ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

FACULTAD DE INFORMÁTICA Y ELECTRÓNICA

ESCUELA DE INGENIERÍA EN ELECTRÓNICA, TELECOMUNICACIONES Y REDES

EVALUACIÓN DEL RENDIMIENTO DE PLATAFORMAS 6LOWPAN, APLICADO AL DISEÑO DEL MONITOREO ESTRUCTURAL DE PUENTES

Trabajo de titulación presentado para optar al grado académico de:

INGENIERO EN ELECTRÓNICA TELECOMUNICACIONES Y REDES

AUTOR: BYRON MAURICIO PALATE SUPE

TUTOR: ING. ALBERTO ARELLANO

Riobamba-Ecuador

2016

ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

FACULTAD DE INFORMÁTICA Y ELECTRÓNICA

ESCUELA DE INGENIERÍA EN ELECTRÓNICA, TELECOMUNICACIONES Y REDES

El Tribunal del Trabajo de Titulación certifica que: El trabajo de Investigación: **"EVALUACIÓN DEL RENDIMIENTO DE PLATAFORMAS 6LOWPAN, APLICADO AL DISEÑO DEL MONITOREO ESTRUCTURAL DE PUENTES"**, de responsabilidad del señor Byron Mauricio Palate Supe, ha sido minuciosamente revisado por los miembros del Tribunal del Trabajo de Titulación, quedando autorizada su presentación.

NOMBRE	FIRMA	FECHA
Dr. Miguel Tasambay, PhD. **DECANO FACULTAD DE INFORMÁTICA Y ELECTRÓNICA**	_____	_____
Ing. Franklin Moreno **DIRECTOR DE ESCUELA DE INGENIERÍA EN ELECTRÓNICA, TELECOMUNICACIONES Y REDES**	_____	_____
Ing. Alberto Arellano. **DIRECTOR DEL TRABAJO DE TITULACIÓN**	_____	_____
Ing. Vinicio Ramos, MSc. **MIEMBRO DEL TRIBUNAL**	_____	_____

NOTA DEL TRABAJO DE TITULACIÓN ESCRITA: _____

"Yo, Byron Mauricio Palate Supe soy el responsables de las ideas, doctrinas y resultados expuestos en este Trabajo de Titulación y el patrimonio intelectual del Trabajo de Titulación pertenecen a la ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO".

_____

Byron Mauricio Palate Supe

DEDICATORIA

Dedico el presente trabajo de manera especial a mi madre, pues fue el principal cimiento para la construcción de mi vida profesional, sentó en mí las bases de responsabilidad y deseos de superación, por ser padre y madre a la vez, por la confianza depositada a lo largo de mi vida principalmente en esta etapa, por el inmenso esfuerzo realizado para darme el estudio desde muy pequeño, todo ese sacrificio se ve hoy reflejado al terminar con éxito mi carrera universitaria.

A mi hermano Santiago y a mi padre Ángel, los cuales fueron la razón de seguir adelante cuando me sentía derrotado, y aunque ya no estén conmigo siempre contaba con su apoyo, sabiendo que se hubiesen sentido orgullosos de mí al cumplir la promesa de alcanzar esta meta y junto a ella varios logros académicos.

Byron.

AGRADECIMIENTOS

A mi madre y mi hermano Paúl, por ser el pilar fundamental en mi vida, guiarme con sus enseñanzas y concejos para cumplir mis metas, sin importar los obstáculos que se presentaron en la familia y en la vida misma.

A mis maestros que han sabido compartir sus conocimientos, apoyo, paciencia y ante todo brindarme su amistad, ya que gracias a ello hoy puedo terminar una etapa más en mi vida.
A todos mis compañeros y amigos por brindarme su apoyo incondicional en todo momento.

Madre Dolorosa Gracias por todo…

Byron.

TABLA DE CONTENIDO

INDICE DE TABLAS

INDICE DE FIGURAS

ÍNDICE DE ANEXOS

RESUMEN

La evaluación del rendimiento de plataformas 6LoWPAN, aplicado al diseño del monitoreo estructural de puentes, desarrollado en la Escuela Superior Politécnica de Chimborazo, Facultad de Informática y Electrónica, se inició con la revisión del concepto internet de las cosas y Smart City, de igual manera el estado del arte del estándar 6LowPan para poder conocer sus principales características como arquitectura, configuración, seguridad. Conjuntamente se revisó las plataformas Contiki-OS y Riot-OS, que son las plataformas a evaluar en la presente investigación. Posteriormente se realizó la simulación de la red de sensores en el software OMNeT++, donde se pudo obtener una idea sobre el consumo de recurso energético que tendría la red, luego se procedió a la configuración y ensamblaje de las motas que intervienen en la red, mediante el software XCTU y el hardware Arduino. Finalmente se implementó las motas sobre puntos específicos de la estructura del puente, como las bases y el punto medio del mismo, para poder evaluar los parámetros de consumo de memoria y velocidad de procesamiento, para la evaluación del consumo energético de cada mota, se procedió a desmontar el escenario y evaluar manualmente este parámetro mediante un multímetro digital Proskit, esto debido a que no se cuenta con algún script, función o librería que pueda realizar este tipo de medición. La evaluación de estos parámetros se los realizo con dos plataformas distintas, primero se realizaron las mediciones con la plataforma Contiki-OS y después con la plataforma Riot-Os, obteniendo así un resultado de eficiencia del 55.55% para Contiki-OS y un 88.88% para Riot-OS. Se pudo determinar que la plataforma Riot es la adecuada para estos tipos de investigaciones, debido a sus bajas tasas de consumos concernientes a recursos de memoria, procesamiento y batería, recomendando así a los estudiantes para la implementación de investigaciones futuras donde necesiten enviar y recibir información en tiempo real, además de un consumo bajo de recursos.

Palabras claves:<IPV6 SOBRE REDES DE ÁREA PERSONAL INALÁMBRICA DE BAJA POTENCIA (6LOWPAN)> <PLATAFORMA (CONTIKI-OS)> <PLATAFORMA (RIOT-OS)> <SOFTWARE (OMNET++)> <HARDWARE (ARDUINO)> <TELECOMUNICACIONES Y REDES>

SUMMARY

Performance evaluation platform 6LOWPAN, applied to the design of structural monitoring of bridges. It was developed in Escuela Superior Politécnica de Chimborazo, Informatics and Electronics Faculty. It began with the review of internet concept of things and Smart City, likewise the state of the art 6LoWPAN standard to know its main features as architecture, configuration, and security. Together the Contiki-OS and Riot-OS platforms were revised, which are platforms to be evaluated in this investigation. Later simulation sensor network software OMNet++ was performed. There, we could get an idea of the consumption of energy resources that would have the network. Then it proceeded to the configuration and assembly specks involved in the network by the XCTU software and Arduino hardware. Finally specks on specific points of the bridge structure, as the foundation and the midpoint were implemented in order to evaluate the parameters of memory consumption and processing speed. For the assessment of energy consumption every speck, proceeded to dismantle the stage and evaluate this parameter manually using a digital multimeter Proskit. It occurs because they do not have a script, function or library that can perform this type of measurement. The evaluation of these parameters was conducted with two different platforms. First measurements were performed with the Contiki-OS platform and the with the Riot-OS platform, thus obtaining a result of efficiency of 55.55% for Contiki-OS and 88.88% Riot-OS. It was determined that the Riot platform is suitable for these types of investigations, due to their low rates of consumption concerning memory resources, processing and battery. It is recommending students for the implementation of future research where they need to send and receive information in real time, and low resource consumption.

Keywords: <IPV6 ON NETWORKS OF PERSONAL AREA WIRELESS LOW POWER (6LOWPAN)> <PLATFORM (CONTIKI-OS)> <PLATFORM (RIOT-OS)> <SOFTWARE (OMNET++)> <HARDWARE (ARDUINO)> <TELECOMMUNICATIONS AND NETWORKS>

INTRODUCCIÓN

Antecedentes

En la actualidad el intercambio de información entre personas o dispositivos es muy fundamental, y los sistemas embebidos poseen un papel muy importante donde el eje principal de su avance, es la interconectividad de los objetos, debido a esto surge un nuevo concepto llamado "Internet de las Cosas" o también conocida como IoT (Internet of Things), el cual describe una red formada por diferentes objetos de uso cotidiano interconectados entre sí.

El Internet de las cosas (IoT) percibe un mundo en el cual los dispositivos que lo conforman pueden ser identificados en el Internet, donde las redes sensoriales incrementan la ubicuidad de las redes, con dispositivos inteligentes de bajo costo y fácil implementación, con estándares como IEEE 802.15.4 en la capa física, 6LoWPAN en la capa de red, y RPL como protocolo de enrutamiento, que se integran en el concepto de IoT para traer nuevas experiencias en las actividades de la vida diaria. (Cama A., De la Hoz E., Cama D., 2012, p. 163)

Desarrolladores de tecnologías inalámbricas valoran las redes sensoriales como una tecnología de gran importancia para el futuro. Empresas como Texas Instruments, Libelium, Microsoft, Intel, entre otros, están sobre esta línea de investigación debido a sus diversas aplicaciones, en campos como el medio ambiente, seguridad, agricultura, industria.

Pero cada mota de una red de sensores inalámbricos trabaja bajo una determinada plataforma o sistema operativo, la cual debe contar con una característica especial para poder formar parte de una red IoT. Esta característica especial es el estándar 6LowPan que mediante su capa adaptación nos provee la implementación del protocolo IPv6 en su plataforma.

Estas plataformas también han evolucionado con el pasar del tiempo, dando mejores beneficios como menor consumo de batería, memoria, procesamiento, todo esto gracias a la unión de varias entidades las cuales desarrollan dichas plataformas operativas para poder ejecutarse en los diferentes sistemas embebidos.

Dando paso a la posibilidad de desarrollar una red de sensores inalámbricos con diferentes plataformas que se pueden encontrar en la actualidad, y principalmente basados en los estándares IEEE 802.15.4, 6LowPan y RPL, los cuales son muy fundamentales para el desarrollo de las Smart Cities así como también de las IoT.

Debido a que Ecuador es un país sujeto a varios desastres naturales tales como temblores, terremotos, deslizamientos de tierras, inundaciones y otros, debemos tener muy en claro que una infraestructura civil en buen estado es de suma importancia para responder ante emergencias que se generan durante estos eventos naturales.

A lo largo de la historia, los desastres naturales han dejado grandes secuelas de destrucción y muerte, lo cual ha permitido que la humanidad aprenda o adopte medidas para poder enfrentarlos y estar preparados. Por lo general en muchas ocasiones no se pueden predecir estos eventos, pero con una adecuada preparación se los podría enfrentar.(Román M., 2006, pp.13-14)

De esta forma, en el presente proyecto se pretende implementar dos escenarios de red 6LoWPany cada uno con plataformas diferente, logrando así conocer la plataforma que nos podría brindar mejores prestaciones en cuestión a procesamiento, consumo de memoria, consumo de batería, esto con el objetivo de construir un prototipo que permita aplicarse a un sistema de monitoreo estructural de puentes.

Formulación del Problema

¿Qué plataforma de 6LowPan es la más eficiente para el monitoreo estructural de los puentes?

Sistematización del Problema

- ¿Cuáles son los parámetros que determinan el nivel de eficiencia de la plataforma 6LowPan?
- ¿Cuál es la tasa de transferencia adecuada para la transmisión de la plataforma 6LowPan seleccionada?
- ¿Cuál es la cantidad de recursos en cuanto a procesamiento y memoria que consume cada uno de las plataformas?
- ¿Cuáles son las ventajas de utilizar una plataforma 6LowPan al monitoreo estructural de un puente?

Objetivos:

Objetivo General:

Evaluar el rendimiento de las plataformas 6LowPan aplicado al diseño de un monitoreo estructural de puentes.

Objetivos Específicos:

- Analizar el estado del arte de 6LowPan.
- Implementar una WSN basado en 6LoWPan con un simulador capaz de soportar Contiki y Riot.
- Desarrollar el sistema de monitoreo estructural de puentes mediante sensores con 6Lowpan.
- Evaluar los parámetros de procesamiento, memoria y batería, debido a que estos son los más fundamentales como lo detallan las RFC 6606 y RFC 4919 para una red 6LowPan.
- Verificar que plataforma es la más adecuada para su eficaz funcionamiento con el protocolo 6LowPan, y comenzar a implementarlo en los nuevos proyectos de IoT disponibles.

CAPITULO I

1. MARCO TEORICO

1.1 Internet de las cosas (IoT)

1.1.1 Introducción

Estamos sufriendo una transformación donde la tecnología nos invade cada día más, ya sea en nuestro trabajo, estudio, hogar, deporte, ocio, en cualquier lugar donde nos encontremos vamos a estar rodeados de tecnología que es muy útil para el desarrollo del ser humano.

Hace 20 años el internet era utilizado simplemente para buscar información, y solo podían tener acceso ciertas personas en el mundo, pero en los últimos 10 años se ha convertido en una potente herramienta tanto para búsqueda como también destinada para otros fines como transaccional, móvil y social.(Everiet A., Pastor J., 2013, p.5)

El internet de las cosas o también llamado Internet de los objetos nos cambiara completamente nuestra vida, ya que solamente con la aparición del internet todo el mundo ha sufrido una completa evolución ya sea en educación. Gobierno, comunicaciones, ciencia, negocios.

El IoT percibe un mundo en el cual todos sus dispositivos que los conforman pueden ser identificados en el internet mediante su dirección IPV6, y tienen la capacidad de interactuar e integrarse independientemente en la red con cualquier otro dispositivo.

1.1.2 Nacimiento del internet de las cosas (IOT)

Según el Grupo de Soluciones Empresariales para Internet (IBSG) de CISCO, El internet de las cosas no es más que, el momento en el cual existan más objetos o cosas que personas conectadas hacia el internet. (Evans D., 2011, p. 2)

En el año 2003 según el IBSG de Cisco, existía alrededor de 63000 millones de personas en el planeta, y un estimado de 500 millones de dispositivos conectados hacia el internet, como nos podemos dar cuenta en dicho año aun no existía el IoT (Evans D., 2011, p. 2). A mediados del año 2008 y 2009, se estima que nació el internet de las cosas, debido al crecimiento de los Smartphone y tablets que elevo el número de dispositivos conectados hacia el internet.

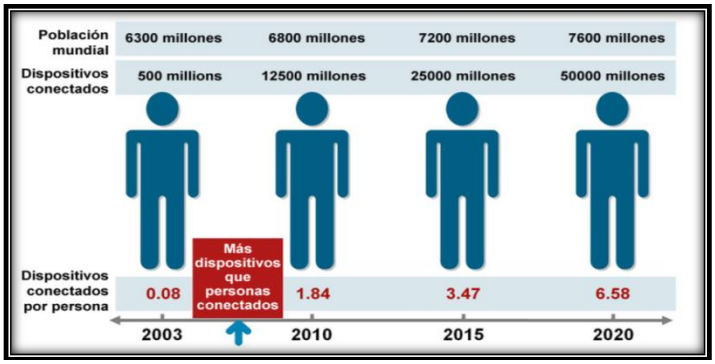


Figura 1-1. Nacimiento del internet de las cosas
Fuente: IBSG de cisco, abril del 2011

Como se puede observar en la figura 1-1, se estima que para el año 2020 exista una población mundial de 7600 millones de habitantes y unos 50000 millones de dispositivos conectados hacia el internet, teniendo en cuenta que el cálculo está hecha en la población mundial pero si lo hacemos con las personas que realmente están conectadas al internet, la cifra de dispositivos por persona se elevaría.

1.1.3 Internet de las cosas en la actualidad

Las bases de esta tecnología se encuentran en el Instituto de Tecnología de Massachusetts (MIT), en el Auto-ID Center que es un grupo fundado en el año 1999, dicho centro es el encargado de las nuevas tecnologías de detección por sensores y la identificación por radiofrecuencia (RFID).(Evans D., 2011, p. 2)

Pero a pesar del avance tecnológico aún existen obstáculos los cuales pueden frenar el avance de las IOT, algunos de ellos son la transición del IPV6, el desarrollo de fuentes de alimentación para dispositivos sensoriales.

Pero existe un punto a nuestro favor ya que se pueden encontrar empresas, gobiernos, organismos, que se encuentran trabajando de forma conjunta para que el desarrollo del Internet de las Cosas siga creciendo cada día más.

1.1.4 El IoT como una red de redes sensoriales

Actualmente el IoT está compuesto de pequeños sensores separados a una distancia razonable, como se puede ver en la Figura 2-1, tenemos varios medios de transporte los cuales poseen sensores para poder obtener control sobre su vehículo e información sobre el sistema de transporte de la ciudad.

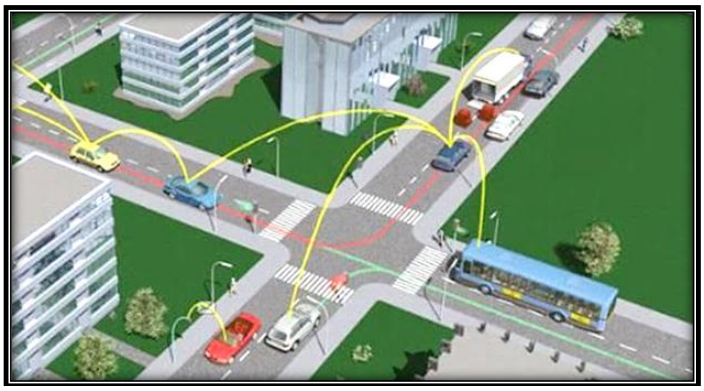


Figura 2-1. Red de sensores en el transporte
Fuente: http://www.ing.una.py/?p=13179

Mientras siga avanzando esta tecnología, podremos contar con una cantidad mayor de sensores los cuales nos permitan obtener datos de todo lo que nos rodea, y así poder guardarlos y analizarlos para tomar una decisión correcta sobre sus funcionalidades.

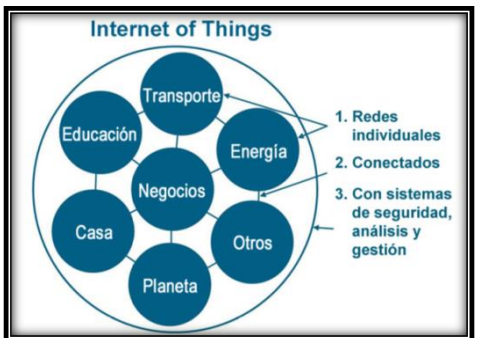


Figura 3-1. IoT, red de redes
Fuente: IBSG de cisco, abril del 2011

Como se puede observar en la figura 3-1 cada sistema podrá contar con su propio conjunto de redes individuales las cuales están interconectadas entre sí, facilitando el intercambio y almacenamiento de sus datos.

1.2 Smart Cities – Ciudades Inteligentes

1.2.1 Introducción

La Organización de Naciones Unidas (ONU) dio a conocer sus estadísticas que para el año 2050 casi el 70% de la población mundial vivirán en centros urbanos, pero esto podría llegar a ser un gran problema en el tema ambiental a menos que podamos establecer un equilibrio conjuntamente con la sociedad. (España, Asociación Española para la Calidad, 2012, p.1) (Patricia M. 2015, p.23)

Según el Instituto Nacional de Estadísticas y censos (Ecuador, INEC, 2015) el 62.77% de la población viven en zonas urbanas y el 37.23% todavía se encuentran en las zonas rurales, esto se debe a que las personas de las zonas rurales buscan oportunidades de desarrollo en las ciudades más cercanas.

Debido al crecimiento de la población en las ciudades, es necesario pensar en la construcción de proyectos que puedan prevenir cualquier desastre natural antes que la ciudadanía tenga un crecimiento mayor.

1.2.2 ¿Qué es una Smart City?

Una *SMART CITY* es la ciudad que utiliza cada avance de la tecnología para mejorar la calidad de vida de los ciudadanos, mediante la integración inteligente de nuevas herramientas o servicios como la economía, el transporte, los negocios, para que puedan ser más amigables e interactivos con los ciudadanos.

Todos los avances de una Smart City tienen presente el concepto, que siempre debe existir un equilibrio entre el consumo de los recursos naturales y el medio ambiente, ya que es un ecosistema en el cual intervienen múltiples agentes.

La Smart City se convierte en una interactiva plataforma digital la cual permite llevar al máximo nivel la economía, el bienestar ciudadano, la sociedad, además busca aprovechar al máximo los presupuestos públicos, precisamente gracias a la mejora de los procesos propios de la ciudadanía. (Fundación Telefónica, 2011.p. 20)

1.2.3 ¿Por qué son necesarias las Smart Cities?

Las Smart Cities están por convertirse en una de las herramientas más potentes en el ámbito de políticas públicas, al integrar el uso de las Tecnologías de la Información y Comunicación (TIC's) en la evolución de la ciudad, esto no solo mejorará notablemente los servicios, sino que construirá una vía sostenible para el desarrollo económico y social en las próximas décadas.

Desde el punto de vista de las municipalidades, disponer de una Smart City ayudaría a la gestión automática y eficiente de las infraestructuras urbanas, obteniendo así algunas ventajas como: la reducción de gastos por parte de la municipalidad, y por otro lado, mejoraría los servicios prestados hacia la ciudadanía.

En la tabla 1-1 se puede apreciar el porcentaje de ahorro de recursos en la provisión de servicios de una Smart city, aplicado a diferentes proyectos que son muy comunes en las ciudades inteligentes.

Tabla 1-1. Ahorros en la provisión de servicios con la Smart City

Área de aplicación	Ahorro
Riego de parques y jardines	15% del agua utilizada
Recogida de basuras	25% en requerimiento de transporte según el tipo de residuos
Gestión de tráfico	17% de emisiones de CO_2 a la atmósfera
Smart Metering	10% en el consumo de energía eléctrica. 7% en el consumo de agua particular

Realizado por: PALATE, Byron, 2015
Fuente: Smart Cities: un primer paso hacia la internet de las cosas, Fundación Telefónica, 2011

El propósito final de una Smart City es gestionar eficientemente todas las áreas que posee la ciudad como pueden ser: seguridad, energía, servicios, educación, salubridad, infraestructuras, transporte, complaciendo así las necesidades de sus ciudadanos (Enerlis, 2012, p.5).

1.2.4 Servicios que puede ofrecer una Smart city

Los principales ejes en los que se basa un proyecto de Smart City a menudo tienen que ver con el transporte urbano, la producción energética, la administración de las infraestructuras de la ciudad, la participación del gobierno, la seguridad de los ciudadanos, la salud, educación y cultura.

Transporte urbano.-El transporte en la ciudad es un problema que se presenta cada día, esto hace que sea una de las principales necesidades y uno de los más implementados en las Smart City, y la cual debe poseer sostenibilidad, seguridad y disponibilidad de los sistemas de transporte.

Producción energética.- La administración de la energía se está volviendo un tema muy importante en nuestro medio debido el uso de energías renovables como: fotovoltaica, geotermal, eólica, biomasa, se hace fundamental para lograr los objetivos marcados, haciendo un uso cada vez más eficiente de la energía.

Administración de las infraestructuras de la ciudad.-Las piezas más básicas de las ciudades son los edificios, ya que poseen un consumo del 40 % de toda la energía mundial,. Ante este consumo desbordante es razonable aplicar la tecnología para mejorar la administración para que sea uno de los ámbitos más destacados de las Smart Cities

Participación del gobierno.- La participación del gobierno en el desarrollo de las Smart Cities es de vital importancia ya que son las principales entidades responsables en tomar decisiones sobre el progreso de la ciudad, pero hoy en día se están relacionando más con la ciudadanía para hacer una participación transparente.

Seguridad de los ciudadanos.- Debido al crecimiento de la ciudad el tema de la seguridad de los ciudadanos se hace cada vez más fuerte, ya que para cubrir toda una ciudad con un sistema de seguridad como puede ser un circuito cerrado de televisión, conllevaría a implementar gran cantidad de recursos.

Salud.-Todas las autoridades deberían establecer una atención primaria a la salud y por lo tanto encontrar en la tecnología un aliado para ofrecer servicios médicos de calidad en el ámbito de la Smart City, y más aún cuando estamos en un entorno en el cual la población está envejeciendo y se incrementan todo tipo de enfermedades.

Educación y cultura.- Estos son unos de los servicios públicos de excelencia ya que toda tecnología es adaptable a la educación y cultura, y así poder mejorar la educación en todos sus niveles ya sea mediante plataformas virtuales u otros medios, a través de los cuales también pueden impulsar la cultura del país.

1.3 Estado del Arte IPV6 sobre redes de área personal inalámbrica de baja potencia 6LowPAN

1.3.1 Introducción

6LowPan es el acrónimo de "IPv6 over Low power Wireless Personal Area Networks", es un estándar de capa adaptación que permiten que paquetes de IPv6 sean llevadas eficientemente dentro de cabeceras pequeñas de la capa de enlace, de manera muy similar a los paquetes definidos por IEEE802.15.4, facilitando así la interacción con todas las cosas a través del uso diario de la Internet y su casi 3 mil millones de usuarios. (Shelby Z., y Bormsnn C., 2011)

Este protocolo principalmente fue ideado para que, hasta los dispositivos más pequeños del mundo puedan ser partícipes del IPV6, principalmente para aquellos dispositivos que poseen la característica de consumir baja potencia formen parte del Internet de las cosas. (Montenegro G., Hui J., Culler D., 2007,)

6LoWPAN es un padrón abierto definido por el Grupo de Trabajo de Ingeniería de Internet (IETF-Internet Engineering Task Force), que también es el organismo de normalización que define muchos de los estándares abiertos utilizados en Internet como UDP, el protocolo TCP y el HTTP.

1.3.2 Arquitectura de red 6LowPan

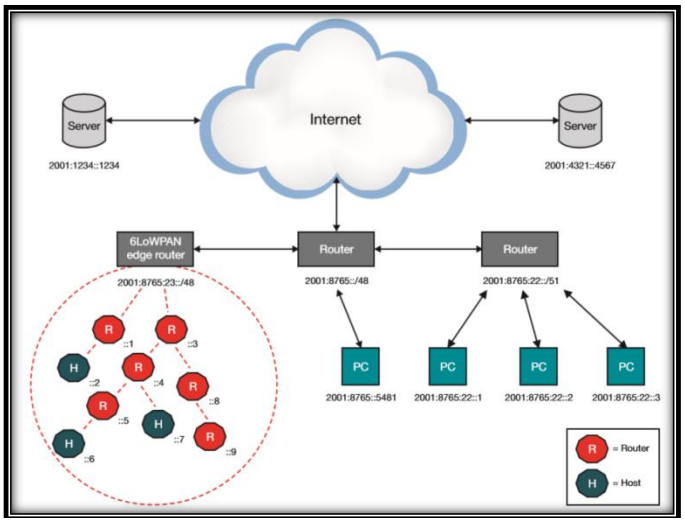


Figura4-1. Red IPv6 con una red de malla 6LoWPAN

Fuente: 6LoWPAN demystified, Texas Instrument**.**

En la Figura 4.1se muestra un ejemplo de una red IPv6 incluyendo una red de malla 6LoWPAN. El enlace hacia el Internet se lo realiza mediante un punto de acceso (AP) como un enrutador IPv6. Varios dispositivos están conectados al AP en una configuración típica, tales como PCs, servidores.

