      PRINTF("\n");
    }
  }
}
/*---------------------------------------------------------------------------*/
#if UIP_CONF_ROUTER
static void set_global_address(void) {
  uip_ipaddr_t ipaddr;
  uip_ip6addr(&ipaddr, 0xaaaa, 0, 0, 0, 0, 0, 0, 0);
  uip_ds6_set_addr_iid(&ipaddr, &uip_lladdr);
  uip_ds6_addr_add(&ipaddr, 0, ADDR_AUTOCONF);
}
#endif /* UIP_CONF_ROUTER */
/*---------------------------------------------------------------------------*/
static void set_connection_address(uip_ipaddr_t *ipaddr) {

uip_ip6addr(ipaddr,0xfe80,0,0,0,0x0200,0x1,0x1,0x101);
}
PROCESS_THREAD(sensores, ev, data)
{
  static struct etimer et;

PROCESS_BEGIN();
  event_data_ready=process_alloc_event();
  adc_setup();

  etimer_set(&et,2*CLOCK_SECOND);
  while(1){
    PROCESS_WAIT_EVENT_UNTIL(ev==PROCESS_EVENT_TIMER);

    adc_res[0] = adc_read(0);
    adc_res[1] = adc_read(1);
    adc_res[2] = adc_read(2);
    adc_res[3] = adc_read(3);

    sprintf(adc_result_string, "%04d,%04d,%04d,%04d,%04d", 2, adc_res[0], adc_res[1],
adc_res[2], adc_res[3]/**/);
    printf("ADC Result [%s] \n", adc_result_string);
    process_post(&udp_client_process, event_data_ready, &adc_result_string);
    etimer_reset(&et);
  }
PROCESS_END();
}
```

En esta sección se realiza la impresión de ***print_local_addresses***, asignación de la dirección IPv6 global con la función ***set_global_address***, la asignación manual de la dirección con ***set_connection_address*** y el proceso "***sensores***" se encarga de capturar la información del ADC cada dos segundos está habilitando el ***etimer_set***, la función ***porcess_post*** Recibe al proceso al que se quiere seguir registrando en las variables ***event_data_ready*** y con ellos se lleva la información apuntada por ***adc_resul_string***.
Al final se realiza un reset del temporizador ***etimer***.

UDP general

```
PROCESS_THREAD(udp_client_process, ev, data)
{
  static struct etimer et;
  uip_ipaddr_t ipaddr;

  PROCESS_BEGIN();
  PRINTF("UDP client process started\n");

#if UIP_CONF_ROUTER
  set_global_address();
#endif

  print_local_addresses();
  set_connection_address(&ipaddr);
```

El proceso UDP, se establece con un reloj ***etimer***, se declara la dirección IP luego se inicializa el proceso y se imprime los resultados del arranque del proceso, las siguientes funciones fijan la dirección IP basándose en los archivos MAC ingresados por Contiki, por último se realiza una impresión de la dirección local y se llama a la función ***set_connection_address*** para establecer conexión con la dirección IP asignada.

INSTALACION DE LA CONECCION UDP

```
  /* new UDP connection with remote host */
  client_conn = udp_new(&ipaddr, UIP_HTONS(3000), NULL);
  udp_bind(client_conn, UIP_HTONS(3001));
```

La Instalación de la conexión por UDP se incluye al proceso principal del aplicativo, primero se crea una conexión y después se asigna a un puerto. ***udp_news*** se encarga de realizar la instalación y tiene como formato ***udp_new(ip, puerto, appstate),*** Se asigna una IP remota y un puerto a la aplicación lo que solo recibirá datos de esa dirección, pero llegado el caso de recibir de una combinación de IPs y puertos se debería escribir ***NULL*** en campo de IPs remoto y en el puerto de valor cero. La variable ***client_conn*** que guarda la conexión UDP, es declarada al inicio del programa como una variable global antes de los procesos. Para el caso se utilizó ***ipaddr*** como dirección y puerto 3001.

While

```
  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, SEND_INTERVAL);

  while(1) {
    PROCESS_YIELD();
    if(etimer_expired(&et)) {
      //timeout_handler((char*)data);
      timeout_handler();
      etimer_restart(&et);
    } else if(ev == tcpip_event) {
      tcpip_handler();
    }
  }
  PROCESS_END();
}
```

Por último se crea un while el cual gestiona el envío del dato recibido o espera la respuesta del servidor. Uno de los dos casos está activo en cada tiempo. El valor tomado como dato va desde el sensor conectado a los puertos habilitados del ADC del microcontrolador, este proceso es tomado por Contiki ***sensores***, quien cada cierto tiempo hace el traspaso del proceso ***Contiki udp_client_process***, para al final ser tomado por la función ***timeout_handler*** quien se encarga de enviar la información por la radio.

```
392       ID:1
395       ID:1   Checking MCUSR...
399       ID:1   CLOCK_SECOND 128
403       ID:1   RTIMER_ARCH_SECOND 7812
406       ID:1   F_CPU 8000000
406       ID:1
413       ID:1   *******Booting Contiki 2.5*******
419       ID:1   rng issues 238
422       ID:1   Initial node_id 1
430       ID:1   EEPROM is corrupt, rewriting with defaults.
801       ID:1   EUI-64 MAC: 0-0-0-1-0-1-1-1
814       ID:1   CSMA ContikiMAC, channel 26 , check rate 8 Hz tx power 0
816       ID:1   RPL Enabled
819       ID:1   Routing Enabled
825       ID:1   UDP client process started
833       ID:1   Client IPv6 addresses: aaaa::200:1:1:101
836       ID:1   fe80::200:1:1:101
852       ID:1   Created a connection with the server fe80::200:1:1:101 local/remote port 3001/3000
853       ID:1   Online
929       ID:1   q
1856      ID:1   Client sending to: fe80::200:1:1:101  dato 2 0 0 0 0 ADC:[2 0 0 0 0]
2742      ID:1
2745      ID:1   Addresses [4 max]
2749      ID:1   aaaa::200:1:1:101
2752      ID:1   fe80::200:1:1:101
2753      ID:1
2756      ID:1   Neighbors [10 max]
2760      ID:1   fe80::200:1:1:101
2760      ID:1
2763      ID:1   Routes [4 max]
2765      ID:1     <none>
2767      ID:1   ---------
2852      ID:1   ADC Result [0002,0000,0000,0000,0000]
2996      ID:1   Client sending to: fe80::200:1:1:101  dato 2 0 0 0 0 ADC:[2 0 0 0 0]
```

***Figura A.1**: Arranque udp-client-adc*

PROGRAMA SERVIDOR. El software del lado del servidor únicamente difiere en que éste no está constantemente enviando mensajes, sino que solo espera datos que lleguen a su transceiver mediante las funciones ***tcpip_handler*** y ***uip_newdata***.

Servidor

```
static uint16_t dato_[4];
static struct uip_udp_conn *server_conn;

PROCESS(udp_server_process, "UDP server process");
PROCESS(sensores,"sensores");
AUTOSTART_PROCESSES(&udp_server_process,&sensores);

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

static char buf[MAX_PAYLOAD_LEN];
static void
timeout_handler(void)
{
 int16_t * pt = (int16_t*)uip_appdata;
```

```
 printf("Server sending to: ");
  PRINT6ADDR(&server_conn->ripaddr);
  sprintf(buf,"%d %d %d %d %d",2,adc_res[0],adc_res[1],adc_res[2],adc_res[3]);
  printf(" ADC:[%s]\n", buf);
  uip_udp_packet_send(server_conn, buf, 30);
}
/*---------------------------------------------------------------------------*/
static void
tcpip_handler(void)
{
static int seq_id;
  int r=0;
  char buf[MAX_PAYLOAD_LEN];
  int16_t buf3[MAX_PAYLOAD_LEN];
  char buf4[MAX_PAYLOAD_LEN];

  if(uip_newdata()) {
//PRINTF("client:[%d %d %d %d %d]", *pt,*(pt+1),*(pt+2) );
 int16_t *pt = (int16_t*)uip_appdata;
 //buf3 = uip_appdata;
 buf3[0] = *pt;
 buf3[1] = *(pt+1);
 buf3[2] = *(pt+2);
 buf3[3] = *(pt+3);
 buf3[4] = *(pt+4);

 PRINTF("Server received: [%d %d %d %d %d] from ",buf3[0],buf3[1],buf3[2],buf3[3],buf3[4]);
  PRINT6ADDR(&UIP_IP_BUF->srcipaddr); // dirección ip del cliente
  PRINTF("\n"); //recibe el dato del cliente en la variable uip_appdata

    PRINTF("data Client:[%s]",(char *)uip_appdata);
    PRINTF("\n");

uip_ipaddr_copy(&server_conn->ripaddr, &UIP_IP_BUF->srcipaddr);
sprintf(buf,"conection ok! (%d)", ++seq_id);
    PRINTF("%s\n",buf);

    uip_udp_packet_send(server_conn, buf, 20) ;
    /* Restore server connection to allow data from any node */
    memset(&server_conn->ripaddr, 0, sizeof(server_conn->ripaddr));
  }
}
static void print_local_addresses(void) {
  int i;
  uint8_t state;

  PRINTF("Client IPv6 addresses: ");
  for(i = 0; i < UIP_DS6_ADDR_NB; i++) {
    state = uip_ds6_if.addr_list[i].state;
```

```
    if(uip_ds6_if.addr_list[i].isused &&
       (state == ADDR_TENTATIVE || state == ADDR_PREFERRED)) {
      PRINT6ADDR(&uip_ds6_if.addr_list[i].ipaddr);
      PRINTF("\n");
    }
  }
}
/*---------------------------------------------------------------------------*/
PROCESS_THREAD(udp_server_process, ev, data)
{
  static struct etimer et;
#if UIP_CONF_ROUTER
  uip_ipaddr_t ipaddr;
#endif /* UIP_CONF_ROUTER */

  PROCESS_BEGIN();
  PRINTF("UDP server started\n");

#if UIP_CONF_ROUTER
  uip_ip6addr(&ipaddr, 0xaaaa, 0, 0, 0, 0, 0, 0, 0);
  uip_ds6_set_addr_iid(&ipaddr, &uip_lladdr);
  uip_ds6_addr_add(&ipaddr, 0, ADDR_AUTOCONF);
#endif /* UIP_CONF_ROUTER */

  print_local_addresses();

  server_conn = udp_new(NULL, UIP_HTONS(3001), NULL);
  udp_bind(server_conn, UIP_HTONS(3000));
etimer_set(&et,CLOCK_SECOND);
  while(1) {
    PROCESS_YIELD();
    if(etimer_expired(&et)) {
      timeout_handler();
      etimer_restart(&et);
    } else if(ev == tcpip_event) {
      tcpip_handler();
    }
  }
  printf("end!\n");
  PROCESS_END();
}
```