Los dispositivos que conforman una red 6LoWPAN típica son: routers y motas. Los routers reenvía el paquete destinado a otra mota de la red 6LoWPAN. Las motas pueden ser dispositivos con sueño, es decir despertarse periódicamente para comprobar su padre al que está asociado (router/coordinador), lo que permite un consumo muy bajo de energía

Las redes 6LoWPAN típicamente operar en el borde, en calidad de redes stub, esto significa que los datos que entran en la red están destinados para uno de los dispositivos dentro de la red 6LoWPAN. Una red 6LoWPAN puede estar conectada a otras redes IP a través de uno o más routers de frontera que reenvían los datagramas IP hacia los diferentes medios de comunicación.

Los routers de borde también pueden apoyar los mecanismos de transición IPv6 para conectar redes 6LoWPAN a las redes IPv4, como NAT64 definidos en el RFC 6146 (Bagnulo M., Matthews P., Van Beijnum I, 2011). Estos mecanismos de transición IPv6 no requieren los nodos 6LoWPAN ya que todo el proceso de traducción de direcciones lo realizara el router de borde.

1.3.3 Configuraciones de 6LowPan

Una red 6LoWPAN puede estar configurada de tres maneras, eso depende de la planificación estructural de la red que se lo realiza al principio.

Simple LowPAN.- Esta configuración permite a la red 6LoWPAN estar conformada por varias motas y un solo router de borde o Gateway, el cual será el encargado de redirigir el tráfico de la red.

Extended LowPAN.-Esta configuración permite a la red 6LoWPAN estar conformada por varias motas y varios Gateway, pero todos estos Gateway se encuentran compartiendo un enlace en común que es la backbone.

Ad-hoc LowPAN.- Esta configuración permite a la red 6LowPan no poseer Gateway, es decir que las motas deberán comunicarse solamente entre ellas.

1.3.4 Información general de la Pila del Sistema

6LoWPAN cambia radicalmente el punto de vista del IoT, debido a que dispositivos como ZigBee, Bluetooth y algunos sistemas propietarios necesitaban de hardware adicional para que puedan conectarse a Internet. 6LoWPAN resuelve este dilema mediante la introducción de una capa de adaptación entre el enlace de datos y la capa de red, para permitir la transmisión de datagramas IPv6 sobre IEEE 802.15.4.

En la figura 5-1 se puede observar la diferencia entre las estructuras de la pila del modelo OSI, la pila de la red WiFi y la pila 6LowPan la cual cuenta con su capa de adaptación.

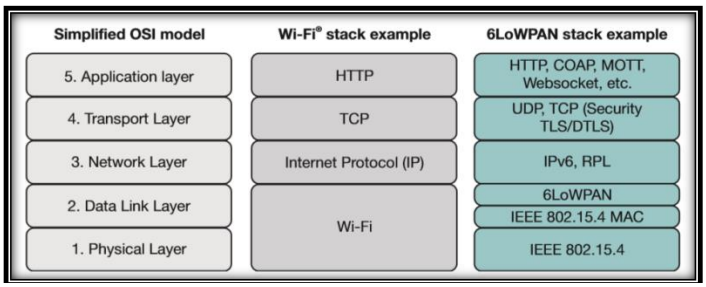


Figura 5-1. El modelo OSI, pila Wi-Fi y la pila 6LowPan
Fuente: 6LoWPAN demystified, Texas Instrument.

1.3.4.1 La capa física

Convierte bits de datos en señales que se transmiten y se reciben a través del aire. En el ejemplo 6LoWPAN, se utiliza IEEE 802.15.4, la cual es responsable de dicha conversión para que las motas puedan comunicarse con las demás motas de su red o con motas pertenecientes a redes externas.

1.3.4.2 La capa de enlace de datos

Proporciona un enlace fiable entre dos nodos directamente conectados mediante la detección y corrección de errores que pueden ocurrir en la capa física durante la transmisión y recepción.

La capa de enlace de datos incluye la capa de acceso al medio (MAC), que proporciona acceso a los medios de comunicación, el uso de características como Acceso múltiple con detección de portadora y evasión de colisiones (CSMA -CA), donde la radio escucha que nadie más está transmitiendo antes de enviar datos a través de aire. Esta capa también se encarga de la elaboración de datos.

En el ejemplo 6LoWPAN, la capa MAC es IEEE 802.15.4. La capa de adaptación 6LoWPAN, proporcionando la adaptación de IPv6 IEEE 802.15.4, también reside en la capa de enlace.

1.3.4.3 La capa de red

Es muy esencial ya que nos provee el direccionamiento durante varios saltos en la red. La IP (Internet Protocol) es el protocolo de red utilizado para proporcionar a todos los dispositivos una dirección IP para el transporte de paquetes desde un dispositivo a otro.

1.3.4.4 La capa de transporte

Genera sesiones de comunicación entre aplicaciones que se ejecutan en los dispositivos finales, permite múltiples aplicaciones en cada dispositivo y tiene su propio canal de comunicaciones. TCP es el protocolo de transporte dominante en el Internet.

Sin embargo, TCP es un protocolo basado en la conexión (incluyendo ordenamiento de paquetes) con gran sobrecarga y por lo tanto no siempre adecuado para dispositivos que exigen bajo consumo de energía. Para esos tipos de sistemas es mejor UDP ya que posee una sobrecarga menor y es un protocolo sin conexión.

1.3.4.5 La capa de aplicación

Es responsable de formateo de datos, también se asegura de transportar los datos en esquemas de aplicaciones óptimas. La capa de aplicación más utilizada en Internet es HTTP que se ejecuta a través de TCP. HTTP utiliza XML, que es un lenguaje basado en texto con una gran sobrecarga. Por lo tanto, no es óptimo para utilizar HTTP en muchos sistemas 6LoWPAN.

Por esta razón, la industria y la comunidad han desarrollado protocolos alternativos para la capa de aplicación, tales como el protocolo de aplicación restringida (COAP) y el de transporte de telemetría de cola de mensajes (MQTT).

1.3.4.5.1 Protocolo de aplicación restringida (COAP)

Un protocolo de mensaje que se ejecuta sobre UDP con un bit-optimizado REST mecanismo muy similar a HTTP.COAP es definido por IETF RFC 7252 (Shelby Z., Hartke K., Bormann C., 2014) y define retransmisiones, mensajes confirmables y no confirmables, soporte para dispositivos con sueño, transferencias de bloques, el apoyo de suscripción y descubrimiento de recursos.

1.3.4.5.2 Transporte de telemetría de cola de mensaje (MQTT),

Un protocolo de código abierto que fue inventado por IBM y se ejecuta a través de TCP. Los datos no se transportan directamente entre los puntos finales en su lugar se utiliza un servidor para retransmitir los mensajes. MQTT introduce la entidad "tema", los dispositivos pueden publicar y suscribirse a diferentes temas.

Una vez que un tema se actualiza, el dispositivo recibirá una notificación y recibirá los datos a través del servidor. Los dispositivos pueden utilizar comodines como # y * para suscribirse a una jerarquía de temas. MQTT soporta varias capas de calidad de servicio (QoS) para asegurarse de que los mensajes se entreguen.

Hay muchos protocolos de capa de aplicación más disponibles que pueden correr sobre el TCP o UDP, los mencionados anteriormente se dirigen específicamente a baja potencia y a las aplicaciones de la IOT.

1.3.5 Protocolo de Internet versión 6 (IPv6) a través de IEEE 802.15.4

El internet de hoy en día se basa principalmente en IPv4 y utiliza direcciones de 32 bits, lo que limita el espacio de 4294967296 (2^{32}) direcciones únicas. El número de direcciones disminuye de forma natural a medida que se asignan direcciones IP a diferentes usuarios o dispositivos.

Según la IANA (Agencia de Asignación de Números de Internet), el agotamiento de direcciones IPv4 disponibles en su reservase produjo el 3 de febrero de 2011 (Sánchez G. 2012, p.19), a pesar de que se había retrasado considerablemente por los cambios de dirección, como la traducción de direcciones de red (NAT).

La limitación del IPv4 estimuló el desarrollo de IPv6 en la década de 1990, que ha estado en el despliegue comercial desde 2006. IPv6 cubre un espacio de2^{128}direcciones IP únicas. Se espera que esto sea suficiente para el crecimiento del Internet en las próximas décadas, incluso con las aplicaciones del Internet de las cosas que según estimaciones podría incluir 50 mil millones de dispositivos conectados para el año 2020(Evans D., 2011).

Para reconocer el aumento de ancho de banda, IPv6 incrementa la unidad máxima de transmisión (MTU) de 576 a 1280 bytes. IPv6 también refleja los cambios y los avances en las tecnologías de la capa de enlace utilizada en Internet.

Por otro lado, IEEE 802.15.4 fue diseñado para servir a un mercado diferente, para las aplicaciones de larga vida útil y que requieren un bajo costo, dispositivos de potencia ultra baja. La tasa de transferencia según este estándar se limita a 250 kbps, y la longitud de trama está limitada a 127 bytes para asegurar bajas tasas de paquetes y de error de bits en un entorno de Radio Frecuencia con pérdida.

El estándar IEEE 802.15.4 utiliza dos direccionamientos: una pequeña dirección de 16 bits y una dirección extendida EUI-64. Estas direcciones reducen la sobrecarga del encabezado y minimizan los requisitos de memoria.

Por último, los dispositivos utilizados para implementar 6LoWPAN suelen ser limitados en términos de recursos, que tiene alrededor de 16 kB de memoria RAM y 128 kB ROM.

La incorporación el protocolo IPv6 en el estándar 802.15.4, se logró mediante la agregación de la capa adaptación y la compresión de cabeceras para poder incorporar la dirección IPv6.

1.3.6 La capa de adaptación 6LoWPAN

Cuando se envía datos a través de las capas MAC y PHY, siempre se utiliza una capa de adaptación. Por ejemplo, RFC 2464 (Crawford M. 1998), define cómo se encapsula un paquete IPv6 en una trama Ethernet, lo mismo también se utiliza para el IEEE 802.11 Wi-Fi. Para 6LoWPAN, RFC 6282 (Hui J., Ed., Thubert P., 2011) define cómo se encapsula una trama de datos IPv6 sobre un enlace de radio IEEE 802.15.4.

1.3.6.1 La compresión de cabecera

El formato 6LoWPAN establecido en la RFC 4944 (Montenegro G., Hui J., Culler D, 2007), trabaja en conjunto con dos sistemas de compresión de cabecera: HC1 que es el encargado de comprimir cabeceras IPv6 y el HC2 que se encarga de la compresión de las cabeceras UDP.

La compresión consiste en reducir el encabezado de 40 octetos de IPv6 con la finalidad de ocupar un espacio muy pequeño de la carga útil de la trama IEEE 802.15.4. Para la compresión se incluyó un campo denominado dispatch (DSP), que es un campo de control compuesto de un conjunto de bits codificado que indican la coexistencia con otras redes, el tipo de compresión y las secuencias de fragmentación.

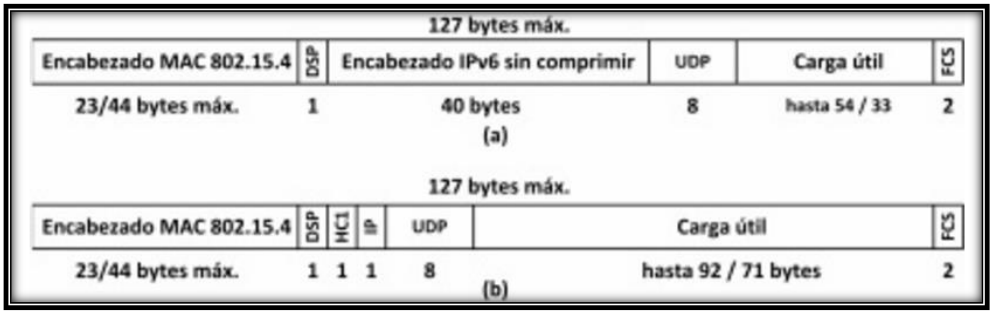


Figura 6-1. Compresión de cabecera 6LowPan
Fuente:http://www.academia.edu/7842663/0_0_0_0_Internet_of_Things_y_est%C3%A1ndar_6LowPan

En la figura6-1, se puede apreciar las tramas del formato IEEE 802.15.4 con el encabezado IPv6 sin compresión (a) y posterior con el encabezado ya comprimido (b).

En la trama donde hay compresión, el octeto HC1 es el resultado de la compresión realizada. La información redundante que se encuentra en el encabezado IEEE 802.15.4, o que se puede manejar desde el Gateway de la PAN, se elimina antes de la compresión por lo que en principio cuarenta octetos se comprimen en un solo octeto simbolizado por HC1. (Espinosa C. 2015)

Cualquier campo del encabezado IPv6 que no se ha comprimido se agrega posteriormente al campo IP formándose el IP hop limit. Adicionalmente un encabezado UDP de 8 octetos debe ser comprimido a 3 octetos y la longitud del payload de la trama IEEE 802.15.4 se amplía a 5 octetos. Un Edge Router debe tener la capacidad de comprimir el encabezado IPv6 y de presentar al internet a los nodos de su red PAN.

La compresión del encabezado IPv6 considera la dirección única de host mientras el edge router guarda el prefijo correspondiente de la dirección de red IPv6 y su respectiva mascara común para todos los nodos de la LowPAN.

En la figura 7-1 se presenta como es la comunicación entre dos host que poseen una dirección IPv6, en dicha comunicación los routers deben ser capaces de analizar las direcciones de destino de cada paquete para poder reenviarlos hacia su destino.

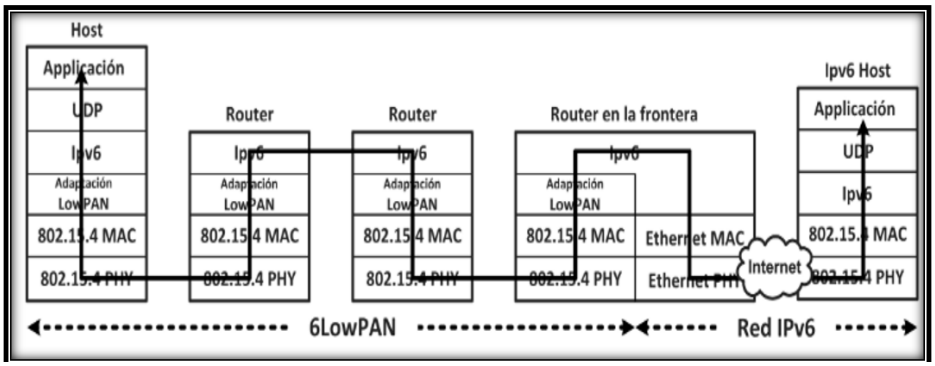


Figura 7-1. Comunicación 6LowPan
Fuente:http://www.academia.edu/7842663/0_0_0_0_Internet_of_Things_y_est%C3%A1ndar_6LowPan

1.3.6.2 Fragmentación y reensamblado

Otro problema de IPv6 son sus MTUs de 1280 octetos por lo que se debe también realizar una fragmentación de paquetes IP en múltiples tramas de la capa de enlace para acomodarse al mínimo MTU de IPv6 requerido y de esta manera poderlos re-ensamblar en el otro extremo.
Para permitir la transmisión de tramas IPv6 a través de enlaces de radio IEEE 802.15.4 las tramas IPv6 necesitan ser divididas en varios segmentos más pequeños (Fragmentos). Para este propósito, los datos adicionales en los encabezados se generan para reensamblar los paquetes en la secuencia correcta en el extremo.

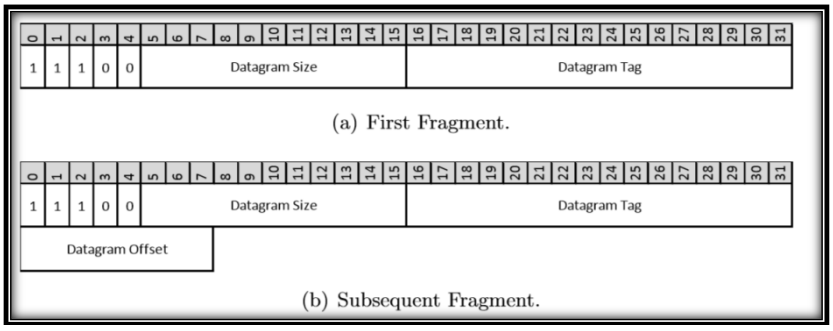


Figura 8-1. Fragmentación de cabecera 6LowPan:
Fuente:http://disco.ethz.ch/theses/fs09/Lars_Schor_IPv6SensorNodes.pdf

Como se aprecia en la figura 8-1, el primer campo es el DatagramSize, mismo que representa el tamaño de todo el paquete IP antes de la fragmentación y sirve para identificar el paquete cuando éste llega en desorden.

El DatagramTag es el siguiente campo que se encarga de identificar al payload fragmentado y finalmente el campo Datagram Offset define el desplazamiento del fragmento dentro del payload original.

Cuando los paquetes de datos son re-ensamblados, se elimina la información adicional añadido en los encabezados y los paquetes se restauran a su formato inicial IPv6. La secuencia de fragmentación es diferente en función al tipo de enrutamiento que se utiliza.

En el caso de una malla con enrutamiento, los segmentos se vuelven a reensamblar solamente en su destino final, mientras que los paquetes en redes con enrutadores se vuelven a reensamblar en cada salto. Es así que en una red de enrutador sobre cada salto debe tener los recursos suficientes para almacenar todos los fragmentos.

Si es posible, la fragmentación debe evitarse siempre que sea posible, ya que impacta negativamente en la vida de la batería de un dispositivo. Por lo tanto, mantener una baja carga (incluye la selección de los protocolos apropiados a nivel de aplicación) y el uso de la compresión de cabecera son de la máxima importancia.

1.3.6.3 Direccionamiento de cabecera

La cabecera adicional "MeshAddressing" se utiliza para soportar el reenvió de capa dos y el reenvió multi-hop de payloads 6LoWPAN.

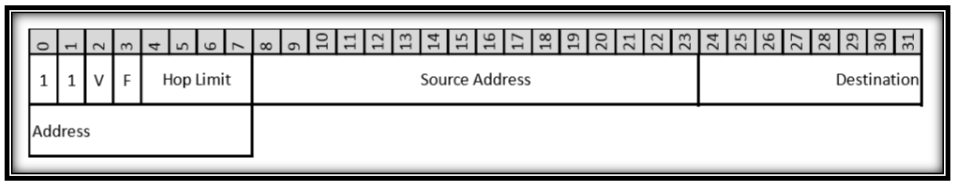


Figura 9-1. Cabecera MeshAddressing
Fuente:http://disco.ethz.ch/theses/fs09/Lars_Schor_IPv6SensorNodes.pdf

En la figura 9-1 se muestra la cabecera MeshAddresing, donde el bit V indica si la dirección de origen es una dirección corta es decir de 16 bits o una larga de 64 bits, El bit F indica lo mismo pero para la dirección de destino final. El campo Hop Limit es similar a la compresión de cabecera. La Dirección de Origen contiene la dirección del creador del paquete 6LoWPAN y el campo Destino corresponde a la dirección de destino final.

1.3.7 Enrutamiento

El enrutamiento es la capacidad de enviar un paquete de datos de un dispositivo a otro dispositivo, a veces a través de múltiples saltos. Dependiendo en qué capa se encuentre el mecanismo de enrutamiento se definen dos mecanismos: bajo malla o sobre rutas.
En la figura 10-1, se representa gráficamente como es el enrutamiento, bajo malla y sobre rutas, que funciona sobre la capa dos y sobre capa tres respectivamente.

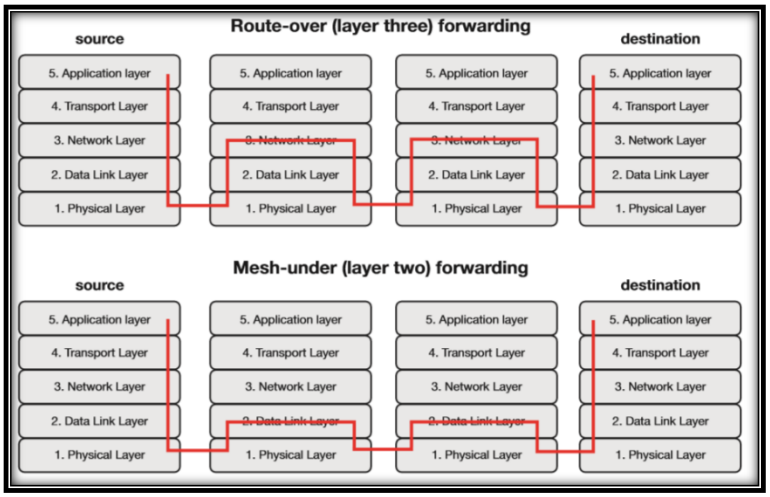


Figura 10-1. Pila bajo-malla y pila sobre-ruta para el reenvío de paquetes
Fuente: 6LoWPAN demystified, Texas Instrument**.**

En redes bajo malla.- el enrutamiento de datos sucede de forma transparente, por lo tanto se consideran las redes para ser una subred IP bajo malla. El único enrutador IP en un sistema de este tipo es el enrutador de borde.

Las redes bajo malla son las más adecuadas para redes más pequeñas y locales.

En las redes sobre rutas.- el encaminamiento se lleva a cabo en el nivel IP, por lo que cada salto en tales redes representa un router IP. El uso de enrutamiento IP proporciona la base para las redes más grandes, más potentes y escalables. El protocolo de enrutamiento más utilizado para redes 6LoWPAN sobre rutas es RPL según lo definido por IETF en RFC 6550. (Winter t., Ed., Thubert P., et al., 2012)

1.3.7.1 Protocolo de Enrutamiento RPL

La RFC 6550 (Winter t., Ed., Thubert P., et al., 2012),especifica el protocolo de enrutamiento IPv6 para bajo consumo de energía y las redes con pérdida (RPL- IPv6 Routing Protocol for Low-Power and Lossy Networks), que proporciona un mecanismo que permita soportar el trafico multipunto-punto de los dispositivos dentro de la red 6LoWPAN hacia un punto central de control (servidor) así como el tráfico punto-multipunto desde el punto central de control a los dispositivos dentro de la red 6LoPWAN.

El soporte para el tráfico punto a punto también está disponible, sin embargo, RPL no es la elección óptima para este tipo de tráfico, ya que los datos en muchos casos necesitan ser transportados a través del enrutador de borde. RPL es compatible con dos modos de enrutamiento diferentes; modo de almacenamiento y el modo de no-almacenamiento.

En el modo de almacenamiento, todos los dispositivos de la red 6LoWPAN configurados como routers mantienen una tabla de enrutamiento y una tabla vecina. La tabla de enrutamiento se utiliza para buscar rutas a dispositivos, y la tabla vecina se utiliza para realizar un seguimiento de los vecinos directos de un nodo.

En modo de no-almacenamiento, el único dispositivo con una tabla de enrutamiento es el router de borde, por lo tanto, se utiliza el enrutamiento de origen. Enrutamiento de origen significa que el paquete incluye la ruta completa que tiene que tomar para llegar al destino.

Por ejemplo , al enviar datos de un dispositivo a otro dentro de la misma red 6LoWPAN , los datos se envían primero desde el dispositivo de origen al router de borde, el router de borde a su vez hace una búsqueda en su tabla de enrutamiento y agrega la ruta completa a la destino en el paquete.

1.3.7.2 Protocolo de descubrimiento de vecinos (NDP) o (ND)

Para que un host pueda configurar su direccionamiento automáticamente, puede comunicarse a través de protocolo de descubrimiento de vecinos (NDP- Neighbor Discovery Protocol), sin embargo muchas de las características del NDP también se incluyen en RPL, y consiste en cuatro tipos de mensajes:

- Solicitud de enrutador (RS)
- Anuncio de enrutador (RA)
- Solicitud de vecino (NS)
- Anuncio del vecino (NA)

Descubrimiento de vecinos de IPv6 (ND) permite a un dispositivo descubrir vecinos, mantener la información de accesibilidad, configurar rutas predeterminadas, y propagar los parámetros de configuración.

El mensaje RS incluye, entre otras cosas, el prefijo IPv6 de la red. Todos los routers de la red envían periódicamente estos mensajes.
Si un host quiere participar en una red 6LoWPAN, se asigna a sí mismo una dirección unicast de link-local (FE80::IID), a continuación envía esta dirección en un mensaje de **NS** a todos los demás participantes en la subred para comprobar si se está utilizando la dirección por otra persona. Si no oye un mensaje de **NA** en un plazo definido, se asume que la dirección es única.

Este procedimiento se llama detección de direcciones duplicadas (DAD). Ahora, para obtener el prefijo de red, el host envía un mensaje de **RS** al router para obtener el prefijo correcto. Con el uso de estos cuatro mensajes, un host es capaz de asignarse una dirección IPv6 única en el mundo.

En el uso de configuración automática de dirección de origen, cada host genera una dirección IPv6 de enlace local utilizando su dirección de IEEE 802.15.4 EUI-64, dirección corta de 16 bits o ambos.

En una configuración bajo-malla, el alcance del link-local cubre toda la red 6LoWPAN, incluso a través de múltiples saltos. La única vez que se necesita una dirección IPv6 enrutable es cuando la comunicación esta fuera de la red 6LoWPAN.

En una configuración sobre-ruta, una dirección de link-local es suficiente para comunicarse con los nodos que están dentro de la cobertura de radio, pero se requiere una dirección enrutable para comunicarse con dispositivos de varios saltos de distancia. Para todas las direcciones unicast, es más eficiente obtenerlos de la dirección local IEEE EUI-64.

1.3.8 Seguridad

La seguridad es una necesidad para los sistemas de IoT y por lo tanto en 6LowPan, siempre presenta un desafío. Debido a la naturaleza de los nodos que tienen un rendimiento muy limitado, también hay más puntos de entrada para un atacante exterior.

Otro aspecto fundamental es que los datos que fluyen en un sistema típico de la IoT no sólo son "datos", el daño potencial es mucho mayor, ya que los datos que fluyen en el sistema se puede utilizar para abrir la puerta, la casa, o activar / desactivar las alarmas de forma remota.

6LoWPAN aprovecha la fuerte seguridad AES-128 de capa de enlace, definida en IEEE 802.15.4. La seguridad de la capa de enlace proporciona autenticación y cifrado, además de la seguridad de capa de enlace, se ha demostrado la seguridad de capa de transporte (TLS-Transport Layer Security) mecanismos para un buen trabajo en los sistemas 6LoWPAN.