```
392      ID:1
395      ID:1   Checking MCUSR...
398      ID:1   CLOCK_SECOND 128
403      ID:1   RTIMER_ARCH_SECOND 7812
406      ID:1   F_CPU 8000000
406      ID:1
412      ID:1   *******Booting Contiki 2.5*******
418      ID:1   rng issues 238
422      ID:1   Initial node_id 1
430      ID:1   EEPROM is corrupt, rewriting with defaults.
800      ID:1   EUI-64 MAC: 0-0-0-1-0-1-1-1
813      ID:1   CSMA ContikiMAC, channel 26 , check rate 8 Hz tx power 0
816      ID:1   RPL Enabled
819      ID:1   Routing Enabled
824      ID:1   UDP server started
831      ID:1   Server IPv6 addresses:aaaa::200:1:1:101
835      ID:1   fe80::200:1:1:101
836      ID:1   Online
929      ID:1   q
2742     ID:1
2745     ID:1   Addresses [4 max]
2749     ID:1   aaaa::200:1:1:101
2752     ID:1   fe80::200:1:1:101
2752     ID:1
2756     ID:1   Neighbors [10 max]
2758     ID:1     <none>
2761     ID:1   Routes [4 max]
2762     ID:1     <none>
2764     ID:1   ---------
3747     ID:1   Never-used stack > 2680 bytes
5948     ID:1   ADC Result [0001,0000,0000,0000,0000]
5955     ID:1   Server sending to: :: ADC:[2 0 0 0 0]
```

***Figura A.2**: Arranque udp-server-adc*

En el simulador COOJA se puede observar el comportamiento de la comunicación realizada por las dos aplicaciones cliente, servidor. Se encuentra que el cliente envía mensajes UDP al ATmega1284p del servidor de forma periódica, el servidor responde con un mensaje UDP una vez recibido el mensaje desde el cliente.

```
3946     ID:1   Server received: [2 0 0 28521 8302] from fe80::200:2:2:202
3949     ID:1   data Client:[]
3953     ID:1   conection ok! (3)
4052     ID:2   Response from the server: 'conection ok! (2)'
4052     ID:2
4062     ID:2   Response from the server: 'conection ok! (3)'
4062     ID:2
4705     ID:2   ADC Result [0002,0000,0000,0000,0000]
4930     ID:2   Client sending to: fe80::200:1:1:101  dato 2 0 0 0 0 ADC:[2 0 0 0 0]
5696     ID:1   Server received: [2 0 0 28521 8302] from fe80::200:2:2:202
5699     ID:1   data Client:[]
5702     ID:1   conection ok! (4)
5942     ID:2   Client sending to: fe80::200:1:1:101  dato 2 0 0 0 0 ADC:[2 0 0 0 0]
6006     ID:1   ADC Result [0001,0000,0000,0000,0000]
6013     ID:1   Server sending to: :: ADC:[2 0 0 0 0]
6550     ID:2   Response from the server: 'conection ok! (4)'
6550     ID:2
6755     ID:2   ADC Result [0002,0000,0000,0000,0000]
6949     ID:2   Client sending to: fe80::200:1:1:101  dato 2 0 0 0 0 ADC:[2 0 0 0 0]
```

***Figura A.3**: Cliente enviado y Servidor Respondiendo*

ANEXO B TARJETA NODO-SENSOR PCB

ESQUEMÁTICO NODO-SENSOR

El desarrollo la tarjeta del nodo sensor fue hecho con la herramienta Proteus 7.9 [41], en la cual se llevan incorporados los sensores de medición, las figuras B1 y B2 muestran el diseño de la tarjeta creada.

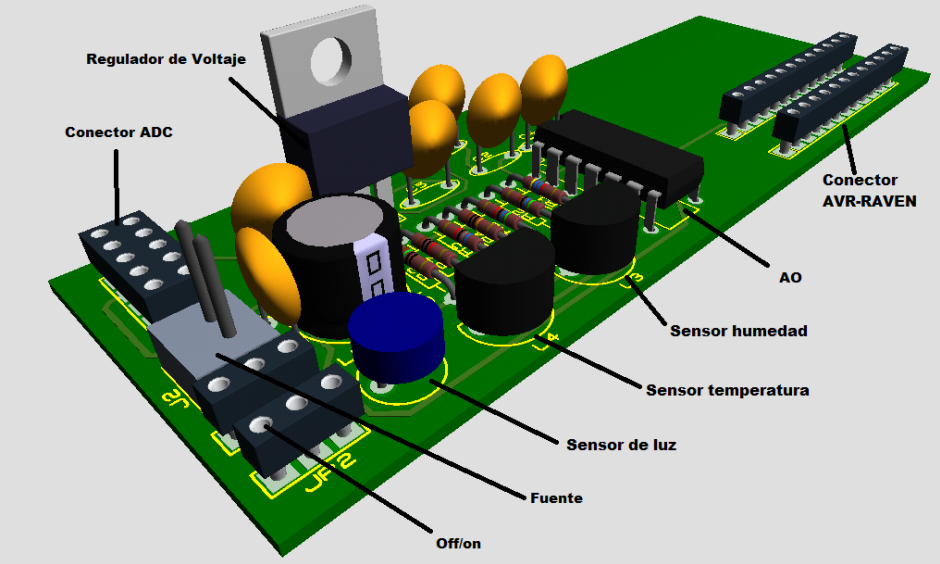

***Figura B.1**: Tarjeta nodo-sensor*

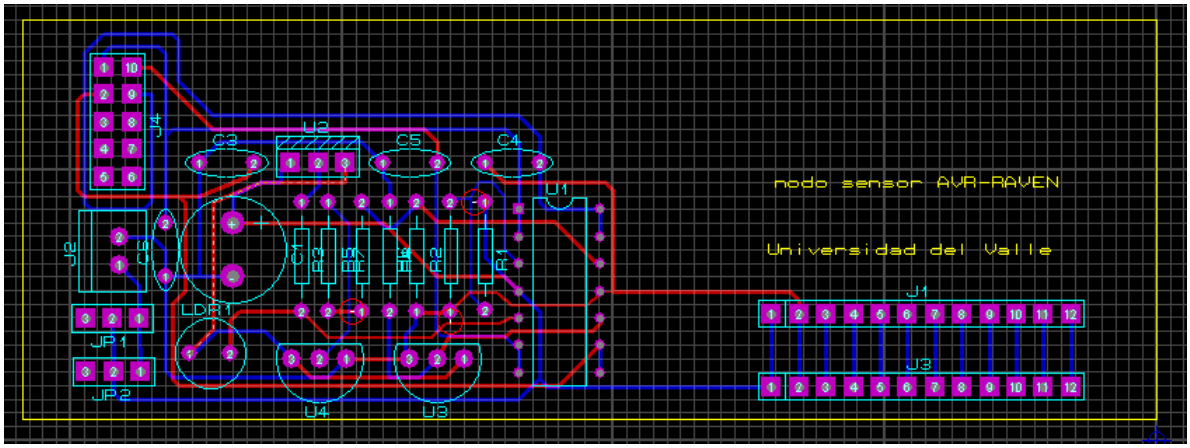

***Figura B.2**: Esquemático nodo sensor*

ANEXO C DISEÑO DE UN PROYECTO DE SIMULACION EN COOJA

Dentro de las herramientas de Contiki viene una carpeta en donde se encuentran las herramientas necesarias para poder depurar el código de las motas en un entorno de simulación

Primero se abre una terminal y se dirige hacías la carpeta en la cual se encuentran las herramientas (tools) de Contiki. Dentro de ella se encuentra el simulador Cooja el cual se ejecuta digitando el comando:

```
sudo ant run
```

Con permisos de usuario y se abre una interfaz como lo ilustra la figura C.1 :

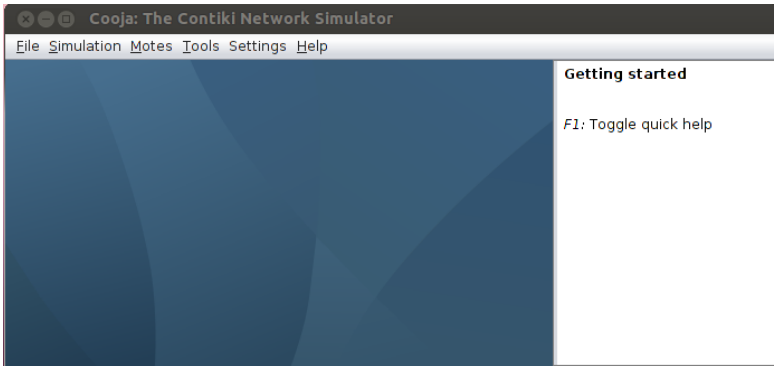

***Figura C.1**: Entrono de Simulación*

Para crear una nueva simulación se va a *files* luego a *New Simulation* y se crea una nueva simulación figura C.2.

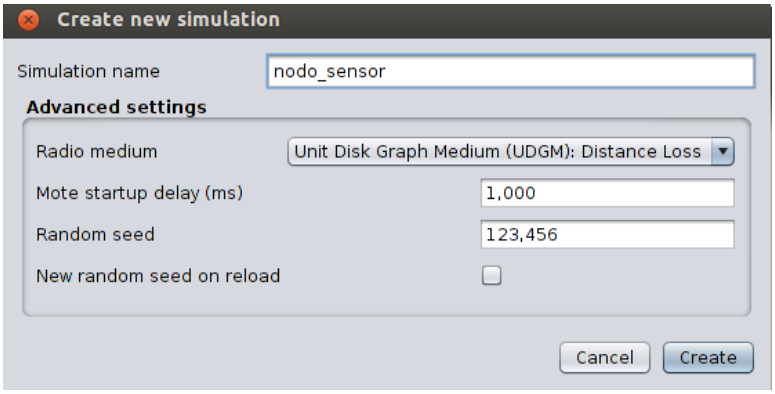

***Figura C.2**: Creación de proyecto COOJA.*

Luego de crear un proyecto se despliega esta la interfaz ilustrada por la figura C.3.

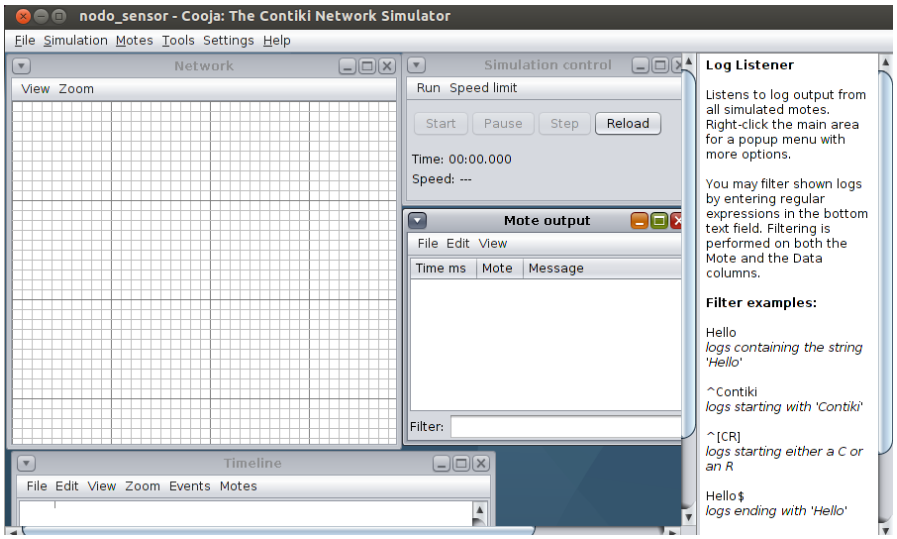

***Figura C.3**: Interfaz de simulación COOJA*

Para crear una mota tipo avr-raven figura C.4, se va a la barra de elementos y se dirige a Motes, luego Add Motes /Créate new mote type /Raven Mote pype.

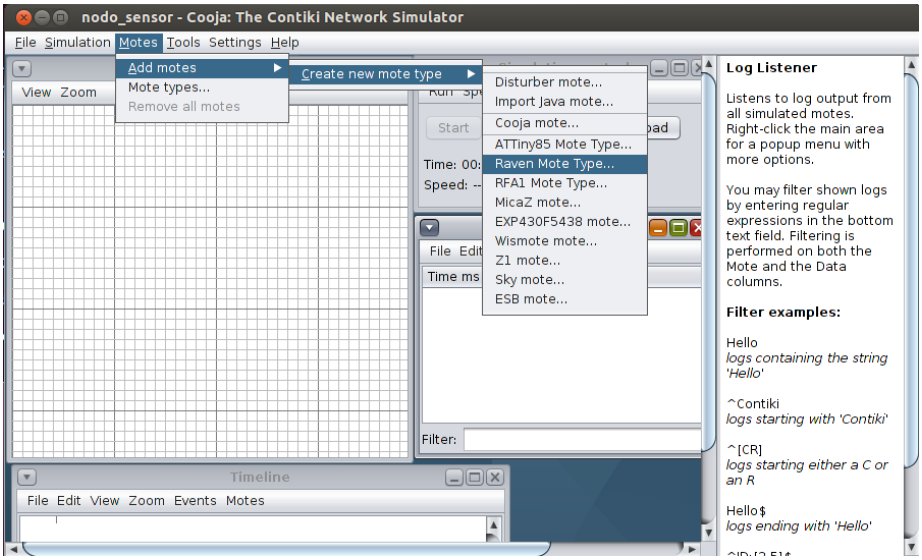