1.3.9 Plataformas que funcionan sobre 6lowpan

Hoy en día existen diferentes plataformas de código abierto para 6lowpan disponibles, pero en nuestro medio la plataforma que más ha sobresalido es Contiki-OS, gracias a la incorporación de IPv6 en las WSN además es de fácil acceso, pero con la evolución de la tecnología surge una nueva plataforma la cual se denomina Riot-OS, dicha plataforma apuesta por ser quien abarque todo el mercado de las WSN e IoT.

1.4 Plataforma Contiki-OS

Figura 11-1. Plataforma Contiki
Fuente: http://electronicsofthings.com

Contiki es una plataforma o también llamado sistema operativo de código abierto desarrollado para algunos ambientes reducidos, el cual proporciona portabilidad, carga dinámica, descarga de procesos y servicios individuales. Desarrollado por el Swedish Institute of Computer Science,

liderado por Adam Dunkels, provee de un sistema de comunicación IP, tanto para IPv4 y para IPv6.

Permite que las aplicaciones se comuniquen directamente con el hardware haciendo que su tiempo de respuesta sea mucho más rápido. Las principales características que posee esta plataforma son:

- Kernel multitarea
- Conectividad TCP/IP
- Un navegador Web
- Cliente telnet
- Interfaz Gráfica de Usuario
- Salvapantallas

1.4.1 Estructura de Contiki-OS

El núcleo es por eventos, pero el sistema soporta varios hilos de ejecución que se encuentran listos solamente esperando por algún evento que los activen. Los hilos de ejecución son implementados como una biblioteca que está vinculado únicamente con programas que requieren explícitamente multi-procesos.

Contiki es implementado en el lenguaje C y ha sido adoptado por una serie de arquitecturas de microcontroladores, posee un condigo que pesa uno cuantos Kilobytes y requiere de unos cientos de bytes de memoria RAM para que pueda funcionar.
El Stack de protocolos de Contiki utiliza la capa física del estándar IEEE 802.15.4, por lo que su trabajo se centra en las capas superiores, Capa MAC y Capa de Red

1.4.2 Arquitectura de Contiki-OS

El funcionamiento de la plataforma de Contiki consiste en su Kernel, las bibliotecas, el cargador del programa, y un conjunto de procesos.

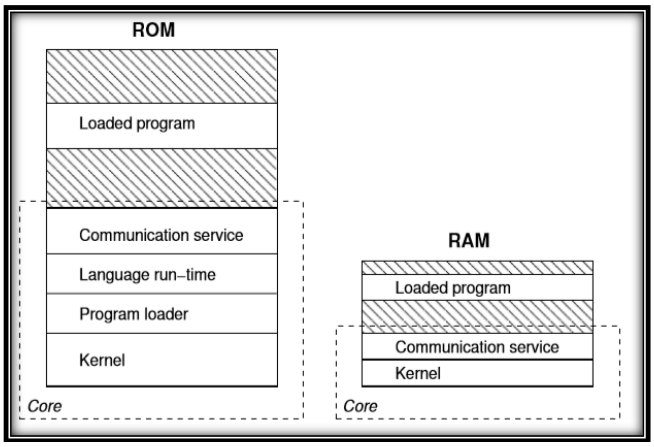


Figura 12-1. Arquitectura de Contiki
Fuente: Contiki-a lightweight and flexible operatingsystemfortinynetworkedsensors

Como se muestra en la Figura 12-1, la plataforma Contiki se divide en dos partes: el núcleo y los programas cargados, la partición se realiza en tiempo de compilación, por lo general, el núcleo está formado por el núcleo Contiki, el cargador del programa, las bibliotecas más utilizadas de tiempo de ejecución de lenguaje y de apoyo, y una pila de comunicación con los controladores de dispositivo para el hardware de comunicación. (Dunkels A., Grönvall B., &Voigt T, 2004)

El núcleo se compila en una sola imagen binaria que se almacena en los dispositivos antes del despliegue. Los programas son cargados en el sistema por el cargador de programa. El cargador de programa puede obtener los binarios del programa, ya sea mediante el uso de la pila de comunicación o mediante el uso de almacenamiento conectado directamente como EEPROM.

Por lo general, los programas que se cargan primero son almacenados en la EEPROM antes de que estén presentes en el código de memoria.

1.4.2.1 Procesos.

Son tareas específicas que se ejecutan muy seguidas y pueden ser iniciados tanto por los usuarios como por el propio programa. Cada proceso posee un protothread el cual es ejecutado al momento de iniciarse el proceso, los procesos pueden pasar por 3 etapas: desactivado, activado y llamado. Un proceso esta desactivado cuando ningún evento puede acceder a él. Esta activado cuando puede ser ejecutado por cualquier evento y esta llamado cuando se está ejecutando su protothread.(Cujilema G. y Cherres P., 2015, p.75)

1.4.2.2 Eventos.

Los eventos son acontecimientos con los cuales se debe ejecutar un proceso, cada evento posee asociado un proceso al cual puede o no enviarle información adicional para que éste lo utilice. Los eventos son almacenados en memoria y son ejecutados en el orden que sean invocados. Los eventos pueden ser llamados desde los protothread o desde cualquier lugar del programa.

1.4.2.3 Protothreads.

Son funciones opcionales que deben aguardar por algún evento para ser activado, se encuentran funcionando bajo el programa principal, esto ayuda a que el dispositivo pueda realizar otras actividades hasta que se presente el evento que necesite al proceso asociado al protothread, por ende un proceso solo está asociado a un solo protothread (Cujilema G. y Cherres P., 2015, p.75).

1.4.2.4 Poll y Post Sincrónico.

Existen casos en donde algunos procesos requieren ejecutarse inmediatamente sin tener que esperar a que se terminen todos los procesos guardados en la cola. Para lograr esto, surgen los siguientes métodos:

- **El post sincrónico.-** ejecuta un evento sin necesidad de estar en cola, interrumpe la ejecución del proceso activo para ejecutar el proceso marcado con post.

- **El poll.-** asigna una bandera a los procesos prioritarios, antes de ejecutar algún proceso se revisa esta bandera, se verifica si algún proceso posee el poll, y si es asíse ejecuta primero este proceso. El post espera a que se termine de ejecutar el proceso activo para luego ejecutar el proceso con el poll activo.

1.4.2.5 Timers

Los timers son los "relojes" del sistema operativo, y se utilizan para temporizar o ejecutar de manera periódica ciertos procesos. Existen varios timers dentro de Contiki: (Cujilema G. y Cherres P., 2015, p.75)

- **Etimer:** es un conjunto de timers que se ejecutan de manera encadenada, cada timer guarda el tiempo de: inicio y expiración, además un puntero que señala: el proceso a ejecutarse al expirar el timer y el próximo timer de la lista.
- **Ctimers (Callback timers):** Utilizado cuando se necesita enviar un parámetro a una función cuando el timer expire, cuando el ctimer es reseteado se elige un nuevo intervalo de tiempo, la función y el parámetro a enviar. Emplea a los etimers para que le notifiquen que el timer ha expirado.
- **Rtimers:** Encargado de ejecutar tareas en tiempo real y funciones en tiempos exactos, su funcionamiento depende del oscilador de 32 kHz que posee la mota y se lo emplea cuando se requiere medir intervalos de tiempo con precisión.

1.4.3 Capa de red en Contiki

Contiki implementa el módulo uIP (micro IP) en su capa de red, pero se han suprimido ciertas funciones debido a las restricciones tanto de hardware como energético que poseen las motas, a pesar de esto se cuenta con un Stack totalmente operativo y funcional.

El módulo de uIP fue desarrollado para trabajar en dispositivos que poseen poca memoria y un hardware limitado como los nodos o motas de una red WSN, debido a esto Contiki adopto este módulo para formar parte de su sistema.

1.4.3.1 Ipv6 en Contiki

Para poder utilizar el Stack 6LoWPAN en Contiki, se crearon los archivos sicslowpan.h, sicslowpan.c y StackIpv6.h, estos archivos se los realizaron con los borradores realizados por el grupo de trabajo de la IETF "IPv6 over Low power WPAN (6LoWPAN)".

Contiki utiliza el protocolo de enrutamiento RPL que fue desarrollado por el grupo IEEE, como ya se describió en la sección anterior.

1.5 Plataforma Riot–OS

Figura 13-1. Riot-OS
Fuente: http://www.Riot-os.org/

Riot-OS es un sistema operativo de código abierto basado en microkernel, coincidiendo con varios requerimientos de software para dispositivos IoT, además cuenta con la implementación de una pila de red adaptativa, proporcionando un completo IPv6, así como los protocolos de redes más restringidos como, 6LoWPAN o RPL.

Al proporcionar una API amistosa para el desarrollador y al mismo tiempo proporcionar características claves como capacidades en tiempo real y eficiencia de energía, RIOT puede alimentar un amplio espectro de dispositivos IoT. RIOT por lo tanto se puede aprovechar para evitar los costes de desarrollo de código y mantenimiento redundantes para aplicaciones de la IoT. (Baccelli E., et al., 2012)

1.5.1 Arquitectura de Riot-OS

La arquitectura de microkernel escrito en ANSI C y el soporte para los multi-threading permite el desarrollo amigable de una API.RIOT está completamente escrito en C, pero también permite el uso de C ++ y el uso de compiladores GNU (GCC) en la última versión, y fue diseñado para:

- Eficiencia energética
- Desarrollo independiente del hardware
- Alto grado de modularidad

1.5.2 Estructura de Riot-OS

RIOT OS apunta exactamente a satisfacer la totalidad de las restricciones de hardware heterogéneo, para llenar los vacíos que existe entre los sistemas operativos para WSANs y las plataformas que corren actualmente en los host de internet.

En la figura 14-1, se puede apreciar la estructura de la plataforma Riot, además de los protocolos utilizados y el tipo de soporte (completo o parcial) que ofrece Riot para cada nivel de sus capas.

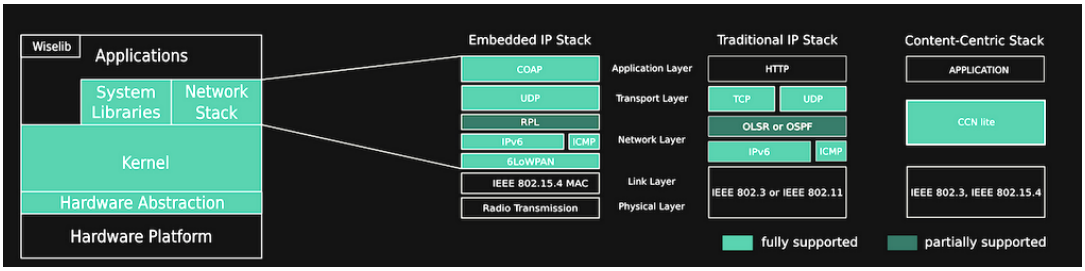


Figura 14-1. Estructura de Riot
Fuente: https://www.Riot-os.org/images/RIOT_network_architecture_dark_updated.png

1.5.2.1 Modularidad

El sistema está diseñado modularmente, para lograr un uso mínimo de memoria, así la configuración del sistema puede ser personalizado para conocer la especificación en particular. Debido a esto el núcleo se reduce a su tamaño mínimo, por lo que requiere sólo unos pocos cientos de bytes de RAM y programas almacenados. Las dependencias entre los módulos se reducen a un mínimo absoluto.

1.5.2.2 Programador tick-less

A diferencia de muchos otros sistemas operativos que utilizan eventos, el planificador de RIOT trabaja sin estos, y se puede considerar como un planificador tick-less. Siempre que no hay tareas pendientes, RIOT cambiará al subproceso inactivo.

El subproceso inactivo posee una única función la cual es determinar el modo de sueño más profundo posible, dependiendo de los dispositivos periféricos en uso. De esta manera, se garantiza para maximizar el tiempo de permanencia en el modo de reposo, por lo tanto, reducir al mínimo el consumo de energía de todo el sistema.

Sólo detecciones externas o generadas por el Kernel pueden despertar al sistema del estado de reposo. Todas las funciones del núcleo se mantienen lo más pequeñas posible, permitiendo al kernel funcionar en sistemas con velocidades muy bajas de reloj.

1.5.2.3 Soporte de hardware

RIOT soporta varias MCUs (unidades de microprocesador o microcontrolador) tales como MSP 16-bit 430 o 32 bits ARM7, y una aplicación básica requiere menos de 5 kbytes de ROM y menos de 2 kbytes de RAM.

Sin embargo, si el microcontrolador ofrece características adicionales como por ejemplo, un controlador de interrupción Vectorial (VIC) que es proporcionada por muchos procesadores ARM, RIOT es capaz de beneficiarse de ellos, porque el código dependiente de la CPU está estrictamente separado de la implementación del núcleo.

Esta implementación del hardware, permite a la plataforma de desarrollo ser independiente de las funciones del kernel, las bibliotecas del sistema y las aplicaciones.

Sistemas embebidos que soportan Riot

- AirfyBeacon
- ArduinoDue
- Arduino Mega 2560
- Atmel samr21-Xplained Pro
- HiKoB FOX
- mbed NXP LPC1768
- MSB-IoT
- Nordic nrf51822 (DevKit)
- OpenMote

Arquitecturas de los Sistemas Embebidos

- MSP430
- ARM7
- Cortex-M0
- Cortex-M3
- Cortex-M4
- x86
- AVR

Soporte del Software Core

- 6LoWPAN: RFC6282 and RFC6775 compliant
- RPL: RFC6550 compliant
- OpenWSN
- CoAP, CBOR, and UBJSON
- Arduino API

1.5.3 Pila de red

Los protocolos de red para sistemas con bajos recursos como 6LoWPAN o RPL se proporcionan como soporte completo para IPv6, UDP y TCP. La implementación de la pila de red es totalmente modular, lo que permite el fácil intercambio de todos los protocolos en cualquier capa.

Además se proporciona una capa de adaptación para el controlador del transceptor de radio, que ofrece una interfaz compatible con IEEE 802.15.4, incluso si el propio transceptor de radio no proporciona este protocolo.

1.5.4 Características de fiabilidad y tiempo real

El kernel de RIOT se dirige a dispositivos utilizados en escenarios con pocos recursos, que requieren máxima fiabilidad y fuertes características de tiempo real. Por lo tanto RIOT soporta multi-threading en tiempo real ya que cuenta con:

- El manejador de interrupciones latencia-cero,
- Tiempos mínimos de conmutación combinado con las prioridades de procesos.

También se garantiza que el hilo o proceso con la más alta prioridad, no se encuentre bloqueado o dormido.

1.6 Software de simulación OMNeT++

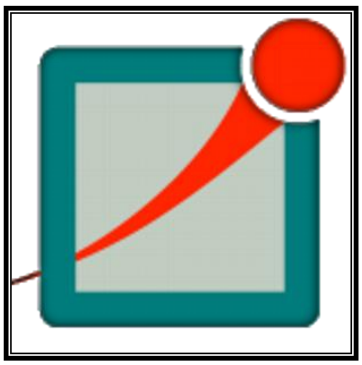


Figura 15-1. Software OMNeT++
Fuente: https://omnetpp.org/

OMNeT++ es un simulador de código abierto, entorno de simulación modular y de arquitectura basada en componentes con una fuerte GUI, basado en componentes C++ como biblioteca de simulación y frameworks, es extensible y modular, sobre todo para simular las construcciones de redes como redes de comunicaciones inalámbricas e inalámbricas, redes en chip, redes de colas, y así sucesivamente (Omnetpp, 2015)

Proporciona una arquitectura de componentes para los modelos, sus componentes (módulos), se programan en C++, luego se insertan en componentes y modelos de mayor tamaño utilizando un lenguaje de alto nivel (NED)

Posee funcionalidad de dominio específico, como el apoyo a las redes de sensores, redes ad-hoc inalámbricas, protocolos de Internet, modelado de rendimiento, redes fotónicas, es proporcionada por modelos frameworks, desarrollado como proyectos independientes.

OMNeT ++ ofrece un IDE basado en Eclipse, un entorno de ejecución gráfica, y una serie de herramientas. Hay extensiones para simulación en tiempo real, la emulación de red, la integración de base de datos, la integración SystemC, y varias otras funciones.

1.6.1 Modelos de Simulación

1.6.1.1 INET Framework

El INET Framework es un paquete de simulación, de la red de comunicaciones de código abierto para el entorno OMNeT++.Contiene modelos para varios protocolos de redes cableadas e inalámbricas

1.6.1.2 Mixim Framework

MIXIM es un simulador mixto que combina diversos marcos de simulación desarrollados para simulaciones inalámbricas y móviles en OMNeT ++. Proporciona modelos y protocolos detallados, así como una infraestructura de apoyo.

MIXIM está previsto para soportar la simulación de redes con más de 1000 nodos, por lo tanto, tiene un bajo consumo de memoria y su estructura modular permite la adaptación del nivel de detalle y por lo tanto el tiempo de ejecución. Una interfaz gráfica de configuración ayuda a elegir los módulos adecuados, apilarlas en capas, y asignar valores a sus parámetros.

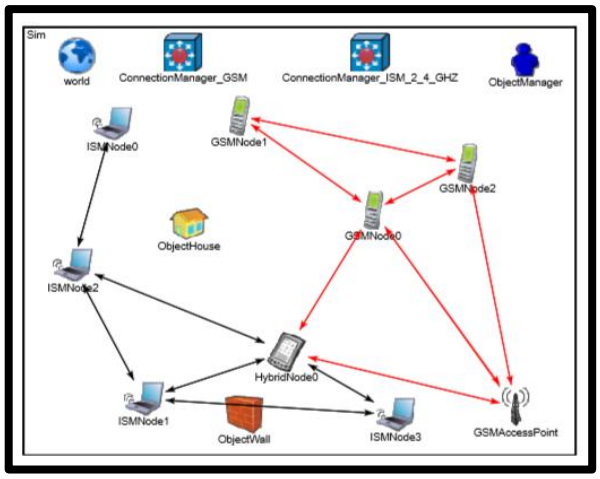


Figura 16-1. Red con Mixim framework
Fuente: https://omnetpp.org/

La figura 16-1 muestra una red de ejemplo MIXIM. El modelo ambiental está contenido en el módulo de utilidad mundo, que se utiliza principalmente para recoger parámetros globales como las dimensiones de la red

El módulo ConnectionManager es responsable de manera dinámica la gestión de las conexiones entre los nodos que interfieren. Se conoce la posición de todos los nodos y se puede consultar posiciones de los objetos de la ObjectManager. En general, MIXIM soporta múltiples administradores de conexión, responsables de diferentes rangos de frecuencia, como las ondas de radio y ultrasonido

1.7 Fenómenos y desastres naturales

Debido a que el planeta se encuentra en constante transformación, varias veces es sometido a colosales fuerzas tectónicas y cambios atmosféricos drásticos. Todos estos fenómenos naturales no son más que manifestaciones normales del entorno geográfico. (Román M., 2006, p.15)

Con el pasar del tiempo, y por fenómenos naturales como sismos, las estructuras de los puentes acumulan un deterioro constante, el cual es muy perjudicial no solo para las personas sino también para la ciudad.

Uno de los motivos de los sismos en todo el continente americano es que existen cuatro grandes placas tectónicas que la transforman en una zona proclive a sufrir terremotos. Las placas son: Placa del Pacífico (cubre toda la zona del Océano Pacífico), Placa Nazca (continente

sudamericano), Placa de Cocos (zona del Océano Pacífico Centroamérica) y la Placa del Caribe (zona del Mar Caribe en Centroamérica). (Diario El Comercio, 2014)

1.7.1 Monitoreo de salud estructural (MSE)

El monitoreo de estructuras se ha incrementado notablemente, y es practicado por ingenierías civiles, mecánicas y aeroespaciales, para averiguar en una etapa temprana si existe daño alguno en las estructuras. Actualmente solo existen métodos visuales o de localización experimental, como acústicos o ultrasónicos. (Marulanda J., et al., 2011)

El desarrollo de sistemas que detecten cambios en las características de vibración de la estructura fueron elaboradas para descubrir daños globales que pueden suceder en cualquier estructura compleja. El monitoreo tiene como objetivo primordial analizar medidas dinámicas como vibración, frecuencia y amortiguamiento, ya que estos cambian de acuerdo a las medidas físicas de la estructura como la firmeza, masa.

1.7.2 ¿Por qué es importante un MSE?

El Monitoreo de Salud Estructural (MSE) identifica los posibles daños en la estructura, mediante el análisis de datos, proporcionando así información sobre el comportamiento de las estructuras, que puede ser comparada con las predicciones del análisis estructural

El estado del arte del MSE está en detectar, localizar y cuantificar el daño que sufren las estructuras al ser sometidas a sismos, huracanes o cargas de tráfico, comparando con datos obtenidos antes de los diferentes desastres.

Los equipos que se emplean en el MSE consisten en sensores, acondicionadores de señal, dispositivos para adquisición de datos y soportes físicos y lógico (hardware y software) para la interpretación. Los sensores más usados son: acelerómetros, extensómetros, termocuplas, detectores de resistencia/temperatura, transformadores lineales de voltaje/desplazamiento y detectores de corrosión. (Marulanda J., et al., 2011)

1.7.3 Variables externas de vibraciones sobre puentes

Debido a que cualquier tipo de puente se encuentra suspendido en al aire o sujeto al suelo mediante sus pilares, existen variables externas las cuales crean vibraciones en los puentes, a continuación se describen las variables más comunes de vibración.

1.7.3.1 Sismos

Esta fuente de vibración se genera en base a un desastre natural y dependiendo de su magnitud puede provocar graves daños a la estructura de un puente. Debido a esto en el presente proyecto se tratara de simular este tipo de vibración externa sobre la estructura de un puente colgante a escala.

En la tabla 2-1 se puede apreciar los distintos niveles de sismos, según el Servicio Geológico de los Estados Unidos.

Tabla 2-1. Magnitudes sismográficas

Magnitud	Descripción	Efectos de un sismo
< 2,0	Micro	No son perceptibles.
2,0-2,9	Menor	Generalmente no son perceptibles.
3,0-3,9		Perceptibles a menudo, no causan daño
4,0-4,9	Ligero	Movimiento de objetos, genera ruido. Poco daño probable.
5,0-5,9	Moderado	Puede causar daños mayores en construcciones sin planes anti-sismos
6,0-6,9	Fuerte	Pueden causar daños irremediables, destruir pueblos
7,0-7,9	Mayor	Causa daños muy graves.
8,0-8,9	Gran	Muy devastadora a varios miles de kilómetros.
9,0-9,9		Muy devastadora a varios miles de kilómetros.
10,0+	Épico	Nunca registrado.

Realizado por: PALATE, Byron. 2016
Fuente: Servicio Geológico de los Estados Unidos

1.7.3.2 Oscilación por el viento

Este tipo de vibración externa es la más común, debido a que los puentes se encuentran suspendidos en el aire, en nuestro entorno esta fuente de vibración es muy ligera debido a que nos encontramos en la cordillera de los andes y esto neutraliza en gran cantidad toda la velocidad con la que puede correr el viento, en el caso de un puente colgante las pequeñas corrientes de aire casi son neutralizadas mediante sus pilotes de cemento que se encuentran firmemente establecidos sobre el suelo.

1.7.3.3 Circulación de transporte pesado

Esta fuente de vibración al igual que las corrientes de aire son muy comunes, con la diferencia que para neutralizar este tipo de vibraciones, los expertos en construcción de puentes realizan un estudio para poder establecer los planos de construcción del puente, es allí donde se establecen reglas o estándares los cuales ayuden a soportar el peso máximo del transporte al igual que neutralizar los movimientos causados por los mismos.

1.7.4 Como afecta un sismo a la estructura de un puente

La estructura de los puentes en muchos de los casos son pilotes de cemento y/o acero, clavados en el piso a gran profundidad, entonces con el sismo se mueven junto con la tierra, estando bien firmes, no corren peligro, salvo que sea un terremoto catastrófico.

Un monitoreo estructural de los puentes ayudaría a tomar decisiones tempranas sobre su mantenimiento, ya que después de un sismo por muy ligero que sea este, ninguna estructura queda en su estado natural, y esto con el pasar del tiempo al enfrentarse a otro sismo de igual o mayor magnitud podría colapsar y causas graves accidentes.

En la figura 17-1, se puede observar la caída del puente en la Av. Las Américas de la ciudad de Guayaquil-Ecuador después del último terremoto registrado el 16 de abril del presente año,

Figura 17-1. Colapso de Puente en Guayaquil-Ecuador
Fuente: http://images.prensa.com

1.7.5 Cronología de movimientos telúricos en Ecuador

Ecuador ha experimentado una fuerte actividad sísmica en los últimos 50 años. Muchos de estos eventos han destruido ciudades enteras. Algunas de las ciudades donde se registraron sismos según el Instituto geofísico de la Escuela Superior Politécnica Nacional (IGEPN) fueron, Latacunga, Ibarra, Ambato, Riobamba, Manabí, Esmeraldas, Cuenca.

El último movimiento telúrico de gran magnitud registrado en Ecuador fue el pasado 16 de Abril del presente año, los datos que proporciona la (IGEPN) son los apreciados en la figura 18-1, la cual tuvo muchas desgracias en ciudades como Pedernales, Muisne, Portoviejo.

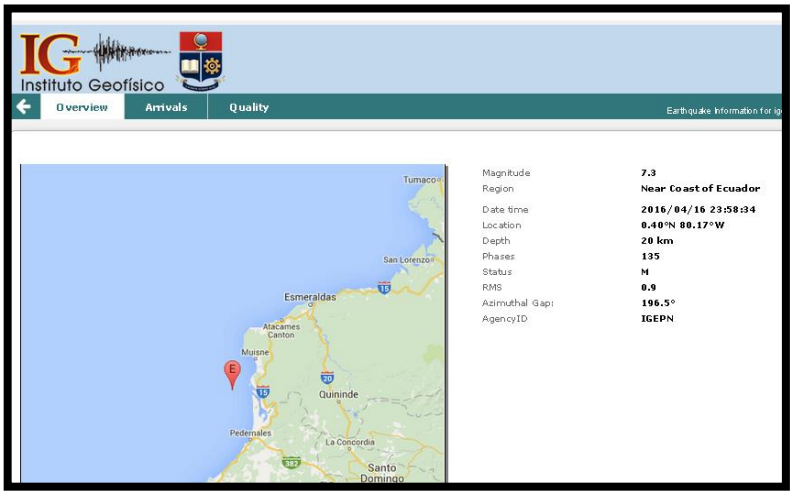


Figura 18-1. Reporte sísmico del Instituto Geofísico
Fuente: http://eventos.igepn.edu.ec/eqevents/event/igepn2016hnmu/igepn2016hnmu-gmapa.png

CAPITULO II

2. MARCO METODOLÓGICO

1.8 Diseño del entorno de evaluación de plataformas 6LowPan y monitoreo estructural de puentes

1.8.1 Introducción

Mediante la evaluación de plataformas 6LowPan, se pretende dar a conocer que plataforma es la más eficiente para su uso en el Internet de las cosas, ya que debido a que existen varios sistemas embebidos de código abierto se produjeron varias plataformas que de una u otra manera satisface o cumple los parámetros necesarios para su funcionamiento en la IoT.

El diseño del entorno es una parte muy importante para el desarrollo del proyecto, ya que en esta sección de define aquellas variables que se medirán para el monitoreo estructural de puentes, además se detallara la manera de procesar la información y de la transmisión hacia la estación base.

Se tiene bien definido todos los requisitos tanto en software como en hardware que son necesarios para el diseño del entorno, ya que no todas las plataformas son compatibles con ciertos sistemas embebidos desarrollados para el IoT.

1.9 Diseño de la WSN basada en 6LowPan

1.9.1 Descripción del entorno

El entorno será evaluado con dos plataformas distintas, tanto Contiki-OS como Riot-OS, esto debido a que nos centraremos en verificar qué plataforma es la más adecuada para su uso en las

IoT, con respecto al consumo de recursos necesarios que hacen fundamental el funcionamiento de las motas.

Los recursos que se analizaran en cada mota están bien definidos, los cuales son energía, memoria y procesamiento, estos datos son muy fundamentales para un buen desenvolvimiento de las redes 6LowPan, como lo indican los estándares definidos en los RFC6606 y RFC 4919.
Para poder evaluar estos parámetros de las plataformas, la red WSN será establecida sobre una maqueta de un puente, desde donde se transmitirá la información recogida por las motas hacia un coordinador.

Como se puede ver en la figura 1-2, el entorno está conformado por 4 motas, 3 de ellas colocadas en puntos estratégicos del puente, y la cuarta recibirá la información provenientes de las motas sensores. Cada mota será instalada en los puntos más vulnerables de la estructura, las cuales siempre deben ser analizadas después de una catástrofe natural como son los movimientos telúricos.

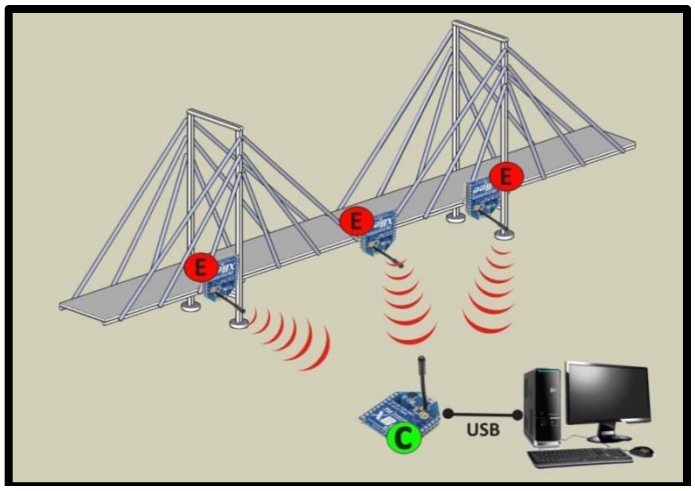


Figura 1-2. Distribución de motas
Fuente: PALATE, Byron. 2015.

1.9.2 Simulación de la WSN en OMNeT++

Antes de proceder a construir nuestro prototipo, es necesario desarrollar un escenario en un entorno de simulación, se optó por usar el entorno de simulación llamado OMNeT++, debido a que entre sus principales características ofrece un resultado tan cercado a la plataforma de Contiki y también de Riot.

1.9.2.1 Escenario WSN con 6LowPan en OMNeT++

El escenario que se implementó en el simulador es igual al propuesto, que consta de 4 motas, 3 de ellas motas sensores quienes obtienen y envían la información y 1 coordinador la cual recibe toda la información y la interpreta.

Las características de cada mota fueron configuradas mediante líneas de código que ofrece el simulador, así también como la distribución de las motas. La dirección ipv6 de cada mota también fue configurada en el simulador.

En la figura 2-2se puede observar la distribución de las motas sobre el escenario de simulación, el mismo que consta tanto de sus sensores así como de su coordinador

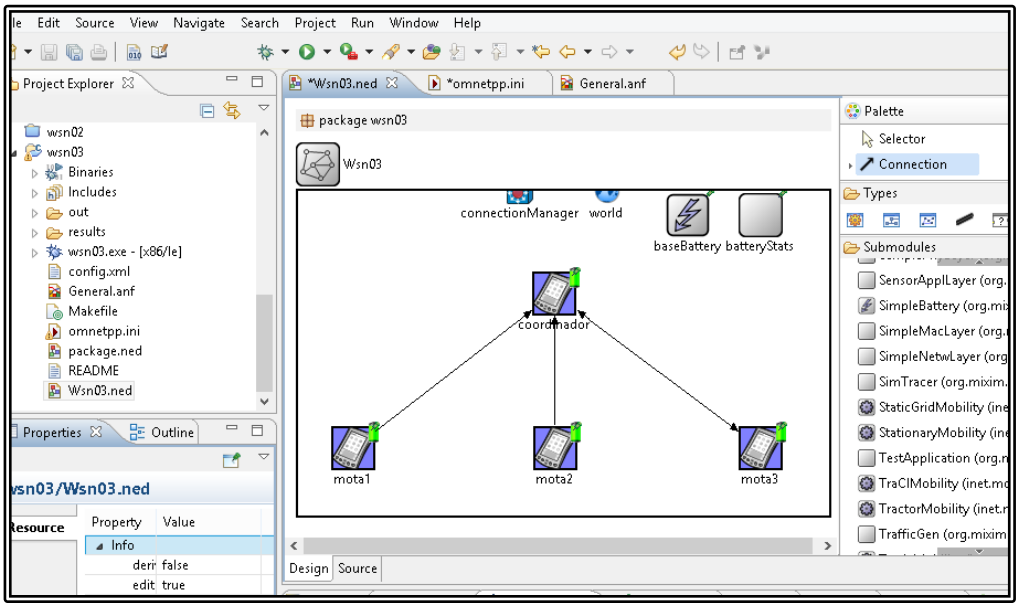


Figura 2-2. Distribución de motas en OMNeT++
Fuente: Palate B. 2015.

En la simulación de OMNeT++ se establece la dirección de envió de paquetes, por lo cual en cada mota sensor se estableció un sentido hacia el coordinador, como se puede apreciar en la figura 2-2.

1.9.2.2 Resultados del escenario WSN con 6LowPan en OMNeT++

Para analizar los recursos que consume cada mota, se debe poner en marcha la simulación y al terminarla se creara una carpeta llamada “Results” en nuestro proyecto, donde se crearan archivos que contienen los resultados de nuestra simulación, existen resultados de todas las capas que conforman el sistema.

En la figura 3-2 se muestra el consumo energético de la mota coordinador, cuyo valor fue de 98mA.

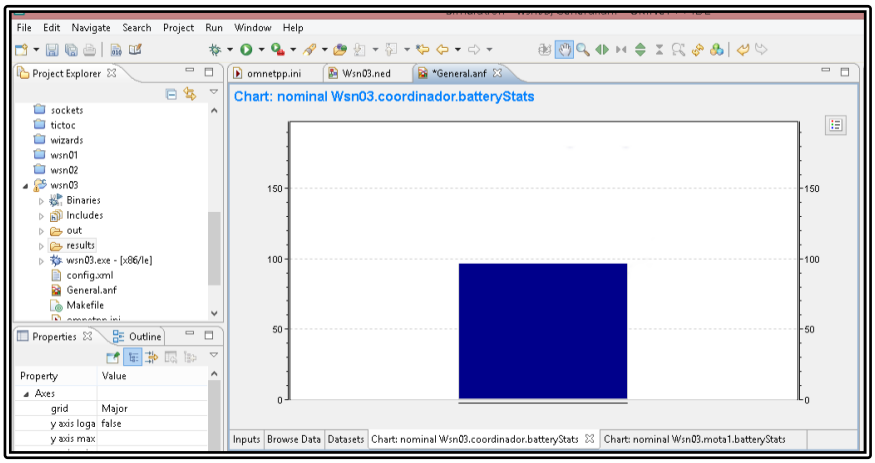


Figura 3-2. Consumo energético mota coordinador
Fuente: PALATE, Byron. 2015.

En la figura 4-2, se puede observar el consumo energético de la mota sensor 1, donde su consumo fue de 103mA.

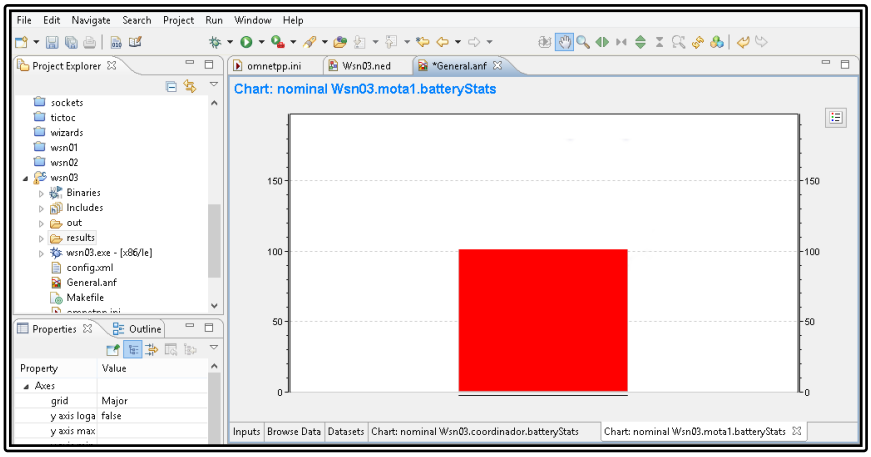


Figura 4-2. Consumo energético mota sensor 1
Fuente: PALATE, Byron. 2015.

1.9.3 Selección del Hardware

En la presente sección se detallara todo el hardware necesario para la implementación física del escenario 6LowPan.

1.9.3.1 Sensor ADXL335

El sensor más adecuado para detectar un sismo es un acelerómetro, debido a que detecta variaciones en sus tres ejes principales (x,y,z), trabaja a temperatura ambiente que oscila entre los -40 C a 85 C, es excepcionalmente ligero, y el tamaño del módulo PCB es solo de 16* 20 mm, es muy conveniente para la incorporación de hardware para proyectos de ingeniería.

Su uso cotidiano es para la implementación de sistemas de videojuegos, dispositivos móviles, o proyectos que necesiten censar algún tipo de movimiento de un dispositivo.

Existen variedad de sensores acelerómetros, pero lo que nos permitió decidir sobre el uso de este sensor fue su bajo costo, fácil implementación y sobre todo sus salidas con valores de tensión analógicas, ya que esto nos permite tener diferentes niveles de variación en sus ejes.

En la tabla 1-2, se muestran las principales características del sensor acelerómetro ADXL335, las cuales fueron ideales para su selección.

Tabla 1-2: Características del sensor Adxl335

Chip sensor	ADXL335
Sensor	Campo magnético de tres ejes
Rango de tensión de funcionamiento	3 V ~ 5 V
Corriente de alimentación	350μa
Interfaz	Salida de Analógica
Temperatura de funcionamiento	-40c ~ 85C
Tamaño (l x W x h)	Aprox. 0.8x0.6x0.4 pulgadas

Realizado por: PALATE, Byron. 2015.
Fuente: PALATE, Byron. 2015

En la figura 5-2, se puede observar las dimensiones del sensor acelerómetro ADXL335, lo cual comprueba su tamaño ideal para su uso en proyectos de ingeniería.

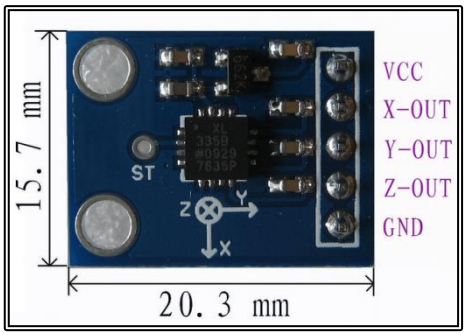


Figura 5-2. Sensor Adxl335
Fuente: http://www.sunfounder.com

1.9.3.2 Arduino Mega 2560 R3

El Arduino mega 2560 R3 es una placa electrónica de código abierto (Open Source), basado en el microcontrolador ATMEGA2560, el cual fue desarrollado para la elaboración de varios proyectos complejos que necesiten muchas entradas analógicas así también como digitales.

Se optó por utilizar este sistema embebido gracias a su potente microcontrolador que es capaz de soportar las plataformas con el stack IPv6 sobre las WSN, las cuales serán evaluadas en el presente documento.

Actualmente estos sistemas embebidos son utilizados para elaborar impresoras en 3D y proyectos de robótica, su infraestructura es totalmente compatible con Shield para Arduino uno y Duemilanove o Diecimila.

Este sistema entra en funcionamiento al conectar su puerto USB al computador el cual proporciona el voltaje necesario para su buen funcionamiento, otra manera de ponerlo en marcha es conectarlo a una batería que brinde un voltaje que se encuentre entre sus rangos permitidos para que así no pueda causar daño alguno.

En la figura 6-2 se puede observar las placas tanto Arduino como genuino mega 2560, su nombre solamente varía de acuerdo al estado en donde se realice su compra.

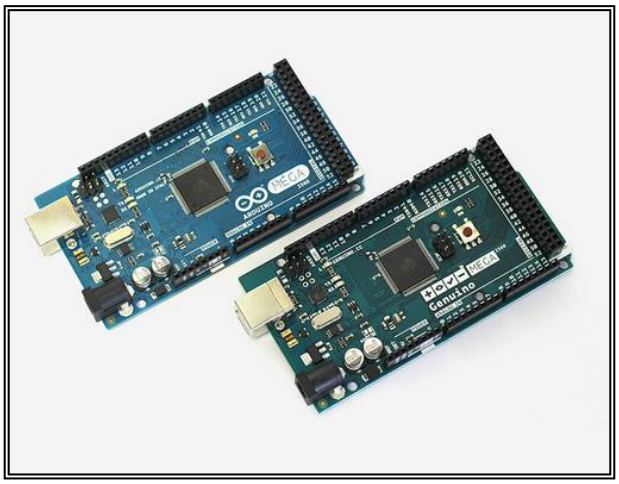


Figura 6-2. Arduino Mega 2560 R3
Fuente: https://www.arduino.cc/en/Main/arduinoBoardMega2560

En la tabla 2-2 se da a conocer las principales características de la placa Arduino mega 2560.

Tabla 2-2: Características Arduino Mega 2560 R3

Características	Valores
Microcontrolador	ATmega2560
Voltaje de Operación	5V
Voltaje de entrada (recomendado)	7-12V
Voltaje de entrada (limite)	6-20V
Pines Digitales E/S	54 (15 proporciona salida PWM)
Pines Analógicos Salida	16
Corriente DC por pin E/S	20 mA
Corriente DC para el Pin 3.3 V	50 mA
Memoria Flash	256 KB pero 8 KB son usados para arranque
SRAM	8 KB
EEPROM	4 KB
Velocidad del reloj	16 MHz
Longitud	101.52 mm
Ancho	53.3 mm
Peso	37 gr.

Realizado por: PALATE, Byron. 2015.
Fuente: https://www.arduino.cc/en/Main/arduinoBoardMega2560

1.9.3.3 Módulos de Comunicación Xbee S1

Xbee S1 o también conocido como Xbee 802.15.4, son módulos de Radio Frecuencia que integran soluciones para proporcionar conectividad inalámbrica a los dispositivos. Estos módulos utilizan el protocolo de red IEEE 802.15.4 para conexiones punto a multipunto o redes peer-to-peer. Están diseñados para aplicaciones de alto rendimiento que requieren baja latencia y sincronización de la comunicación predecibles.

La figura 7-2 es una representación gráfica del módulo de comunicación Xbee S1 usado en la implementación del presente trabajo.

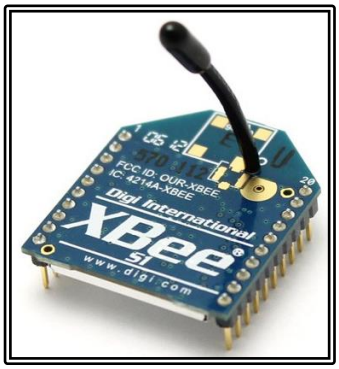


Figura 7-2. Módulo de Comunicación Xbee
Fuente: https://ecs7.tokopedia.net

Los módulos de Radio Frecuencia XBee son ideales para la transmisión en baja potencia, aplicaciones de bajo costo. Surgen de la familia de productos de RF XBee, estos módulos son fáciles de usar, y son totalmente compatibles con otros productos XBee que utilizan la misma tecnología. Los usuarios tienen la capacidad de sustituir un módulo XBee por otro con un mínimo tiempo de desarrollo y de riesgo.

En la tabla 3-2 se da a conocer las principales características técnicas del módulo de comunicación XBee s1,

Tabla 3-2: Características del módulo Xbee S1

Rendimiento	
Potencia de salida	1 mW (0 dBm)
Alcance de la Señal	Hasta 100 pies (30 m)
Sensibilidad del receptor	-92 dBm

Velocidad de datos RF	250 Kbps
Frecuencia de operación	2.4 GHz
Velocidad de datos de interfaz	Hasta 115,2 Kbps
Redes	
Topologías de red compatibles	Punto a punto, punto a multipunto
Manejo de errores	Reintentos y reconocimientos
Opciones de filtración	PAN ID, Canal, y las direcciones de 64 bits
Capacidad del canal	16 Canales
Direccionamiento	65.000 direcciones de red disponibles
Energía	
Tensión de alimentación	3.0 - 3.4 VDC
Corriente de transmisión	45 mA (@ 3.3 V) en modo de refuerzo 35 mA (@ 3,3 V) en modo normal
Corriente recibida	50 mA (@ 3.3 V)
Corriente en modo sleep	<10 µA a 25° C
Propiedades físicas	
Tamaño	2,438 cm x 2,761 cm
Temperatura de funcionamiento	-40 ° C a 85 ° C (industrial)

Realizado por: PALATE, Byron. 2015.
Fuente: http://www.digi.com/products/xbee-rf-solutions/modules/xbee-802-15-4#specifications

1.9.3.4 Shield Xbee

El módulo o Shield XBee es compatible con los estándares 802.15.4 ZigBee / IEEE para la comunicación en las redes de sensores inalámbricas de bajo consumo energético El Shield es fácil de usar, de bajo consumo energético, trabaja en la banda de operación de frecuencia de la ISM de 2,4 GHz. El Shield es muy utilizado en los países de Estados Unidos, Canadá, Australia, Israel y Europa.

El shield XBee se caracteriza por no requerir ninguna configuración para su funcionamiento, dicho shield se puede apreciar en la figura 8-2.

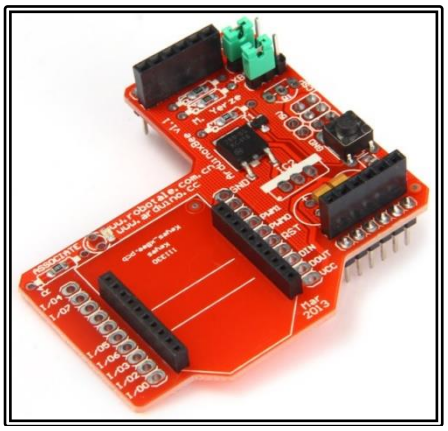


Figura 8-2. Shield módulo de Comunicación Xbee
Fuente: http://www.everbuying.net/product1093202.html

1.9.4 Configuración de Hardware y Software

En esta sección se detallara las herramientas software que se utilizaron para poder configurar el hardware necesario para el desarrollo del proyecto.

1.9.4.1 XCTU

XCTU es un software libre desarrollado por DIGI que funciona sobre Windows, fue diseñado para que los desarrolladores puedan interactuar con los módulos Digi RF de una manera gráfica, permitiendo así que su configuración y testeo sea de una manera muy fácil y amigable.

En la figura 9-2 se puede apreciar la pantalla principal del software de configuración XCTU, el cual fue usado para configurar nuestros módulos de comunicación.

Figura 9-2. Software XCTU
Fuente: http://www.digi.com/products/xbee-rf-solutions/xctu-software/xctu

Para configurar los dispositivos xbee, abrimos el software XCTU y seleccionamos la opción "Descubrir Radios", donde seleccionamos el puerto conectado a nuestro xbee, una vez encontrado el modulo ya solo nos queda seleccionarlo y configurarlo con nuestras necesidades. Terminada la configuración seleccionamos la opción "Write" como se puede apreciar en la figura 10-2, la cual guardara la configuración actual en nuestro módulo xbee.

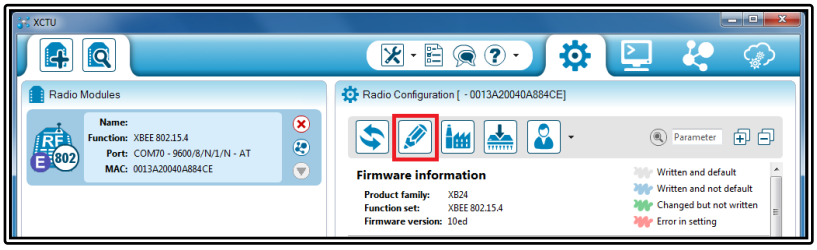


Figura 10-2. Opción write en xctu
Fuente: http://xbee.cl/xbee-serie-1-configuracion/

Campos importantes

Para configurar correctamente un módulo xbee, se debe tener muy en cuenta algunos parámetros los cuales indican si el modulo es coordinador o dispositivo final, dichos parámetros se pueden apreciar en la tabla 4-2.

Tabla 4-2: Campos de configuración en Xbee

DH	DestinationAddress High
DL	DestinationAddress Low
MY	16-bit Source Address
ID	PAN ID
SH	Serial Number High
SL	Serial Number Low
CE	CoordinatorEnable

Realizado por: PALATE, Byron. 2015
Fuente: PALATE, Byron. 2015

A continuación se detallara uno a uno los campos de la tabla 4-2.

Dirección alta de destino (DH): Es la primera mitad de la dirección del módulo de destino. Los módulos XBee tienen una dirección de 64 bits en su parte posterior, por lo que DH es la parte alta de 32-bit de esta dirección. (Ver figura 11-2).

Dirección baja de destino (DL): Es la dirección dela parte baja de 32-bits del Xbee de destino. (Ver figura 11-2).

MY Address (MY): Esta es la dirección de 16 bits de origen de un XBee, es una dirección única para cada módulo en particular dentro de la red.

Pan ID: es el identificador de la Red de Área Personal en la que trabajaran los módulos xbee, solo los XBee asignados a un PAN ID pueden comunicarse entre sí.

Número alto de serie (SH): Esta dirección ya viene configurada por defecto en el software

Número bajo de serie (SL): Esta dirección ya viene configurada por defecto en el software

Habilitar coordinador (CE): En esta opción se puede elegir si el modulo trabajara como coordinador o como dispositivo final/Router.

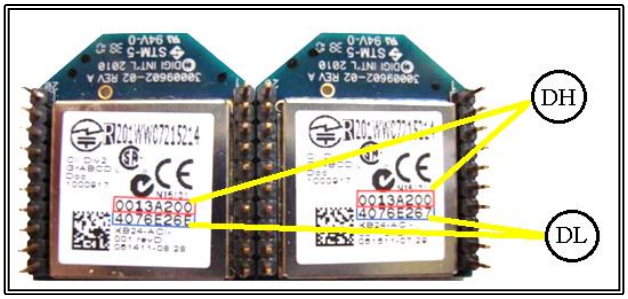


Figura 11-2. Direcciones físicas Xbee
Fuente: http://xbee.cl/xbee-serie-1-configuracion/

Sleep Mode

XCTU nos ofrece la configuración de modo Sleep la cual nos ayudara para extender la duración de la batería de un módulo Xbee, debido a que la mota entra en estado de "Dormir" un determinado tiempo y despierta por otro determinado tiempo para revisar si tiene que transmitir datos.

Los campos que se deben configurar son:

SM: Escoge el modo de Sleep y lo activa.

- SM=1 and SM=2: Pin-controlador sleep mode, pone en alto el pin 9 conectándolo a 3.3V para entrar en modo dormir.

- SM=4 and SM=5: Cyclic sleep mode, habilita el modo dormir pero despierta en un horario fijo, para lo cual debe ser configurado los parámetros ST, SP

ST: Define el periodo de inactividad de la mota antes de ir a Dormir, se ingresan únicamente valores hexadecimales.

C8 (hexadecimal) = 200 (decimal) x 10 = 2segundos

SP: Define el tiempo de la duración que la mota dormirá, se ingresan únicamente valores hexadecimales.

190 (hexadecimal) = 400 (decimal) x 10 = 4 segundos

En la figura 12-2 se muestra la configuración del modo sleep de la mota sensor en el software xctu.

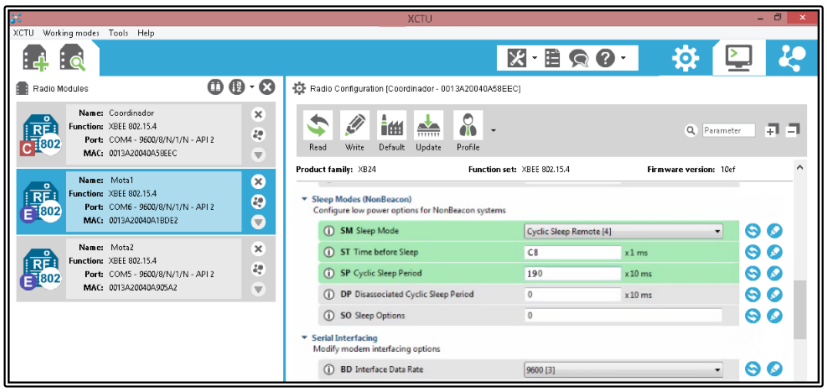


Figura 12-2. Configuración modo sleep
Fuente: PALATE, Byron. 2015

1.9.4.2 Arduino IDE

Es un entorno de desarrollo integrado (IDE), desarrollado por Arduino especialmente para programar todos sus sistemas embebidos tanto Arduino como Genuinos. Como se puede ver en la figura 13-2 este entorno posee un editor de texto para el código, un área de mensajes, comunicación con puertos serie, es el entorno más conocido desarrollado y mantenido por su empresa.

Figura 13-2. Arduino IDE
Fuente: PALATE, Byron. 2015

Arduino IDE es un software multiplataforma de licencia libre que puede compilar y cargar sus programas o sketchs fácilmente sobre sus plataformas, su único problema es que no marca los errores sobre el código, es decir solo los notifica pero no los muestra exactamente su ubicación.

1.10 Direccionamiento Ipv6

El direccionamiento IPv6 de cada mota está dado por un prefijo de red que es el único conocido por el contexto 6LowPan conjuntamente con la dirección DH y DL de cada módulo Xbee, de esta manera se asegura que la dirección IPv6 de cada mota no se repita en la red.

De esta forma las direcciones de las motas sensores y coordinador quedaría establecidas como se muestran en la tabla 5-2.