***Figura C.4**: Creación tipo de mota*

Luego en contiki process / firmware se dirigiré hacia la carpeta *Examples* en la cual se alojan los programas en la carpeta *udp-ipv6-tesis* para el cual se modifica los archivos *udp_client.c* y *udp_server.c*, en este caso se ha renombrado como *udp-client-adc.c* y *udp-server-adc.c* para realizar

la aplicación, en la gráfica C.5 muestra la configuración de un nodo servidor encargado de recibir los datos de tres nodos clientes.

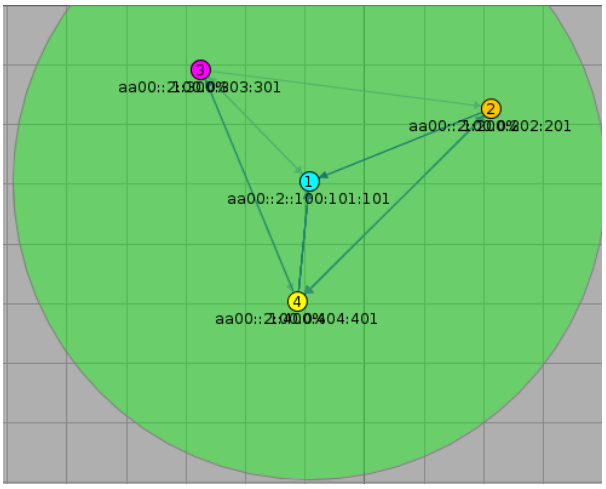

***Figura C.5**: Visualización de conexión para 3 nodos cliente y un nodo servidor*

Dentro de la herramienta Cooja se encuentra un visualizado del muestreo del tiempo. Permitiendo así visualizar el comportamiento de la comunicación como lo representa la figura C.6.

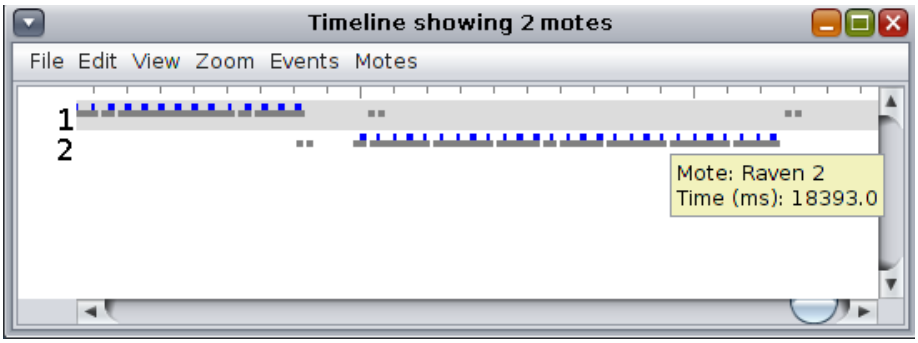

***Figura C.6**: Línea de tiempo de comportamiento de los nodo sensores*

La figura C.7 ilustra la conexión establecida por el cliente y el servidor haciendo uso del simulador cooja