Tabla 5-2: Direccionamiento de las motas

Mota	Dirección IPv6
Coordinador	0xfe80,0x0000,0x0000,0x0000,0x0213,0xA200,0x40A5,0x8EEC
Mota Sensor 1	0xfe80,0x0000,0x0000,0x0000,0x0213,0xA200,0x40A1,0xBDE2
Mota Sensor 2	0xfe80,0x0000,0x0000,0x0000,0x0213,0xA200,0x40A9,0x05A2
Mota Sensor 3	0xfe80,0x0000,0x0000,0x0000,0x0213,0xA200,0x40A1,0xBD51

Realizado por: PALATE, Byron. 2015.
Fuente: PALATE, Byron. 2015

1.11 Vectores Información de las Motas

El vector de información es un campo de la trama UDP en la cual se inserta la información recogida por los sensores para luego ser transmitida mediante 6LowPan, esta información es transmitida a la mota coordinador en donde se quitan los diferentes campos y se extrae la información enviada por las motas sensores para procesarla

1.11.1 Vector de Información del Coordinador

El vector de información del coordinador es irrelevante, esto debido a que únicamente envía números del 1 al 9 en su vector. La mota coordinador envía periódicamente mensajes multicast a todos los nodos a la dirección ff02::1/128, en un principio estos mensajes son enviados para descubrir nuevas motas vecinas, luego envía mensajes de sincronización de temporizadores con sus vecinos asociados.
La importancia de estos mensajes son los demás campos, ya que contienen información muy valiosa como es la dirección de origen, destino, puertos.

1.11.2 Vector de Información de las Motas

Como se puede ver en la tabla 6-2, el vector de información de las motas está formado por tres dígitos que representa el eje x,y,z respectivamente, si se encuentra un "0" es debido a que no hubo movimiento en el eje, si se encuentra "1" existe movimiento ligero, "2" movimiento moderado y "3" movimiento fuerte.

Tabla 6-2: Vectores de información de motas

Posición	[0]	[1]	[2]
Dato	"0/1/2/3"	"0/1/2/3"	"0/1/2/3"

Realizado por: PALATE, Byron. 2015.
Fuente: PALATE, Byron. 2015.

1.12 Configuración de Contiki

Como primer paso debemos descargarnos o clonar los archivos necesarios de Contiki-OS desde el repositorio oficial de GitHub como se muestra en la figura 14-2, el cual contiene todos los archivos de configuración y librerías, entre los cuales se encuentra IPv6Stack, ejemplos core, cpu.

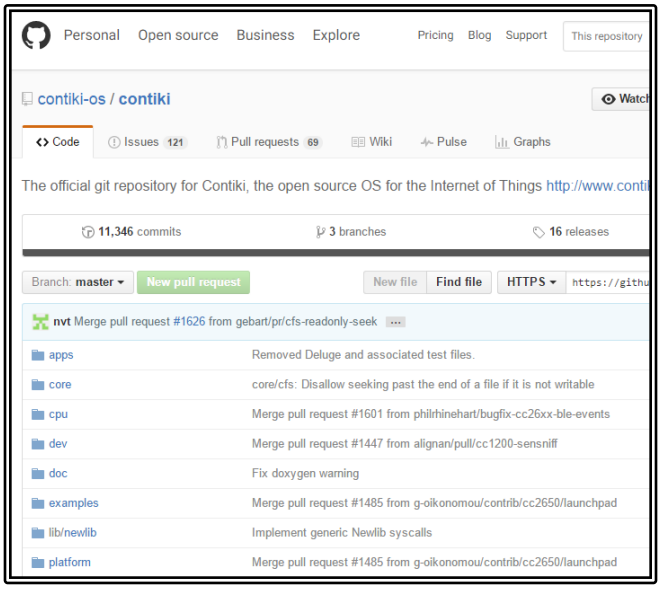


Figura 14-2. Repositorio Contiki
Fuente: https://github.com/Contiki-os/Contiki

Contiki posee un ejemplo de 6LowPan llamado IPv6UDP, el cual permite familiarizarnos con su código desarrollado en C, al ser desarrollado en este lenguaje es muy similar a la programación en el IDE de Arduino para lo cual abrimos nuestro entorno de desarrollo y comenzamos a realizar nuestro programa basándonos en el ejemplo antes mencionado.

La librería IPv6Stack.h posee toda la configuración necesaria para la comunicación mediante el estándar 6LowPan la cual inicializa el protocolo IPv6 y además el puerto UDP, para esto también necesita la librería llamada XbeeMacLayer.h, la cual inicializa el módulo Xbee para la transmisión inalámbrica de datos.

1.12.1 Archivo de Configuración

Debemos tener en cuenta el nodo que vamos a configurar, debido a que dependiendo de esto debemos modificar el archivo de configuración de Contiki que se encuentra en el directorio clonado de GitHub.

Debido a que Contiki está basado en C, tenemos que modificar el archivo Contiki_config.h como se muestra en la figura 15-2, se debe especificar si la configuración de Contiki trabajara como una mota sensor o como un coordinador, el parámetro a configurar es:

Para motas sensores:

- UIP_CONF_ROUTER 0

Para motas coordinador o routers

- UIP_CONF_ROUTER 1

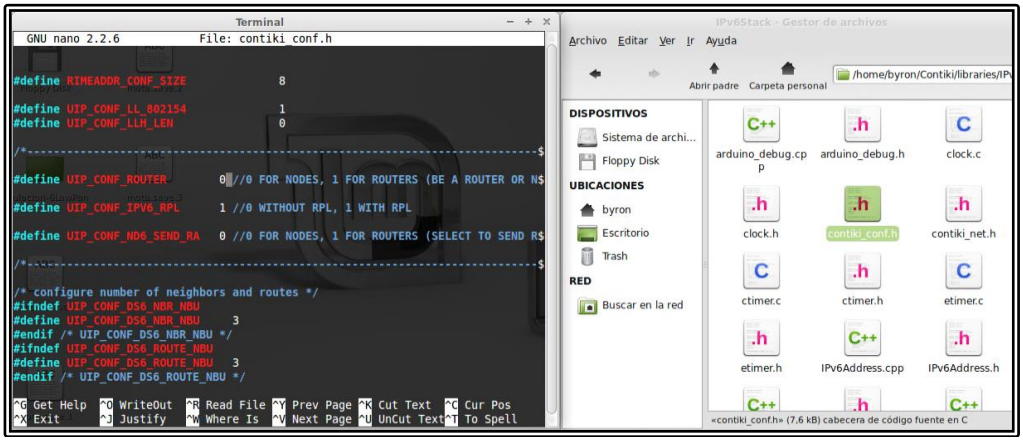


Figura 15-2. Archivo de configuración Contiki
Fuente: PALATE, Byron. 2016

1.12.2 Programación en Contiki

Luego de configurar el archivo de Contiki, comenzamos a realizar nuestro programa en nuestro entorno de desarrollo, como primer punto debemos llamar a las librerías necesarias como son IPv6Stack.h y XbeeMacLayer.h para que el programa pueda interpretar los comandos establecidos, continuando con la programación se puede decir que el parámetro más importante es la dirección IPv6 de cada mota, donde:

- Todas las motas sensores deberán tener como dirección de destino de sus mensajes la dirección IPv6 de la mota coordinador.

- La mota coordinador deberá establecer en su configuración las direcciones IPv6 de las motas sensores, debido a que esta recibirá los mensajes de todas las motas.

Además de esto, aquí también debemos especificar si el programa va hacer cargado sobre la mota sensor o sobre la mota coordinador, como se puede observar en la figura 16-2, el parámetro a editar es CONF_ROUTER&&0/1.

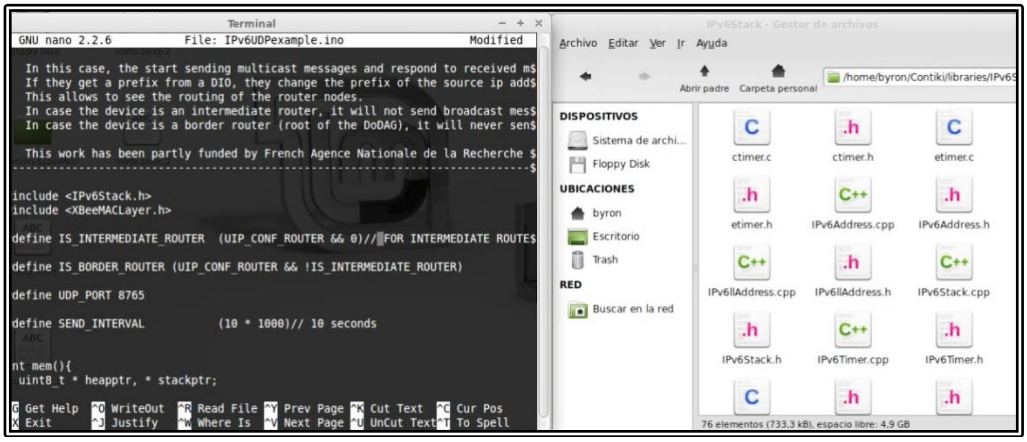


Figura 16-2. Programación Contiki
Fuente: PALATE, Byron. 2016

1.12.2.1 Consumo de memoria

El estado de memoria del dispositivo es muy importante debido a que el dispositivo posee una RAM muy limitada, debe poseer una cantidad de memoria suficiente para poder ejecutar las variables locales del programa. Para obtener el valor de memoria utilizaremos los punteros de "heap" y "stack" que colocan su valor actual en variables.

El puntero del stack se inicia en la parte superior de RAM y crece hacia abajo. El puntero de heap comienza justo por encima de las variables estáticas, y crece hacia arriba.

Con la ayuda de estos punteros podemos declarar variables de tipo uint8_t, que son las adecuadas para el cálculo sobre los bits, ya que uint8_t es una forma abreviada de un entero sin signo de 8 bits de longitud.

Como primer paso declararíamos las variables

uint8_t * heapptr, * stackptr

Luego utilizáramos temporalmente el punto Stack

stackptr = (uint8_t *) malloc (4)

Asignamos el valor del puntero heap a la variable heapptr

heapptr = stackptr

Liberamos la memoria nuevamente, es decir los conjuntos stackptr la volvemos a 0.

free(stackptr)

Guardamos el valor del puntero del stack

stackptr = (uint8_t *) (SP)

Por último al restar estas dos variables nos devolverá el resultado de la memoria disponible.

stackptr - heapptr;

Todo este procedimiento lo podemos guardar como una función para que sea llamada en cualquier parte del programa donde se requiera conocer la memoria disponible para la ejecución del sketch.

1.12.2.2 Velocidad de procesamiento

Para conocer la velocidad de procesamiento es decir saber cuánto tarda o cuantas veces puede hacer el proceso por segundo de un programa en el sistema embebido, se utilizó la función micros (), disponible en Contiki, el cual devuelve el valor en microsegundos al ser llamado, de esta manera se puede conocer el tiempo en el que se inicia y finaliza el proceso.

Para hacer uso de esta función se debe declara una variable de tipo "unsignedlong", debido a que la velocidad de procesamiento superara muy rápidamente a un valor entero ordinario. "Unsignedlong" amplían el tamaño de las variables para el almacenamiento numérico y almacenar 32 bits (4 bytes). A diferencia de long estándar, este no almacena números negativos, por lo que su rango de 0 a 4294967295 (2 ^ 32 - 1).

Su declaración sería algo similar a esto:

```
unsignedlongtiempoInicial = micros();

{ Aqui el programa }

unsignedlongtiempoFinal = micros();
```

Por último tendríamos que buscar la diferencia de estos valores para encontrar el tiempo de procesamiento del programa, cabe recalcar que el tiempo de procesamiento se lo realiza en microsegundos y no en milisegundos.

1.12.2.3 Consumo energético

El consumo energético de cada mota es muy indispensable, ya que es una de las principales inconvenientes y debilidades de una red WSN, ya que las motas deben de funcionar de manera autónoma por un muy largo tiempo.

Para reducir el consumo de energía en el sistema, es necesario hacer uso del modo de trabajo denominado "Sleep", como se explicó en el apartado de configuración de los módulos Xbee estos pueden trabajar en este modo el cual nos ahorrara gran cantidad de consumo energético.

Debido a que no existe una función o procedimiento en la programación para determinar este tipo de recurso, se optó por realizar las mediciones de manera manual, es decir evaluar el consumo energético de las motas una por una, tanto el consumo que genera el sistema embebido por sí solo, como también el consumo al funcionar junto al Shield y el módulo de transmisión XBee en caso del coordinador, y también con el sensor en caso de las motas.(Ver apartado 3).

1.12.3 Proceso de Arranque de Contiki

El proceso de arranque de la plataforma Contiki atribuye a la inclusión de varias librerías y procesos que son iniciados por etapas, por lo cual se puede representar en un Diagrama de flujo ilustrado en la figura 17-2.

El arranque de los dispositivos que se basen en Contiki lo harán de esta manera, sea nodo, router o coordinador, después del arranque la tarea que debe ejercer cada mota es diferente y será un flujograma diferente.

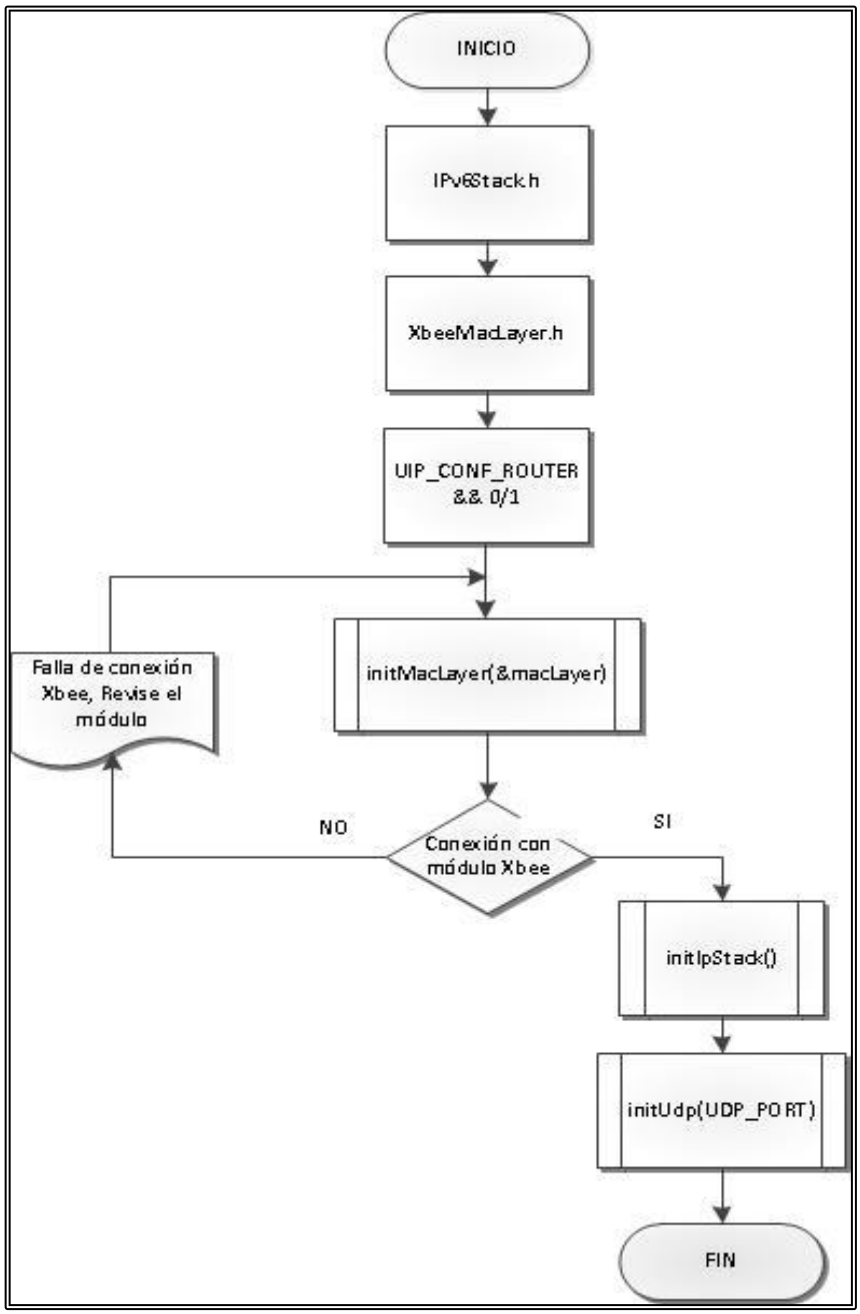


Figura 17-2. Arranque Contiki
Fuente: PALATE, Byron. 2015

1.12.4 Funcionamiento de las motas sensor

Las motas sensores después de su arranque con el sistema, siguen un esquema el cual les permite tomar decisiones en base a los valores censados para poder enviárselos hacia el coordinador, dicho proceso también puede esquematizarse en un flujograma y se representa en la figura 18-2.

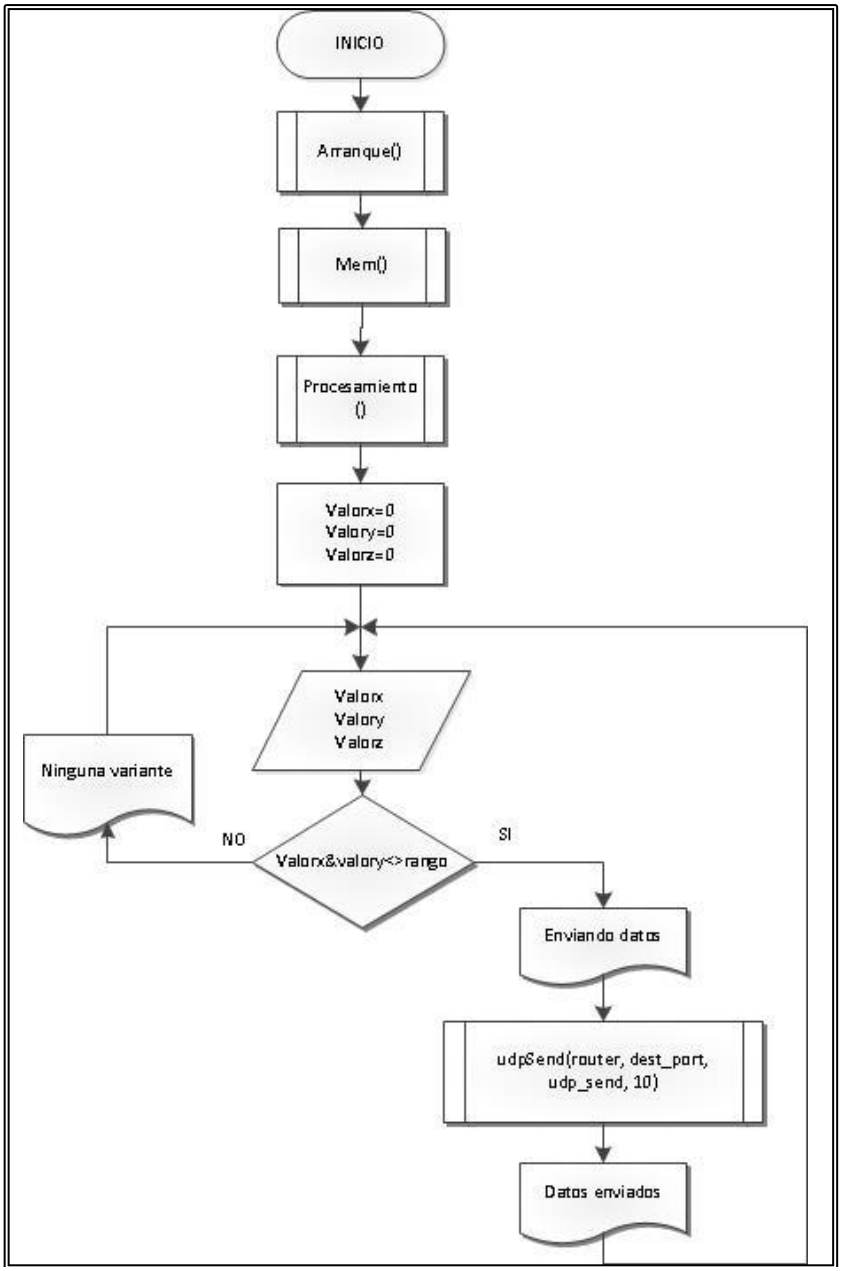


Figura 18-2. Funcionamiento de las motas sensores
Fuente: PALATE, Byron. 2016

1.12.5 Funcionamiento de la mota coordinador

La mota coordinador después de su arranque posee un esquema diferente, debido a que este recibirá los mensajes de las motas para poder alertar sobre un acontecimiento anormal en la estructura del puente, dicho procedimiento se representa en la figura 19-2.

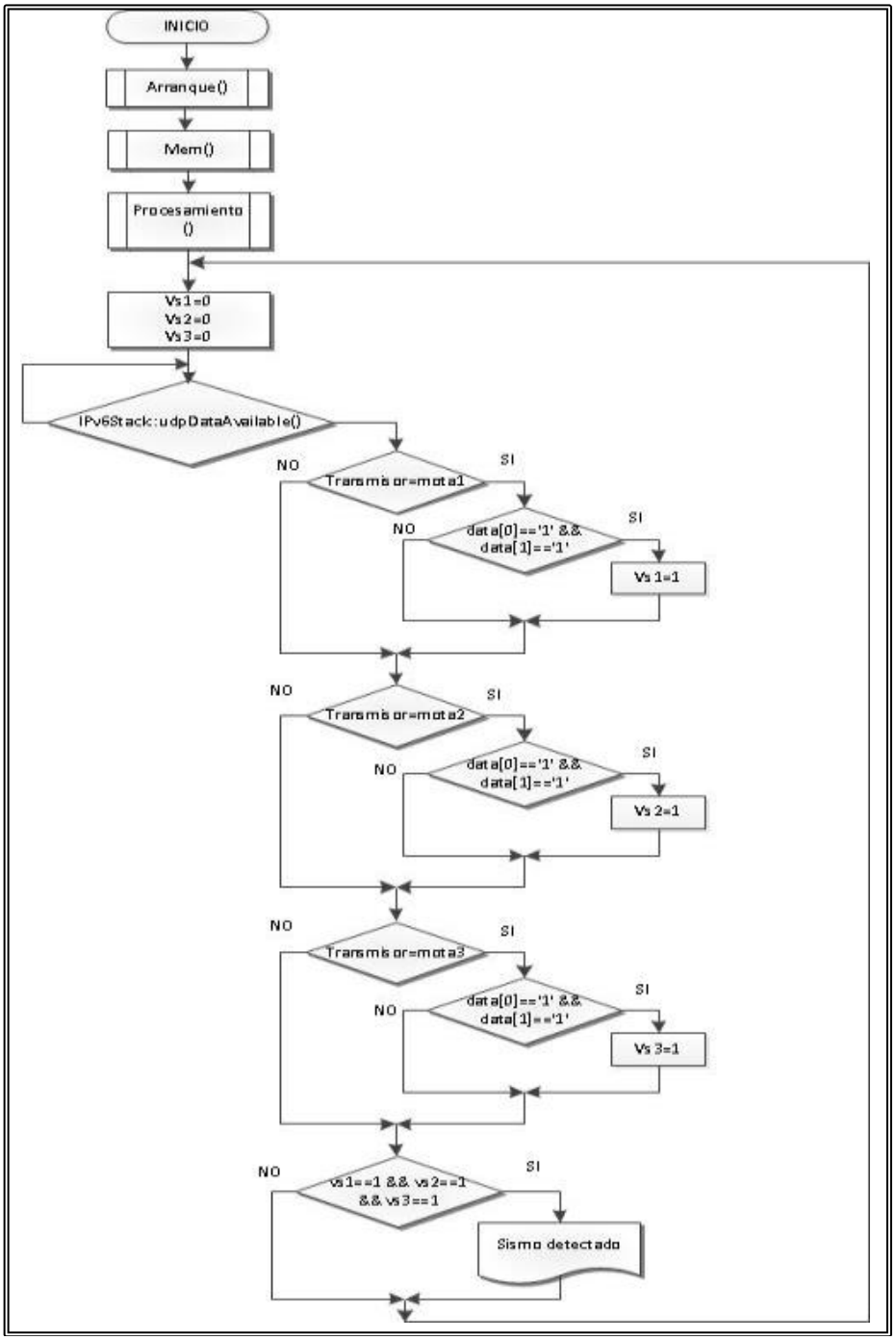


Figura 19-2. Funcionamiento de la mota coordinador
Fuente: PALATE, Byron. 2016

1.13 Configuración de Riot

Para la configuración de Riot al igual que la configuración de Contiki, debemos clonar o descargar el repositorio Riot desde la página de GitHub como se muestra en la figura 20-2.

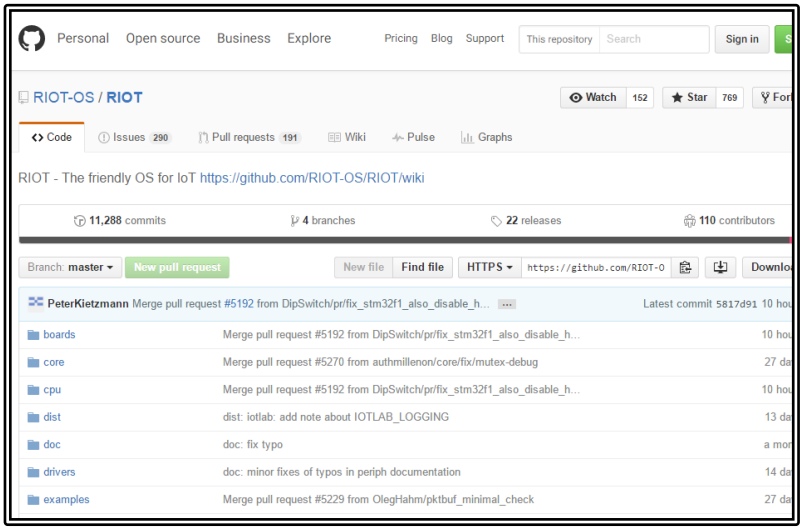


Figura 20-2. Repositorio Riot
Fuente: https://github.com/RIOT-OS/RIOT

Al igual que Contiki, Riot posee archivos de configuración como librerías, ejemplos, Makefiles, también se puede observar en la figura 21-2, que en la carpeta boards contiene todos los sistemas embebidos donde se puede cargar Riot, en los cuales se encuentra Arduino-mega2560

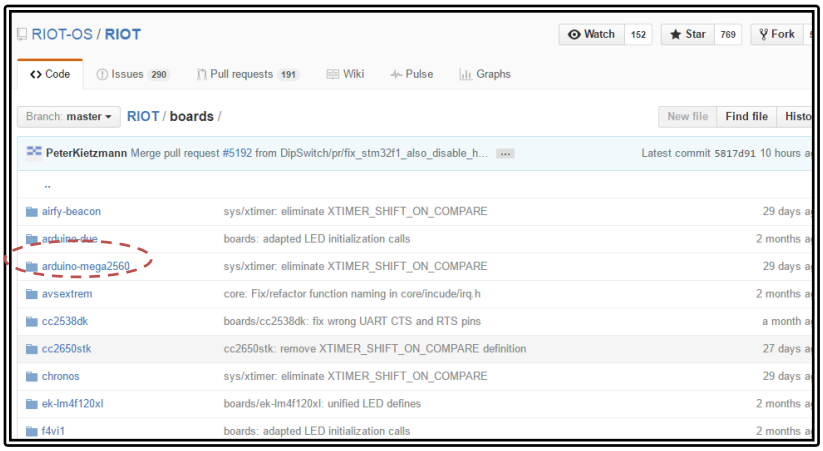


Figura 21-2. Board Arduino Mega 2560 en Riot
Fuente: https://github.com/RIOT-OS/RIOT

Debido a que la arquitectura de la placa Arduino mega es diferente a la arquitectura de las demás placas que pueden correr RIOT, esta plataforma implementa librerías para poder trasformar la configuración Arduino en configuración que Riot lo pueda interpretar, como por

ejemplo la trasformación de los PINS Arduino por GPIOS de Riot. Además también trasforma la función loop() en la función Main() que es la principal de Riot.

A continuación podemos apreciar en la figura 22-2, la librería que transforma los pines de Arduino, las librerías se encuentran en lugares como ~/arduino-mega2560/include.

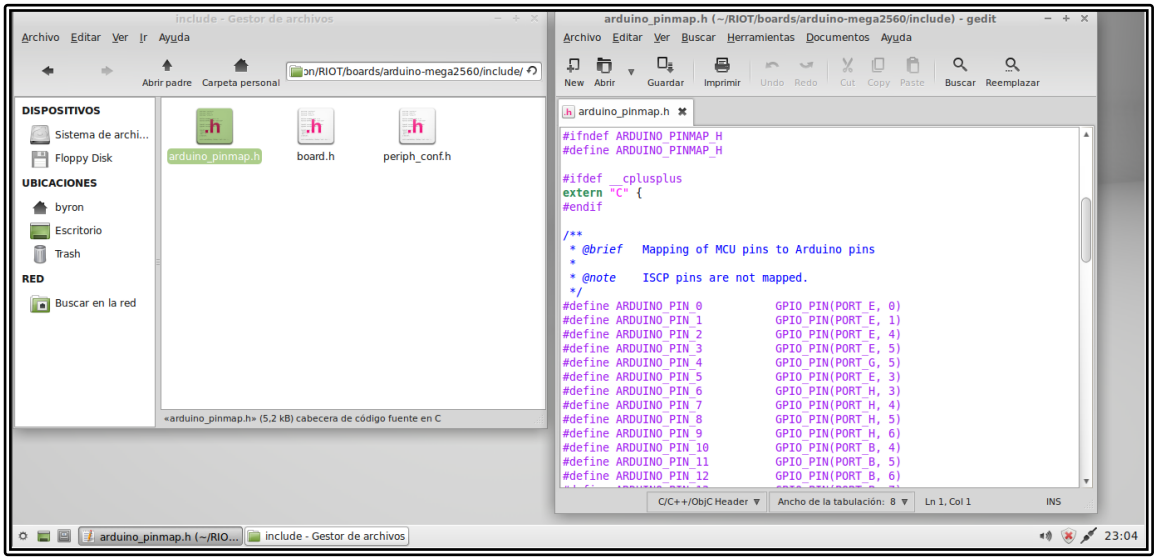


Figura 22-2. Librería pinmap de Riot
Fuente: PALATE, Byron. 2016

Para poder ejecutar RIOT es muy necesario del archivo MAKEFILE, debido a que en este se configuran los parámetros obligatorios de la plataforma para que pueda compilar sobre arduino.

1.13.1 Archivo de configuración

1.13.1.1 Makefile

Es un archivo de configuración que nos permite establecer parámetros necesarios para qué Riot nos permita ejecutar el archivo sobre las distintas placas, la configuración de este archivo es diferente para cada sistema embebido.
Las líneas de configuración más importantes de este archivo son 5:

- APPLICATION = default (Este es el nombre del proyecto y del ejecutable que se creara, por lo tanto en este caso se llamara mota y coordinador)

- BOARD ?= native (Esta es la placa donde se compilara el proyecto, por defecto viene configurado como native, pero en nuestro caso colocaremos arduino-mega2560 debido a que es nuestra placa de compilación)

- RIOTBASE ?= $(CURDIR)/../.. (Aquí colocaremos la ruta absoluta del directorio RIOT en nuestro sistema)

- USEMODULE += arduino (hace uso del módulo arduino para poder compilar Riot)

- include $(RIOTBASE)/Makefile.include

Creamos una carpeta y copiamos los archivos Makefile y Readme del directorio arduino_hello_world que se encuentra en /examples, mediante el editor nano de linux editamos el archivo Makefile copiado y establecemos los parámetros mencionados anteriormente, dichos pasos se encuentran ilustrados en la figura 23-2.

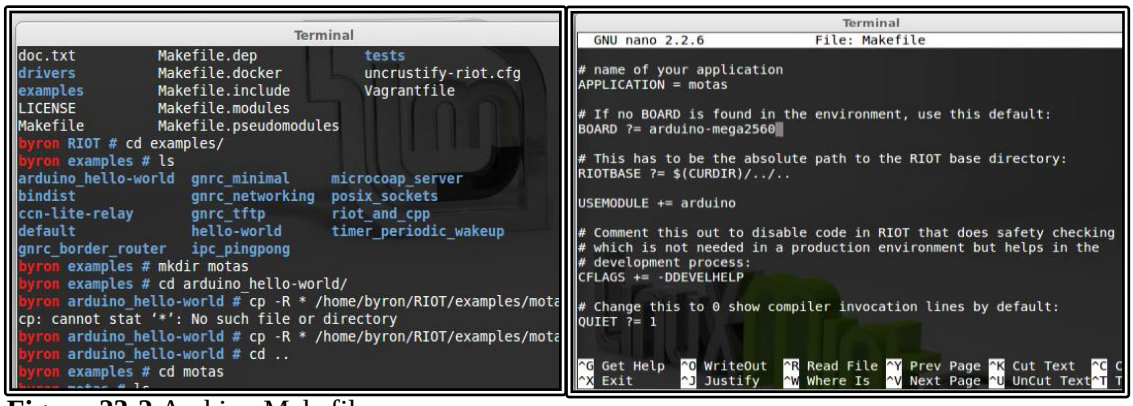


Figura 23-2. Archivo Makefile
Fuente: PALATE, Byron. 2016

Para que Riot pueda ejecutar sus librerías en Arduino necesita su sketch con una extensión .sketch, para lo cual copiamos el sketch realizado en Arduino pero con una extensión .sketch, esto en la carpeta donde se encuentra el makefile.

El archivo Makefile mediante el parámetro USEMODULE+=arduino, se encarga de llamar a la librería arduino_board.h que se encuentra en ~/board/arduino-mega2560/include/, además también llama a otras librerías como arduino_pinmap.h. Riot por sí solo no puede interpretar los comandos 6LowPan de Arduino para ello es necesario establecer un módulo extra el cual se encarga de llamar a librerías y establecer los parámetros necesario para el uso de 6LowPan en arduino, dicho modulo se denomina USEMODULE+=sixlowpan.

1.13.2 Programación en Riot

Las direcciones ip de las motas y el coordinador son las mismas que se estableció en el escenario de Contiki, esto debido a que se utilizó los mismos dispositivos y las direcciones ip están asociados a las direcciones físicas de cada uno de los módulos xbee.

Toda la configuración de su Stack IPv6 de Riot se encarga el módulo USEMODULE+=sixlowpan, conjuntamente con USEMODULE+=arduino el cual mediante sus librerías ayuda a interpretar la programación arduino y transformarla a Riot.

En la figura 24-2 se muestra el contenido del archivo Makefile, donde se encuentran los módulos sixlowpan y arduino.

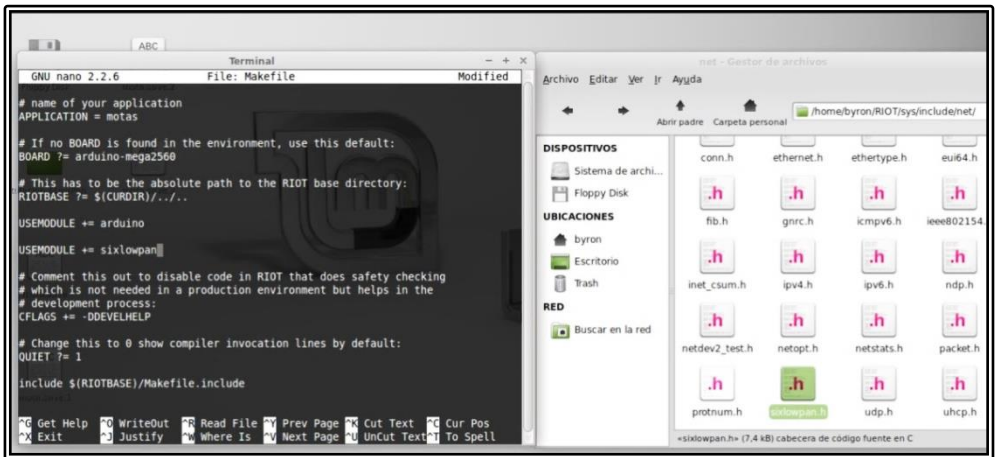


Figura 24-2.Módulosixlowpan
Fuente: PALATE, Byron. 2016

1.13.2.1 Consumo de memoria

Riot necesita crear sus archivos binarios para poder ejecutarse sobre sus sistemas embebidos, esto se logra mediante el comando make flash, dicho comando inicia el Makefile y mediante sus librerías generan los archivos binarios.

También aparece un archivo temporal denominado "_sketches.cpp", el cual el script copia secuencias de comandos para la unión con arduino (pre.snip y post.snip), junto con el contenido de todos los archivos * .sketch contenidos en la carpeta de la aplicación .

Al generar los archivos binarios de Riot, está ya nos permite visualizar el total de memoria que consumió el programa, este valor se nos despliega por categorías como son memoria ocupada en texto, datos, y al final su valor en número decimal y hexadecimal.

1.13.2.2 Velocidad de procesamiento

Riot también posee librerías como xtimer, rtimer, que pueden ayudar a conocer la velocidad de procesamiento de cada programa al igual que se realizó en Contiki.

Después de generar los archivos binarios, se pude hacer uso del comando "make term", el cual le da una interfaz ("serie") para la mota, de manera que se puede ver su salida y enviar algunos comandos a la misma.

De esta manera podremos ver los datos de salida configurados en la mota y poder conocer la velocidad de procesamiento, esto debemos hacerlo tanto para cada mota como para el nodo coordinador debido a que sus velocidad de procesamiento no serán los mismos.

1.13.2.3 Consumo energético

Debido a que los módulos de comunicación xbee se los configuro en el escenario de Contiki, estos ya no necesitan ser configurados nuevamente para Riot.

Al igual que en el escenario de Contiki, la medición del consumo energético se lo realizara de forma manual con un multímetro digital y se evaluara de dos formas distintas.

La primera será el consumo tanto del funcionamiento de la mota por sí sola, es decir solo el sistema embebido sin ningún sensor o módulo de comunicación, y también se evaluara el consumo energético de toda la mota en conjunto es decir con su respectivo sensor y módulo de comunicación.

1.13.3 Proceso de Arranque del archivo Makefile

Makefile sigue un proceso para su ejecución, el cual carga en memoria la configuración realizada en su archivo, además verifica si se encuentran disponibles las librerías necesarias para la creación de sus archivos binarios, dicho procedimiento se muestra en la figura 25-2.

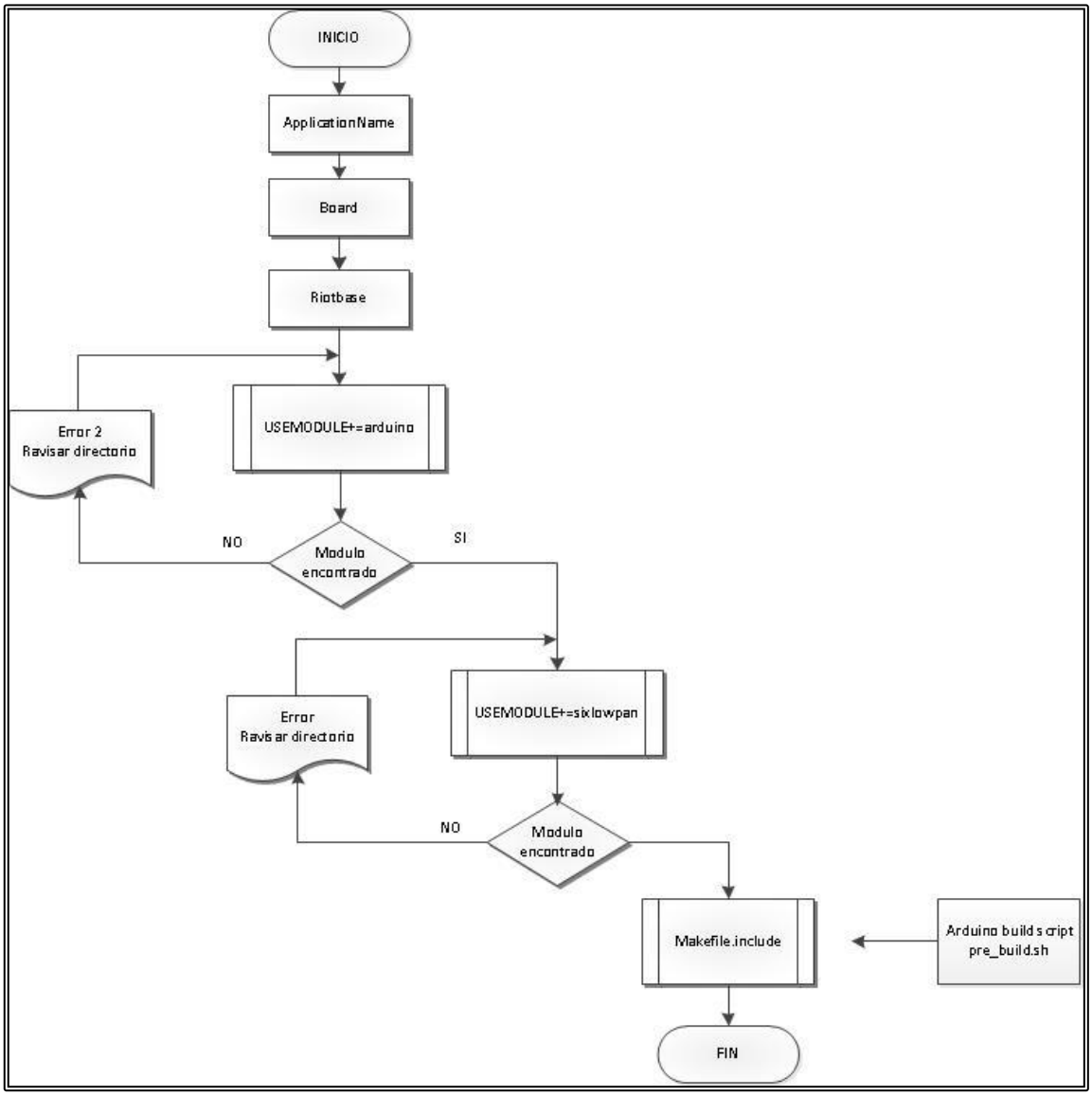


Figura 25-2. Arranque Makefile
Fuente: PALATE, Byron. 2016

1.13.4 Funcionamiento de las motas sensor

Después de generar los archivos binarios, el código se encuentra listo para ser ejecutado, dichas ejecución se representan en la figura 26-2.

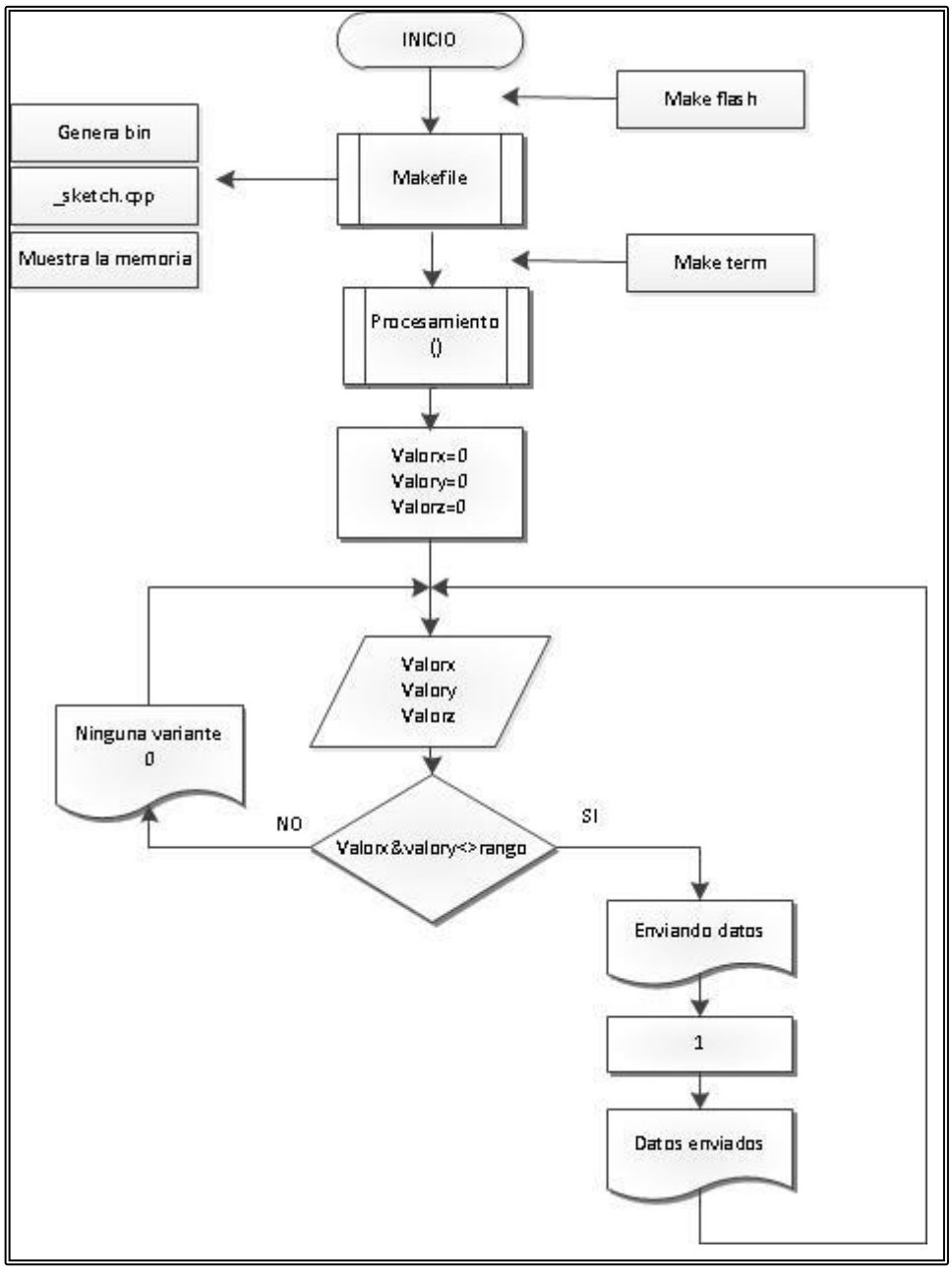


Figura 26-2. Funcionamiento mota sensor Riot
Fuente: PALATE, Byron. 2016

1.13.5 Funcionamiento de la mota coordinador

El funcionamiento de la mota coordinador es muy diferente a la mota sensor, esto debido a que el coordinador debe estar pendiente de los paquetes recibidos a través de su radio, dicho funcionamiento se muestra en la figura 27-2.

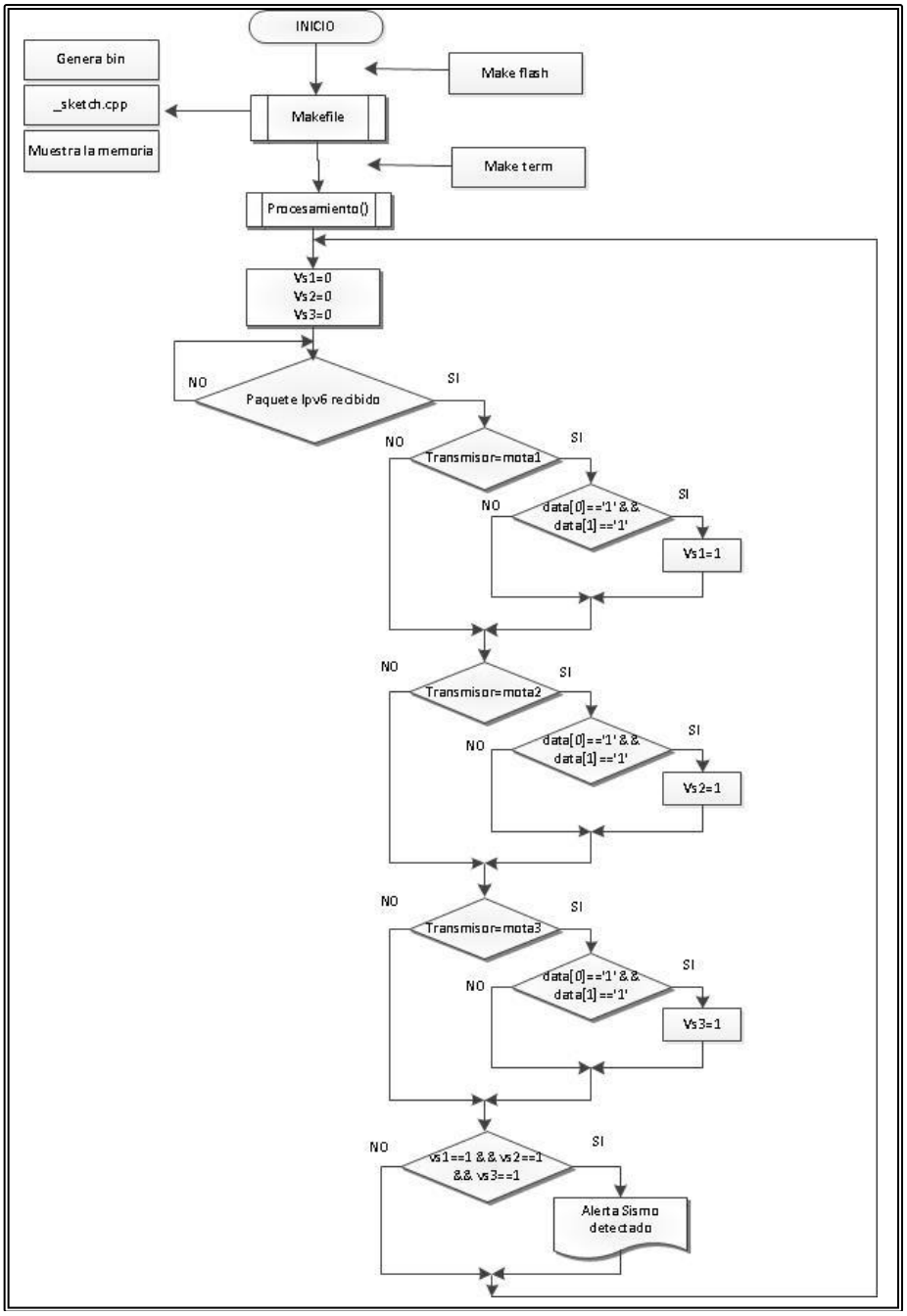


Figura 27-2. Funcionamiento mota coordinador Riot
Fuente: PALATE, Byron. 2016

1.14 Diseño de la fuente de vibración sísmica

A continuación se procede a detallar los procesos necesarios para la elaboración de la fuente de vibración para el presente proyecto.

1.14.1 Calculo de Amplitudes de vibración sísmica

La magnitud Richter se puede calcular gráficamente mediante una tabla con valores establecidos por los estudios de Richter, para la cual es muy fundamental la estimación de tiempo entre las ondas primarias y secundarias (Δt), debido a que este tiempo nunca suele ser el mismo en cada sismo, para el presente proyecto se tomaron valores estimados.

En nuestro caso se pretende simular un sismo establecido de 3, 5 y 7 grados en la escala de Richter correspondientes a los niveles menor, moderado y fuerte respectivamente, para lo cual como ya se dijo anteriormente debemos establecer valores estimados de Δt y encontrar la amplitud de las ondas S para poder graficarlas y comprobar su resultado, para ello partimos de la fórmula de magnitud sísmica establecida por Richter:

$$M = log_{10}(A) + 3log_{10}(8\Delta t) - 2.92$$

$$M = log_{10}\left(\frac{A*\Delta t^3}{1.62}\right)$$

$$A = \frac{10^M * 1.62}{\Delta t^3}$$

Dónde:

A = amplitud de las ondas S en milímetros.

Δt = tiempo en segundos desde el inicio de las ondas P (Primarias) al de las ondas S (Secundarias). (Valor estimado)

M = magnitud arbitraria pero constante a terremotos que liberan la misma cantidad de energía.

En la tabla 7-2 se puede apreciar las diferentes amplitudes obtenidas con respecto a las magnitudes establecidas. Estos valores pueden ser graficados en cartas sismográficas de Richter las cuales son utilizadas para aproximaciones en casos de estudios.

Tabla 7-2: Amplitud de magnitudes sísmicas

Magnitud	Δt(s)	Amplitud(mm)
3	18	0.28
5	18	27.78
6.5	18	878.41

Realizado por: PALATE, Byron. 2016.
Fuente: PALATE, Byron. 2016.

En la figura 28.2, se puede comprobar la amplitud obtenida mediante las formulas, para una magnitud de 3 grados.

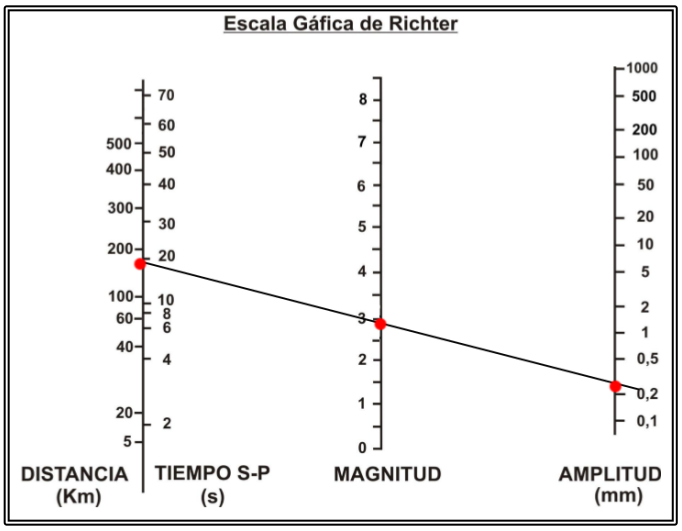


Figura 28-2. Carta sismográfica para sismo de magnitud 3
Fuente: PALATE, Byron. 2016

En la figura 29.2, se puede comprobar la amplitud obtenida mediante las formulas, para una magnitud de 5 grados.