```
3946     ID:1   Server received: [2 0 0 28521 8302] from fe80::200:2:2:202
3949     ID:1   data Client:[]
3953     ID:1   conection ok! (3)
4052     ID:2   Response from the server: 'conection ok! (2)'
4052     ID:2
4062     ID:2   Response from the server: 'conection ok! (3)'
4062     ID:2
4705     ID:2   ADC Result [0002,0000,0000,0000,0000]
4930     ID:2   Client sending to: fe80::200:1:1:101  dato 2 0 0 0 0 ADC:[2 0 0 0 0]
5696     ID:1   Server received: [2 0 0 28521 8302] from fe80::200:2:2:202
5699     ID:1   data Client:[]
5702     ID:1   conection ok! (4)
5942     ID:2   Client sending to: fe80::200:1:1:101  dato 2 0 0 0 0 ADC:[2 0 0 0 0]
6006     ID:1   ADC Result [0001,0000,0000,0000,0000]
6013     ID:1   Server sending to: :: ADC:[2 0 0 0 0]
6550     ID:2   Response from the server: 'conection ok! (4)'
6550     ID:2
6755     ID:2   ADC Result [0002,0000,0000,0000,0000]
6949     ID:2   Client sending to: fe80::200:1:1:101  dato 2 0 0 0 0 ADC:[2 0 0 0 0]
```

***Figura C.7**: Comunicación establecida en el simulación COOJA*

El simulador permite corroborar que la conexión ente los nodos ha sido establecía, esto es visualizado por una interfaz que permite ver el comportamiento de la radio de los dispositivos simulados, figura C.8.

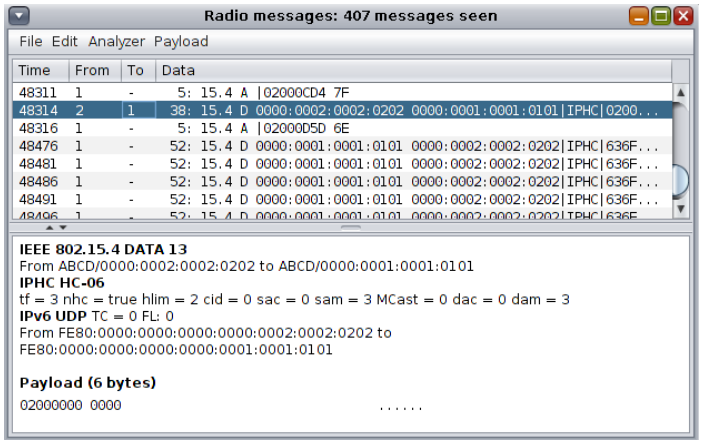

***Figura C.8**: Comprobación de trafico de Red*

ANEXO D INSTALACIÓN Y CONFIGURACIÓN DEL SERVIDOR APACHE

Para configurar servidor web ya sea local se ejecuta una serie de comandos en Ubuntu Linux 12.04 necesarios para configurar el servidor apache, modulo de php, gestor de base de datos MSQL y un administrador de base de datos phpMyadmin.

1. actualización de repositorios
   sudo apt-get update
2. instalar el servidor apache
   sudo apt-get install apache2
3. reiniciar servidor apache
   sudo /etc/init.d/apache2 restart
4. instalar el servidor MySQL
   sudo apt-get install mysql-server mysql-common mysql.client
5. se confirma contraseña para el servidor en este acaso le colocamos user
   comprobar si MySQL quedo bien instalado
   mysql –u root –p
6. instalar el modulo de PHP
   sudo apt-get install php5 libapache2-mod-php5 php5-mysql
7. para habilitar la carpeta donde se alojan las sentencias php y html se debe habilitar los permisos y cambiar los permisos del root
   sudo chown user:adm /var/www
   user es uno de usuario de la cuenta el cual pertenece a los administradores.
8. instalar el gestor de la base de datos
   sudo apt-get install phpmyadmin
9. crear un enlace simbólico para poder realizar el ingreso al gestor de base de datos phpmyadmin
   sudo ln -s /usr/share/phpmyadmin /var/www/

Configuración del servidor.

1. Se debe tener instalado en el equipo los siguientes paquetes: Apache2, php5, php-pear, postgresql, pgadmin3, en caso de poseerlos en otro apartado se describirá el proceso de instalación de estos.
2. De debe tener a la mano los archivos de configuración: apache.conf, la carpeta con los archivos del proyecto sensores.tar.gz.
3. Inicia sesión como administrador sudo su, solicita la confirmación de contraseña
4. Digita en la consola lo siguiente: root@VirtualBox-13:/#nautilus
5. Luego se hace doble clic en la carpeta etc y en la carpeta apache2
6. Se debe renombrar el archivo apache2.conf que se encuentra la primera vez que se instala apache, se cambia el nombre por apache2_copia.conf.
7. Se abre el archivo anterior con el editor de texto y llega hasta la parte final de este, se busca la siguiente linea <VirtualHost 127.0.0.1:80> y cambiala por: <VirtualHost *:80>
8. Vuelve de nuevo a /etc, pero ahora busca el archivo hosts que se encuentra en esta carpeta.
9. Abre el archivo anterior con el editor y llega hasta la parte final de este, coloca lo siguiente: **127.0.0.1 sensor.localhost** antes de la siguiente línea: **# The following lines are desirable for IPv6 capable hosts.**
10. Cierra el archivo y cierra la ventana de nautilus

11. Luego en la consola escribe el comando **clear**
12. Ahora se reinicia apache de esta manera: **/etc/init.d/apache2 restart**
13. Se debe crear una carpeta en /home, que se la llamo **sfprojects**
14. Dentro de **/home/sfprojects** se coloca el archivo llamado **sensores** en el cual se encuentra contenido los archivos de la aplicación.
15. Se abre una consola se digita root@nombre_equipo:/#**chmod –R 777 sensores**, para dar todos los permisos a la carpeta, si se quiere cambiar al propietario de la carpeta se digita **chown nombre_grupo:nombre_usuario nombre_carpeta**
16. Luego se abre una ventana del navegador y digita sensor.localhost y debería aparecer la página inicial del proyecto.

ANEXO E CODIGO DE ESCUCHA DEL PUERTO

Server6.c