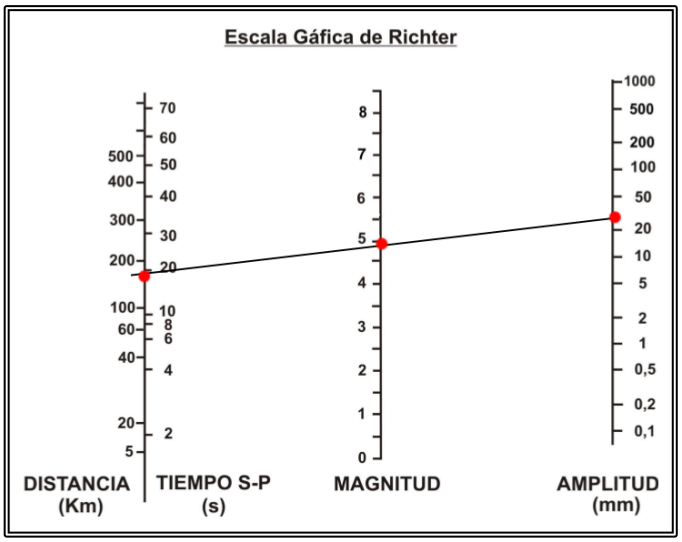


Figura 29-2. Carta sismográfica para sismo de magnitud 5
Fuente: PALATE, Byron. 2016

En la figura 30.2, se puede comprobar la amplitud obtenida mediante las formulas, para una magnitud de 6.8 grados.

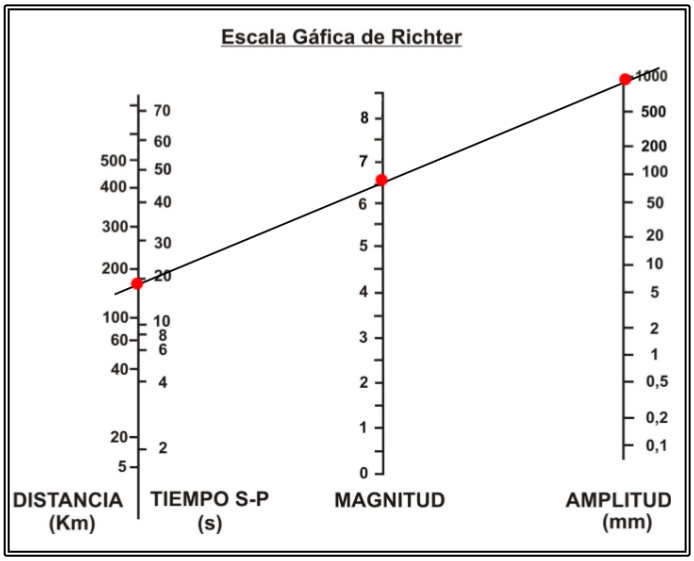


Figura 30-2. Carta sismográfica para sismo de magnitud 6.5
Fuente: PALATE, Byron. 2016

1.14.2 Configuración de sensores para las amplitudes

Una vez obtenidas las amplitudes de los sismos, procedemos a establecer los rangos en los sensores para que puedan detectar el nivel de sismo.

Los acelerómetros ADXL335 debido a que es un sensor analógico, entrega como respuesta valores entre 0 y 1024, los cuales indican el nivel de variación de sus ejes, es así que 0 no tendría variación y 1024 estarían en su variación máxima. Tomando en cuenta estos valores conjuntamente con la escala sísmica de Richter, para el presente trabajo se configuraran rangos de valores en los acelerómetros como se aprecia en la tabla 8-2, los cuales puedan interpretar la magnitud del sismo a la cual es sometida la maqueta de nuestro puente. Cabe recalcar que estos valores solamente son de prueba, es decir que estos rangos no están definidos en ningún otro sitio.

Tabla 8-2: Rangos de los ejes del acelerómetro

Magnitud	Descripción	Amplitud	Rango
<2,0	Micro	0.03	0-102
2,0-2,9	Menor	0.03-0.22	103-204
3,0-3,9		0.28-2.21	205-306
4,0-4,9	Ligero	2.78-22.06	307-408
5,0-5,9	Moderado	27.78-220.65	409-510
6,0-6,9	Fuerte	277.78-2206.47	511-612
7,0-7,9	Mayor	2777.78-22064.67	613-714
8,0-8,9	Gran	27777.78-220646.73	715-816
9,0-9,9		277777.78-2206467.32	817-918
10,0>	Épico	2777777.78+	919-1024

Realizado por: PALATE, Byron. 2016.
Fuente: PALATE, Byron. 2016.

1.14.3 Diseño de la fuente de vibración sísmica

Debido a que nuestro proyecto necesariamente necesita una fuente que permita generar vibraciones sobre el puente, se optó por el uso y configuración de servomotores de 9.4Kgcm de torque como se muestra en la figura 31-2, el cual nos permitirá soportar el peso que ejerce el puente junto a su base sobre los servomotores, para poder generar la vibración necesaria.

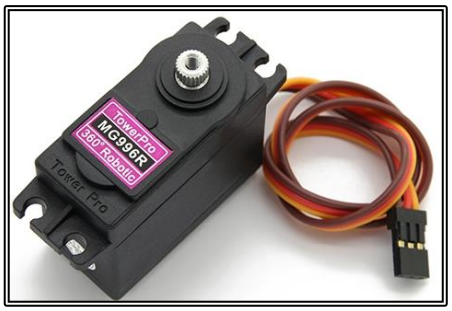


Figura 31-2. Servomotor MG996R
Fuente: https://www.hobbyking.com/mobile/products.asp?idparentcat=1707

Estos motores nos ofrecen movimientos de 0 a 180 grados y viceversa, para nuestro caso de estudio nos ayudaran a simular vibraciones sobre un eje, debido a que este trabajo de titulación no se centra en la investigación y análisis sísmico como tal, dicha fuente de vibración es una buena opción para poder constatar el funcionamiento de nuestras motas sensores. Pero para trabajos futuros que se centren en el análisis sísmico se recomienda trabajar con sismógrafos y fuentes de variación en sus 3 ejes.

Estos motores se instalaron en una base rectangular la cual posee un peso de 5Kg para que pueda permanecer fija sobre el suelo, dicha base se puede apreciar en la figura 32-2.

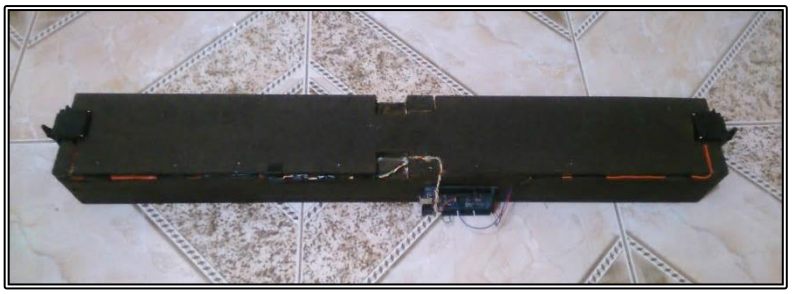


Figura 32-2. Fuente sismográfica
Fuente: PALATE, Byron. 2016

Dichos servomotores deben generan una fuerza sobre la base del puente, la cual será calculada a continuación, para ellos es necesario conocer la distancia del brazo del servomotor y el torque del mismo como se puede observar en la figura 33-2, dicho valor varía según el modelo del motor, en nuestro caso el modelo MG996R ofrece un torque de 9.4Kg*cm

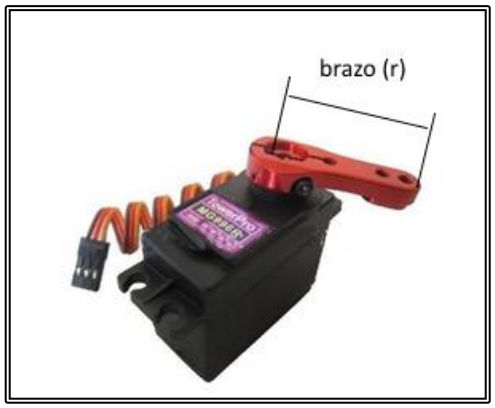


Figura 33-2. Distancia del brazo del motor
Fuente: PALATE, Byron. 2016

$\tau = F*r$ [N m]

F = fuerza

r = distancia brazo

$\tau = 9.4$ Kgcm

$\tau = 0.094$ Kgm

$\tau = 0.922$ Nm

r = 1.5 cm

r = 0.015 m

$$F = \frac{0.922\ Nm}{0.015\ m}$$

F = 61.47 N

$$F = 61.47\ [\frac{Kg*m}{S^2}]$$

CAPÍTULO III

3. MARCO DE PRUEBAS Y RESULTADOS

1.15 Introducción

Después de haber realizado el diseño de la red, en el presente capitulo se detallará la constitución del ensamblaje así como también las pruebas de rendimiento realizadas de toda la red operativa.

1.16 Ensamblaje de Equipos

1.16.1 Composición de las motas

Mota sensor 1, 2, 3.- Estas motas son las encargadas de censar las variaciones en los ejes del sensor ADXL335 y transmitirlas de forma inalámbrica si sus datos se encuentran fuera de un rango establecido, por lo cual está formado por un sistema embebido (Arduino Mega 2560 R3), un módulo Xbee S1, un Shield Xbee y un sensor acelerómetro.

En la tabla 1-3, se muestra la conexión de los pines del sensor sobre la placa arduino mega 2560.

Tabla 1-3: Conexión sensor adxl335

Sensor	Pines	Arduino Mega
ADXL335	Vcc	Pin 12
	Gnd	Pin 8
	Datos x	Pin 11
	Datos y	Pin 10
	Datos z	Pin 9

Realizado por: PALATE, Byron, 2015.
Fuente: PALATE, Byron. 2016.

Para un mejor entendimiento sobre la conexión del sensor y los módulos, se puede observar la figura 1-3, la cual muestra el sensor y su Xbee S1 conectados en la placa Arduino.

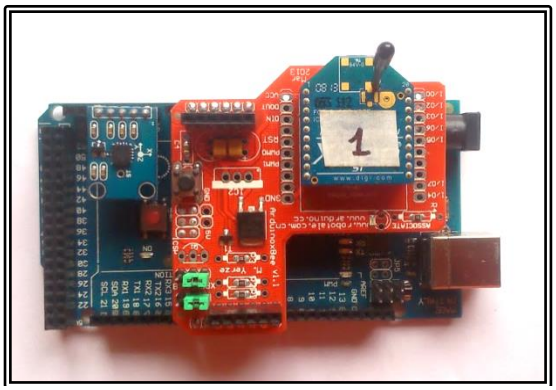


Figura 1-3. Mota sensor 1
Fuente: PALATE, Byron. 2016

Mota coordinador.-Es la mota más importante, debido a que realiza todo el trabajo de procesamiento de la información que fue enviada por las motas sensores.
Está conformada por un sistema embebido (Arduino Mega 2560 R3), un módulo Xbee S1 y un Shield Xbee, todos estos componentes se encuentran ilustrados en la figura 2-3.

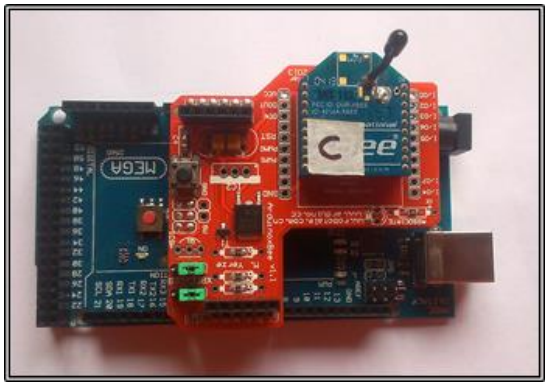


Figura 2-3. Mota coordinador
Fuente: PALATE, Byron. 2016

1.17 Protección de las motas

Debido a que estos sistemas WSN fueron desarrollados para estar expuestos a la intemperie, y así poder sensar su entorno, es necesario la protección de cada una de las motas, es así que se optó por protegerlas con cajas de acrílico que cuenten con un espacio adecuado para que la mota quede segura y libre de cualquier fallo como se puede observar en la figura 3-3.

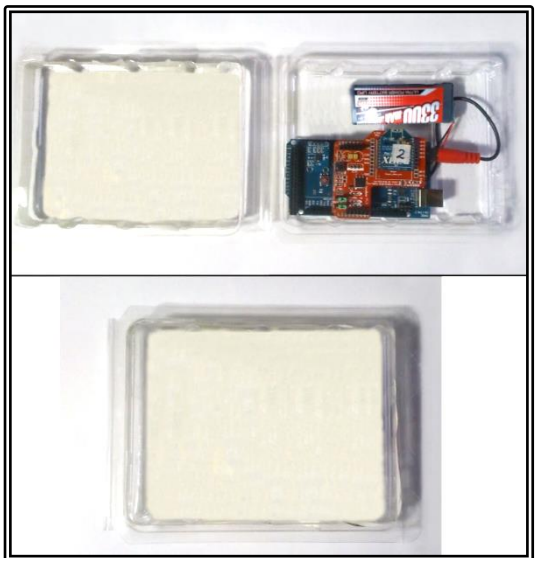


Figura 3-3. Protección de las motas
Fuente: PALATE, Byron. 2016

1.18 Diseño Experimental de la topología

En esta sección se detallará la instalación de toda la red 6LowPan, así como también la variable de vibración sobre el puente.

1.18.1 Instalación Red 6LoWPAN

Una vez que los equipos se encuentran con la protección adecuada, se procede a colocarlos sobre una maqueta de la estructura de un puente y posteriormente colocarlos sobre una fuente de vibraciones externa para poder constatar su buen funcionamiento.

1.18.1.1 Instalación motas sensores 1, 2, 3

En la siguiente figura 4-3 se puede apreciar la instalación de las motas sensores en el escenario establecido de un puente.

Figura 4-3. Instalación de las motas
Fuente: PALATE, Byron. 2016

1.18.2 Instalación de la fuente de vibración sísmica

En la figura 5.3 se puede apreciar la fuente de vibración, dicha fuente se encuentra debajo de la base del puente, y simulara un sismo el cual será detectado por las motas sensores.

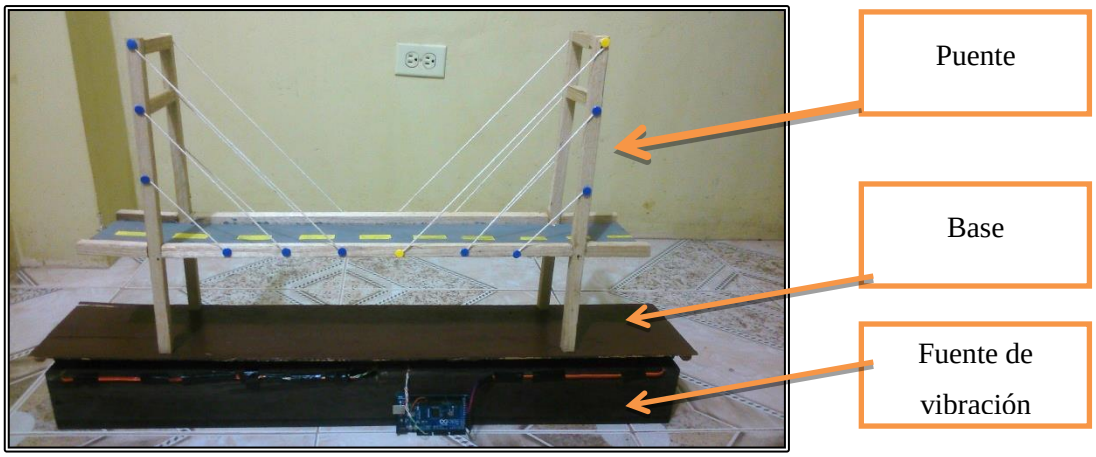


Figura 5-3. Fuente de vibración sísmica
Fuente: PALATE, Byron. 2016

1.19 Pruebas de Funcionamiento

Una vez que se realizó el ensamblado e instalación de las motas, es necesario comprobar su funcionamiento para poder evaluar los parámetros necesarios y así cumplir con los objetivos establecidos en el presente trabajo de titulación.

Para realizar las pruebas de funcionamiento, se tomó como referencia la duración del último terremoto del 16 de abril del presente año, el cual tuvo una duración de 42 segundos en el epicentro, y esta corta duración basto para destruir variarías ciudades y pueblos a su alrededor, debido a esto la fuente sísmica actuó alrededor de 1 minuto para cada magnitud.

Las muestras adquiridas en la simulación fueron alrededor de 300, dichas muestras fueron las mismas durante todo el tiempo establecido como se puede observar en la figura 6-3, en Contiki 4259 bytes libres, en Riot 4689 bytes libres, pudiendo llegar a la conclusión que el consumo de memoria y procesamiento no son variables dependientes del tiempo y magnitud.

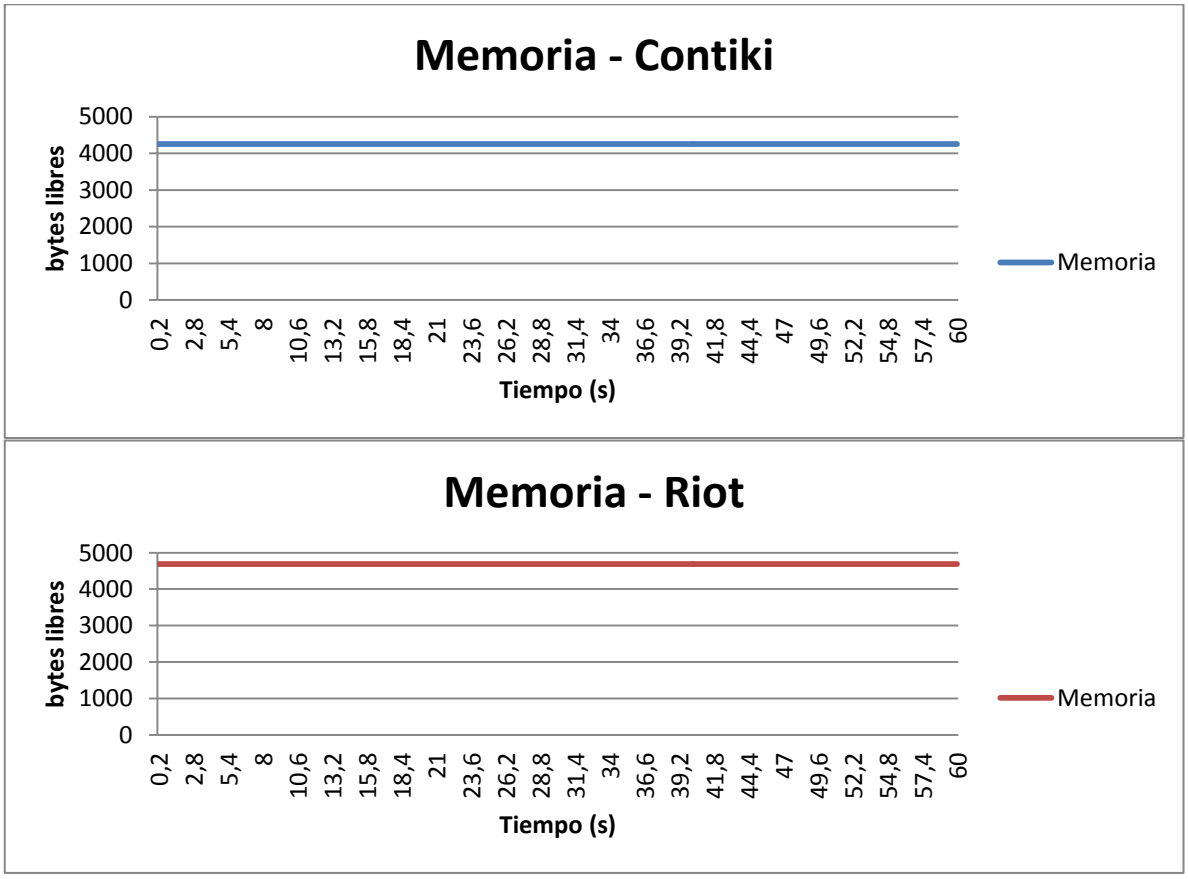


Figura 6-3. Muestras de consumo de memoria
Fuente: PALATE, Byron. 2016

Para aclarar mejor la figura 6-3, el consumo de memoria es el mismo en cualquier instante de tiempo y ante cualquier magnitud, esto debido a que las motas están programadas para sensar la magnitud y enviarlas, lo único que cambiará son los rangos censados por las motas, dichos rangos se explicaron en la sección 2.7.1.

1.19.1 Contiki-OS

1.19.1.1 Inicialización de los componentes

En la figura 7-3 se puede observar la inicialización de la mota coordinador.

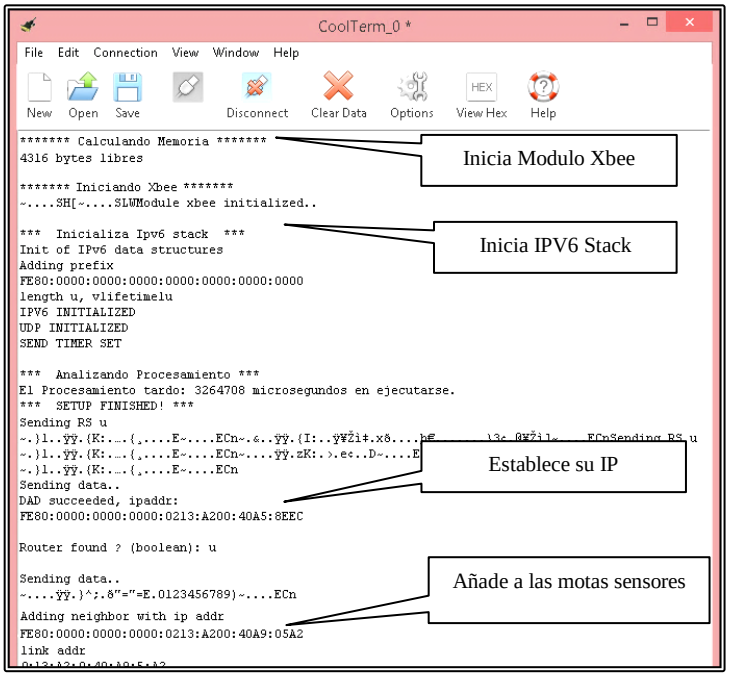


Figura 7-3. Inicialización del coordinador Contiki
Fuente: PALATE, Byron. 2016

En la figura 8-3 se puede observar la inicialización de la mota sensor 1.

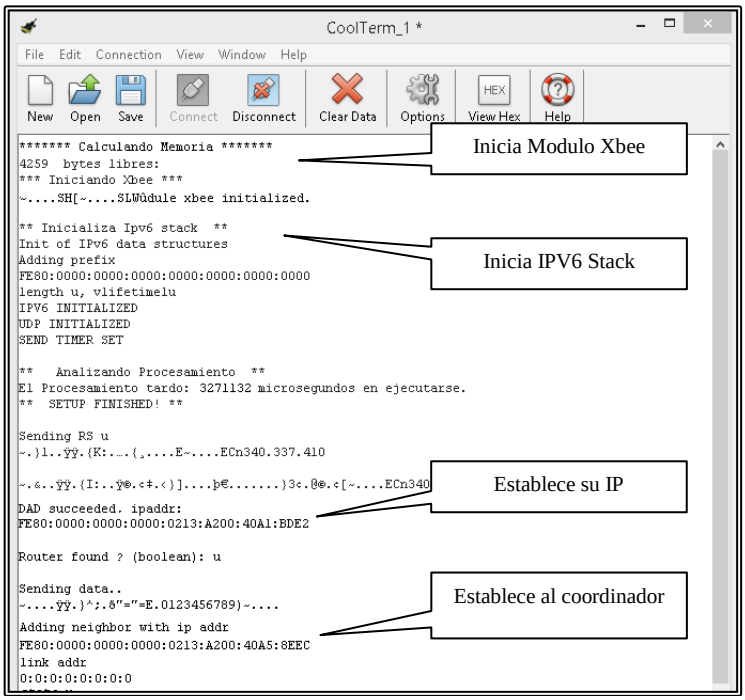


Figura 8-3. Inicialización mota 1 Contiki
Fuente: PALATE, Byron. 2016

En la figura 9-3 se puede observar la inicialización de la mota sensor 2.

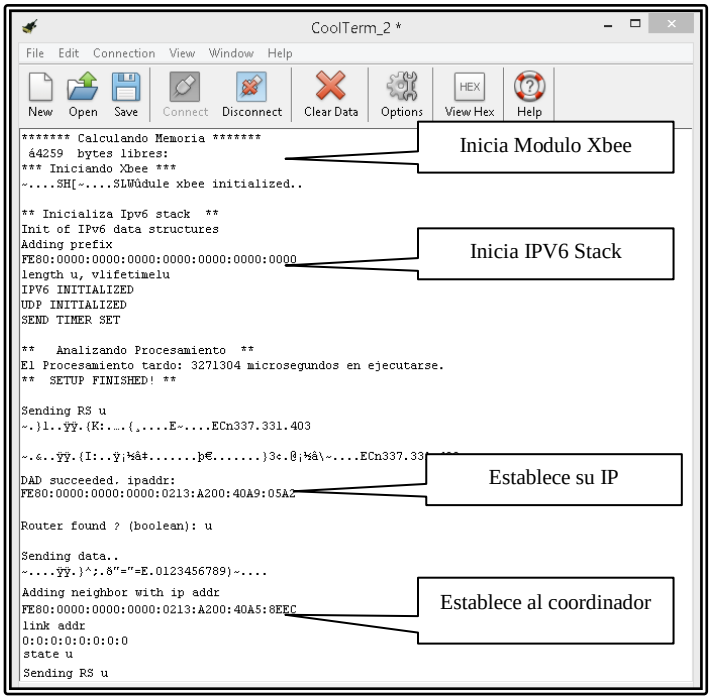


Figura 9-3. Inicialización mota 2 Contiki
Fuente: PALATE, Byron. 2016

En la figura 10-3 se puede observar la inicialización de la mota sensor 3.