```
/*
  Example IPv6 UDP server.
  Copyright (C) 2010 Russell Bradford

  This program is free software; you can redistribute it and/or
  modify it under the terms of the GNU General Public License
  as published by the Free Software Foundation; either version 2
  of the License, or (at your option) any later version.

  This program is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License (http://www.gnu.org/copyleft/gpl.html)
  for more details.
*/

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <time.h>

#define PORT 3000
#define MESSAGE "ok cliente!!"


FILE*datos;
FILE*stream;
```

```
int main(void)
{
  int sock;
 char cont=0;
  socklen_t clilen;

  //time stuff
  char tid[28];
  struct tm *local;
  time_t t;
  //end time stuff

  struct sockaddr_in6 server_addr, client_addr;
  int16_t buffer[1024];
  //int buffer[24];
  char addrbuf[INET6_ADDRSTRLEN];

  /* create a DGRAM (UDP) socket in the INET6 (IPv6) protocol */
  sock = socket(PF_INET6, SOCK_DGRAM, 0);

  if (sock < 0) {
    perror("creating socket");
    exit(1);
  }

#ifdef V6ONLY
  // setting this means the socket only accepts connections from v6;
  // unset, it accepts v6 and v4 (mapped address) connections
  { int opt = 1;
    if (setsockopt(sock, IPPROTO_IPV6, IPV6_V6ONLY, &opt, sizeof(opt)) < 0) {
      perror("setting option IPV6_V6ONLY");
      exit(1);
    }
  }
#endif

  /* create server address: this will say where we will be willing to
     accept datagrams from */

  /* clear it out */
  memset(&server_addr, 0, sizeof(server_addr));

  /* it is an INET6 address */
  server_addr.sin6_family = AF_INET6;

  /* the client IP address, in network byte order */
  /* in this example we accept datagrams from ANYwhere */
  server_addr.sin6_addr = in6addr_any;
```

```
  /* the port we are going to listen on, in network byte order */
  server_addr.sin6_port = htons(PORT);

  /* associate the socket with the address and port */
  if (bind(sock, (struct sockaddr *)&server_addr,
              sizeof(server_addr)) < 0) {
    perror("bind failed");
    exit(2);
  }

  while (1) {

    /* now wait until we get a datagram */
    printf("waiting for a datagram...\n");
    clilen = sizeof(client_addr);

    if (recvfrom(sock, buffer, 1024, 0,
                        (struct sockaddr *)&client_addr,
                        &clilen) < 0) {
      perror("recvfrom failed");
      exit(4);
    }

    /* now client_addr contains the address of the client */
    printf("got '%d' from %s\n", buffer,inet_ntop(AF_INET6, &client_addr.sin6_addr, addrbuf,
INET6_ADDRSTRLEN));

if (cont<=14){

 cont ++;

unsigned char buf[12];
unsigned  int datoT0 = buffer[0];

float datoT1 = ((1.8*buffer[1]/1023)*(9.67)); // fuente a 6.3V

float datoT2 =((1.8*buffer[2]/1023)*1000);  // fotorresistencia

float datoT3 = ((1.8*buffer[3]/1023)*100);// temperatura

float datoT4 = (((1.8*(1.1*buffer[4])/(0.5822*1023)-0.7439))/0.032); // humedad

t=time(NULL);
local=localtime(&t);

sprintf(buf,"%0d %0.2f %0.2f %0.2f %0.2f",datoT0,datoT1,datoT2,datoT3,datoT4);
printf ("contador[%d]\n",cont);
printf ("ADC[%s]\n",buf);
```

```
printf("%s %s\n",buf,inet_ntop(AF_INET6, &client_addr.sin6_addr,addrbuf,
INET6_ADDRSTRLEN));

stream = fopen("D:\\Datos.txt","a+");

fprintf (stream,"[%d] ",cont);
fprintf(stream,"hora %02d:%02d:%02d \n",local->tm_hour,local->tm_min, local->tm_sec);

fprintf(stream,"%0d %0.2f %0.2f %0.2f %0.2f  ",datoT0,datoT1,datoT2,datoT3,datoT4);

fprintf(stream, "%s\n",inet_ntop(AF_INET6, &client_addr.sin6_addr,addrbuf,
INET6_ADDRSTRLEN));

fseek(stream, cont, SEEK_SET);

fclose(stream);
}
else { cont=0;
  remove( "D:\\Datos.txt");
}
    printf("sending message back\n");

    if (sendto(sock, MESSAGE, sizeof(MESSAGE), 0,
            (struct sockaddr *)&client_addr,
                sizeof(client_addr)) < 0) {
      perror("sendto failed");
      exit(5);
    }
  }
  return 0;
}
```

ANEXO F ARCHIVO GENERADO POR Server6.c

Datos.txt

```
16:38:00 1 6.65 100.57 25.18 68.00  fe80::ff:fe00:1
16:38:05 4 6.65 106.58 25.18 68.00  fe80::ff:fe00:1
16:38:08 1 6.64 100.30 25.00 68.64  fe80::ff:fe00:1
16:38:21 1 6.60 100.82 24.68 68.69  fe80::ff:fe00:1
16:38:24 1 6.63 104.23 25.00 68.64  fe80::ff:fe00:1
16:38:29 1 6.64 106.82 25.68 68.31  fe80::ff:fe00:1
16:38:32 1 6.62 106.00 25.00 68.64  fe80::ff:fe00:1
16:38:45 1 6.64 106.82 25.68 68.56  fe80::ff:fe00:1
16:38:49 1 6.63 100.70 25.20 68.64  fe80::ff:fe00:1
16:38:53 1 6.64 103.83 25.63 68.81  fe80::ff:fe00:1
```

ANEXO G CREAR LA BASE DE DATOS CON PHYMYADMIN

Pasos para crear una base de datos:

1- Introducir el nombre de la base de datos y establecer el cotejamiento utf8_general_ci posteriormente hacer clic en crear
2- A continuación se introduce el nombre para cada tabla incluida en la base de datos creada, en este caso son nodo1 y nodo2.
3- Se establece en número de campos (filas) para cada tabla, como id, código, dirección_ip, nombre, temperatura, humedad, luz, voltaje, fecha y hora. Se hacer click en continuar.
4- Se indicar el tipo de cada campo (in,float,char..) y finalmente guardar.

ANEXO H IMPLEMENTACIÓN DE LA APLICACIÓN EN LA PLATAFORMA

IMPLEMENTACIÓN PROGRAMA CLIENTE Para generar los archivos binarios de la plataforma ATmega 1284p, es necesario estar dentro de la carpeta u*dp-ipv6-adc* en la que se encuentran alojados los programas, se digita el comando.

```
make    TARGET=avr-raven
```

El cual genera los archivos *udp-server.avr-raven* y *udp-client.avr-raven*. Estos son archivos .ELF, los cuales no puede ser implementados en la arquitectura, por lo tanto se necesita generar los archivos *client-adc.hex* y *server-adc.hex;* para esta tarea se digita para el caso del cliente como en el servidor en la consola los comandos.

```
avr-objcopy -R .eeprom -R .fuse -R .signature -O ihex udp-client-adc.avr-raven client-adc.hex
avr-objcopy -j .eeprom --set-section-flags=.eeprom="alloc,load" --change section-lma .eeprom=0 -O ihex udp-client-adc.avr-raven client-adc.eep
```

El resultado de este proceso generara el archivo *client-adc.hex* que tiene el formato pertinente para ser cargado en la arquitectura, como también se genera el archivo *client-adc.eep* que tiene la dirección MAC del nodo, Esta puede ser escrita en la memoria EEPROM.

Para implementar el archivo en el ATMEGA1284p es necesario tener instalado la librería avrdude, si no se tiene instalado se digita en consola. Aunque se puede realizar este proceso haciendo uso del programa AVR-STUDIO de ATMEL soportado para Windows:

```
sudo apt-get install avrdude
```

Con la ayuda del programador AVR DRAGON [25] de la empresa ATMEL se digita los comandos de Linux para programar la arquitectura.

```
sudo avrdude -p m1284p -c dragon_jtag -P usb –U flash:w:client-adc.hex
sudo avrdude -p m1284p -c dragon_jtag -P usb –U eeprom:w:client-adc.eep
```

ANEXO G ESQUEMÁTICO NODO SENSOR