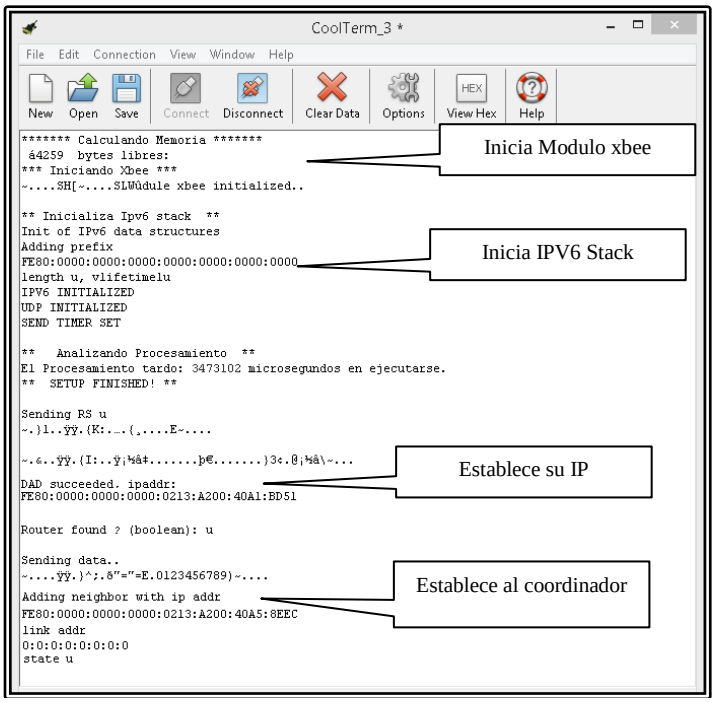


Figura 10-3. Inicialización mota 3 Contiki
Fuente: PALATE, Byron. 2016

1.19.1.2 Envió y recepción de datos

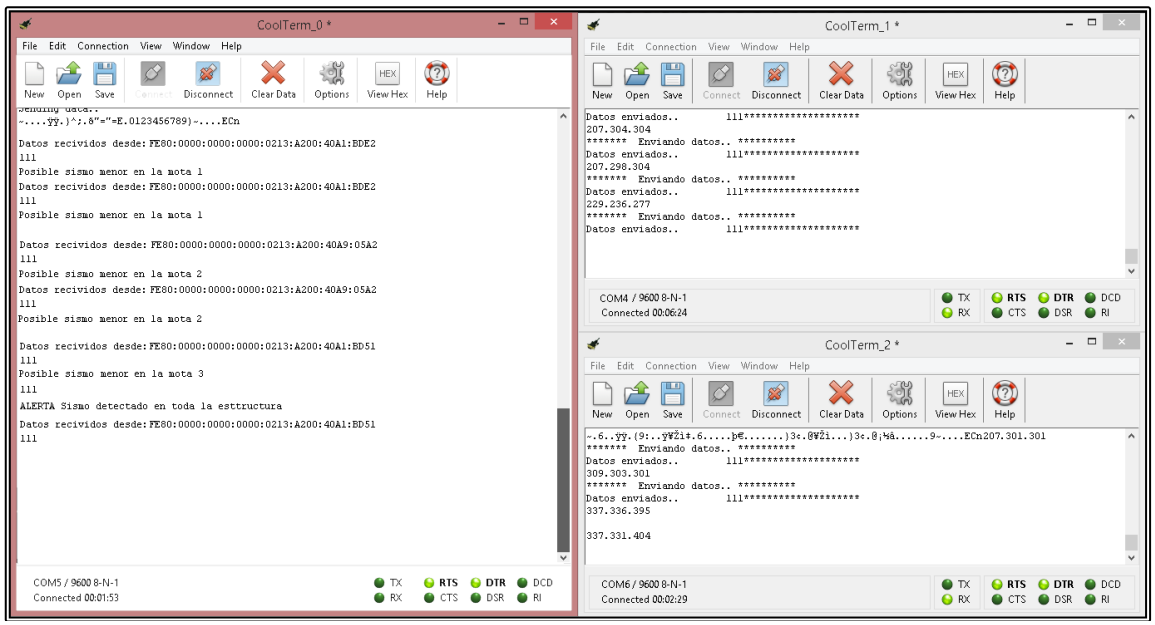


Figura 11-3. Envío y recepción de datos en Contiki
Fuente: PALATE, Byron. 2016

En la figura 11-3 podemos ver una variación mayor a 205 y menor a 306 en los ejes, correspondiente al rango menor del sismo establecidos en la sección 2.7.1.

El coordinador es el encardo de interpretar esa información y tomar decisiones para mostrar un mensaje sobre la mota que envió el dato o mostrar una alerta si el sismo es detectado en las tres motas al mismo tiempo.

1.19.2 Riot-OS

1.19.2.1 Inicialización de componentes

En la figura 12-3 se puede observar la inicialización de la mota coordinador.

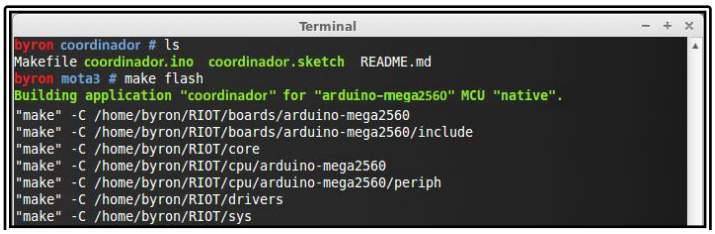


Figura 12-3. Inicialización del coordinador Riot
Fuente: PALATE, Byron. 2016

En la figura 13-3 se puede observar la inicialización de la mota sensor 1.

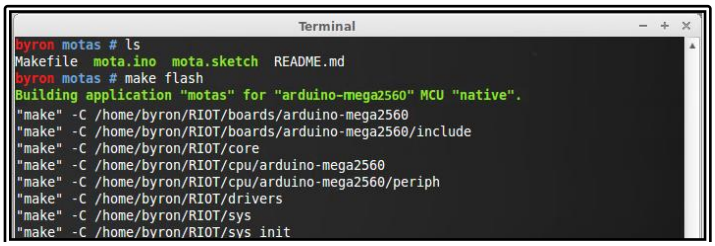


Figura 13-3. Inicialización mota 1 Riot
Fuente: PALATE, Byron. 2016

En la figura 14-3 se puede observar la inicialización de la mota sensor 2.

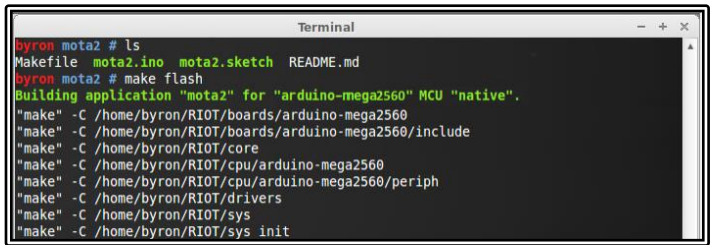


Figura 14-3. Inicialización mota 2 Riot
Fuente: PALATE, Byron. 2016

En la figura 15-3 se puede observar la inicialización de la mota sensor 3.

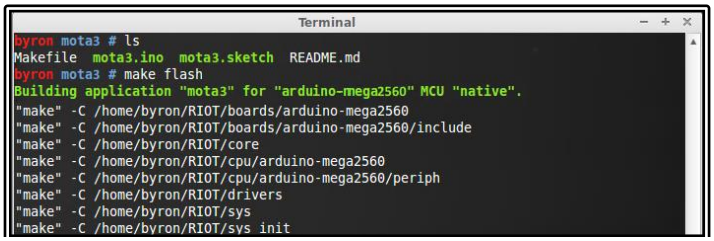


Figura 15-3. Inicialización mota 3 Riot
Fuente: PALATE, Byron. 2016

1.19.2.2 Envío y recepción de datos

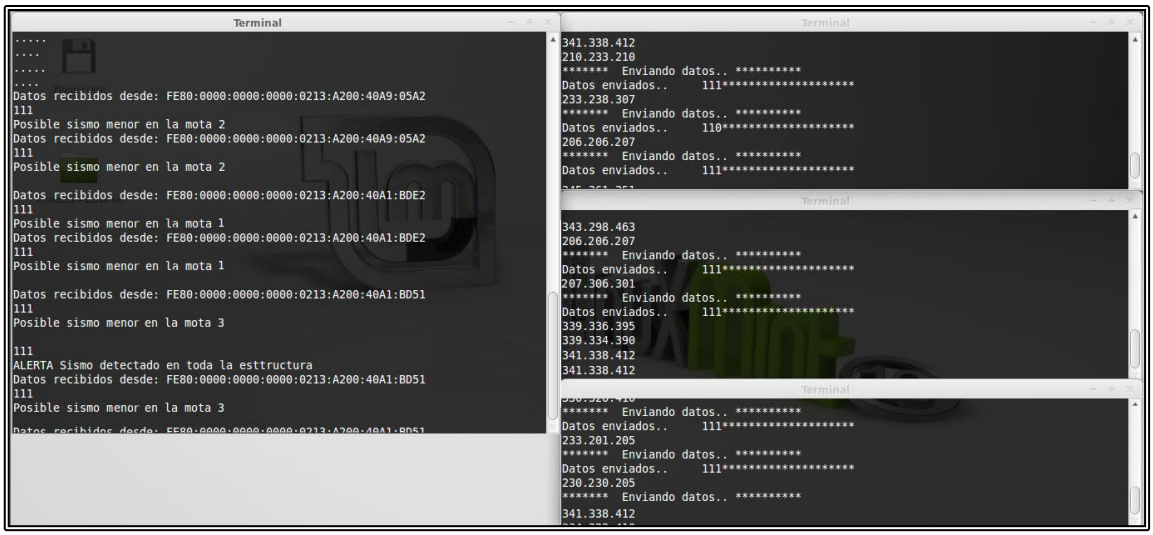


Figura 16-3. Envío y recepción de datos Riot
Fuente: PALATE, Byron. 2016

Al igual que en la configuración de Contiki, al existir una variación en los ejes del sensor y si se encuentran en los rangos establecido en la sección 2.7.1, las motas proceden a enviar información hacia el coordinador.

Como podemos apreciar en la figura 16-3, se produjo un sismo de magnitud medio, este fue detectado en las tres motas y así envían la alerta hacia el coordinador, el cual al detectar la alerta de las tres motas emite un mensaje de un posible sismo sobre la estructura.

1.20 Evaluación de los parámetros

En una red WSN con 6LowPan existen parámetros muy importantes como son memoria, procesamiento, consumo de batería, los cuales si consumen muchos recursos son un problema muy grande para estas redes, así lo detallan el RFC 6606 y RFC 4919, debido a que muchos de los dispositivos que emplean radios IEEE 802.15.4 se limitan en su velocidad de procesamiento, la memoria, y / o la disponibilidad de energía, (Kushalnagar N., Montenegro G. Schumacher C. 2007).

Basándonos en estos problemas se muestran los resultados obtenidos en cada una de las motas que intervienen en la red.

1.20.1 Memoria

1.20.1.1 Contiki-OS

En la figura 17-3 se puede observar el resultado de calcular la memoria en la mota coordinador configurado con la plataforma Contiki-OS, donde muestra que existen 4316 bytes libres de memoria de un total de 8192 bytes que dispone el Arduino Mega

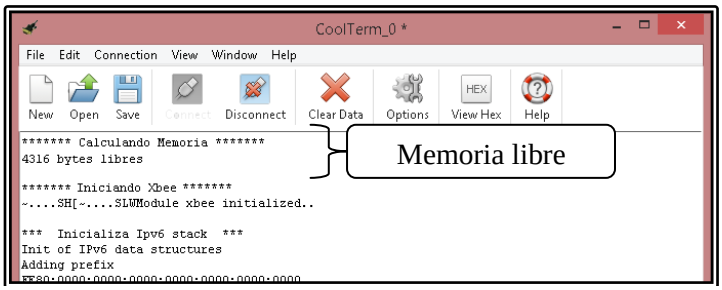


Figura 17-3. Memoria coordinador Contiki
Fuente: PALATE, Byron. 2016

Como se puede observar en las figuras 18-3, figura 19-3, y figura 20-3, la memoria libre de las motas sensores son las mismas, esto debido a que ejecutan el mismo programa para censar variaciones en los tres ejes.

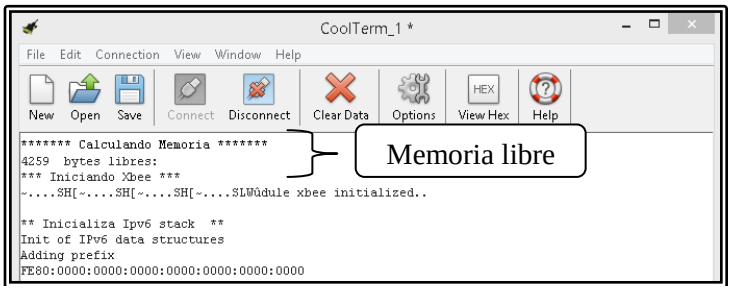


Figura 18-3. Memoria mota 1 Contiki
Fuente: PALATE, Byron. 2016

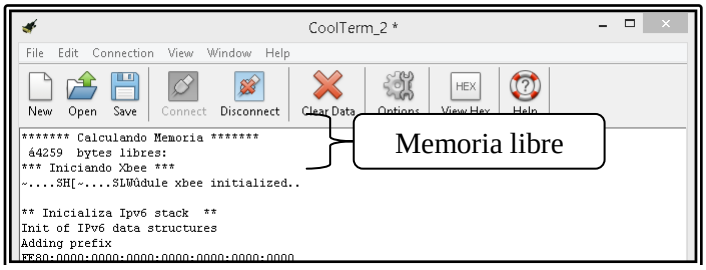


Figura 19-3. Memoria mota 2 Contiki
Fuente: PALATE, Byron. 2016

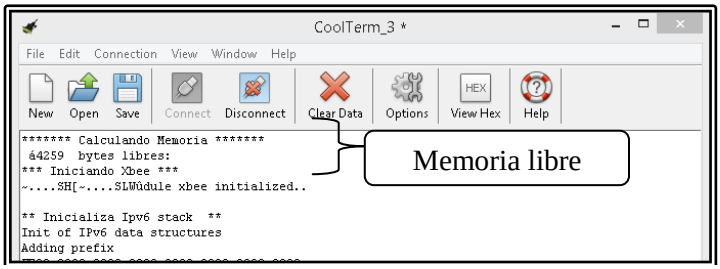


Figura 20-3. Memoria mota 3 Contiki
Fuente: PALATE, Byron. 2016

Las motas sensores poseen 4259 bytes de memoria libre con respecto a 8192 bytes de la memoria total que posee Arduino.

1.20.1.2 RIOT-OS

Después de generar los archivos binarios en Riot, make flash también nos muestra el estado de la memoria las cuales se los puede apreciar en la figura 21-3 que muestra la memoria consumida en la mota coordinador.

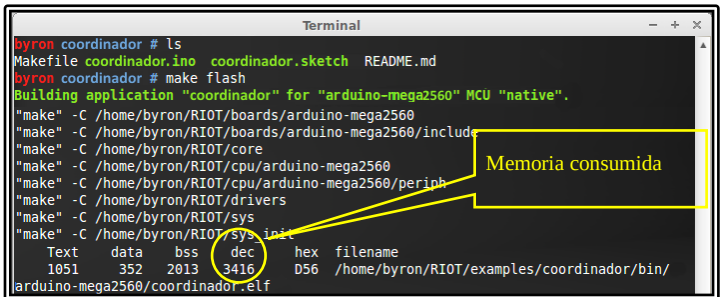


Figura 21-3. Memoria coordinador Riot
Fuente: PALATE, Byron. 2016

En las figuras 22-3 figura 23-3 y figura 24-3, las motas sensores poseen el mismo consumo de memoria, esto debido a que ejecutan el mismo código para poder censar las variaciones y transmitirlas

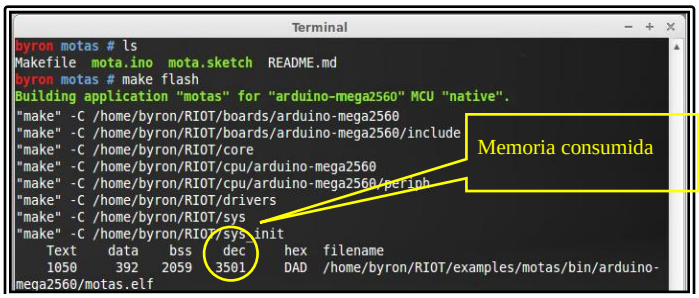


Figura 22-3. Memoria mota 1 Riot
Fuente: PALATE, Byron. 2016

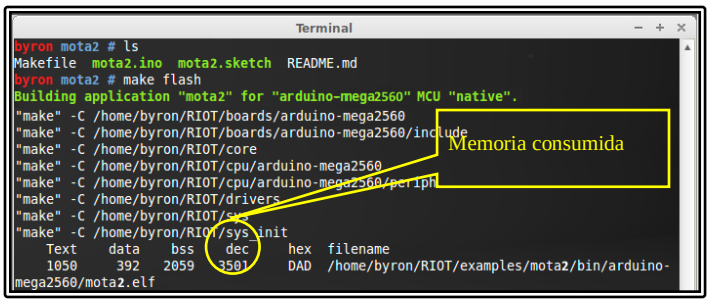


Figura 23-3. Memoria mota 2 Riot
Fuente: PALATE, Byron. 2016

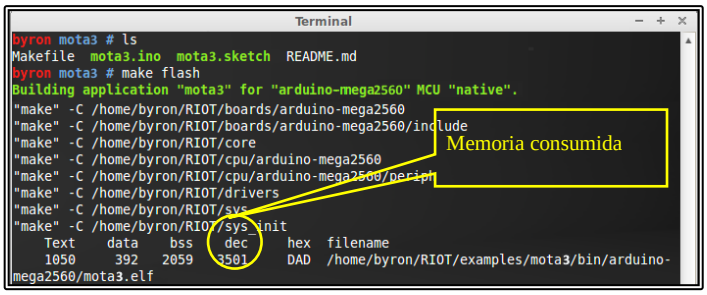


Figura 24-3. Memoria mota 3 Riot
Fuente: PALATE, Byron. 2016

El consumo de memoria de las motas sensores es de 3501 bytes, por lo tanto poseen una memoria libre de 4689 bytes de un total de 8192 bytes existentes.

1.20.2 Velocidad de Procesamiento

1.20.2.1 Contiki-OS

Como se puede observar en la figura 25-3, el tiempo de procesamiento del coordinador fue de 3264708 µs.

Figura 25-3. Procesamiento coordinador Contiki
Fuente: PALATE, Byron. 2016

La figura 26-3 muestra el tiempo de procesamiento de la mota sensor 1, la cual fue de: 3271132µs.

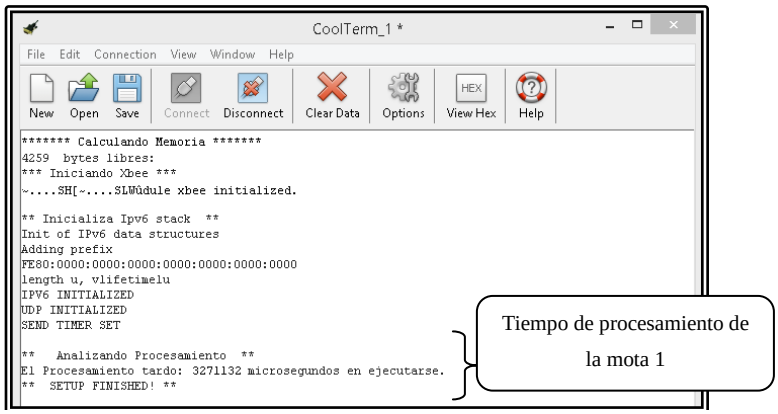


Figura 26-3. Procesamiento mota 1 Contiki
Fuente: PALATE, Byron. 2016

La figura 27-3 muestra el tiempo de procesamiento de la mota sensor 2, la cual fue de: 3271304µs

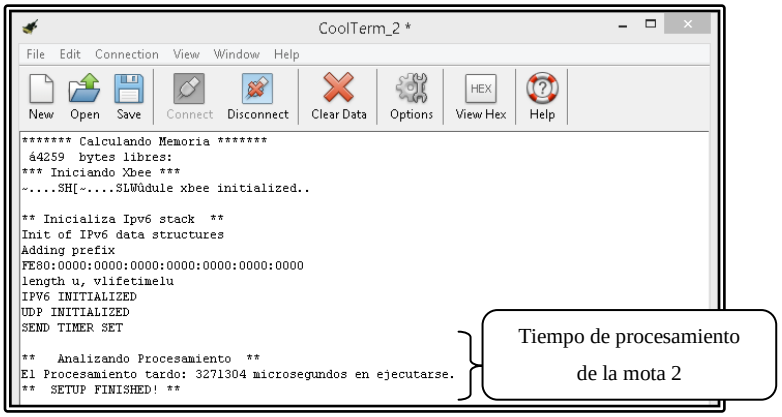


Figura 27-3. Procesamiento mota 2 Contiki
Fuente: PALATE, Byron. 2016

La figura 28-3 muestra el tiempo de procesamiento de la mota sensor 3, la cual fue de: 3473102µs

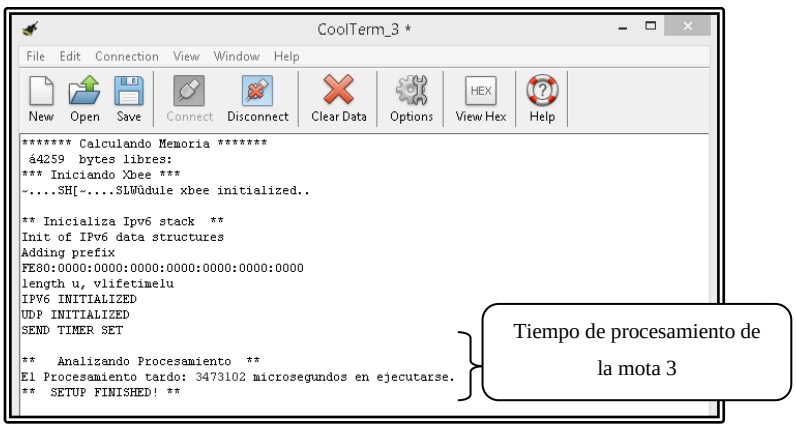


Figura 28-3. Procesamiento mota 3 Contiki
Fuente: PALATE, Byron. 2016

Como se puede observar en las figuras 26-3, figura 27-3 y figura 28-3,las motas poseen un tiempo de procesamiento casi idéntico entre ellas con un pequeño rango de variación, debido a esto podemos obtener un tiempo de procesamiento promedio de cada mota, dicho tiempo se encuentra en un valor estimado de 3320061µs.

1.20.2.2 RIOT-OS

Una vez obtenido los archivos binarios del programa, a continuación se puede obtener los datos de salida del mismo, mediante "make term" como se explicó en el apartado de procesamiento en el capítulo anterior. Los archivos de la carpeta bin que se generan al ejecutar el comando "make flash" son muy necesarios para este procedimiento, debido a que "make term" se basa en estos archivos para poder ejecutar el programa.

En la figura 29-3 se muestra el tiempo de procesamiento de la mota coordinador: 3051010 µs

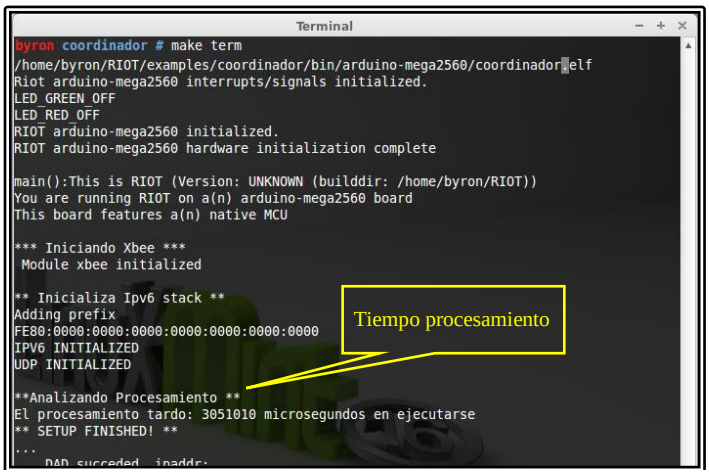


Figura 29-3. Procesamiento coordinador Riot
Fuente: PALATE, Byron. 2016

El tiempo de procesamiento de la mota sensor 1 es de 3152460 µs como se muestra en la figura 30-3.

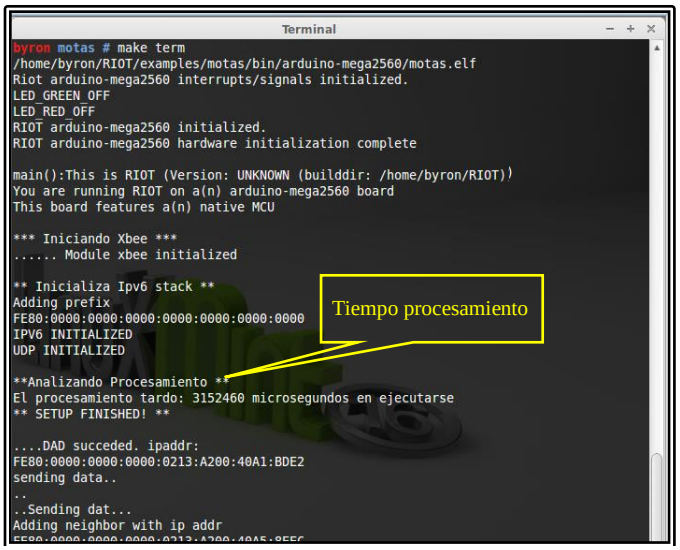


Figura 30-3. Procesamiento mota 1 Riot
Fuente: PALATE, Byron. 2016

El tiempo de procesamiento de la mota sensor 2 es de 3111550 µs como se muestra en la figura 31-3.

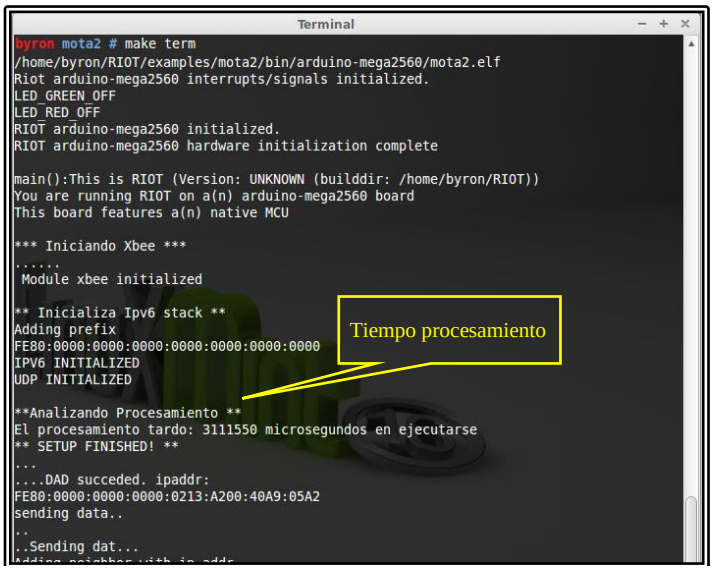


Figura 31-3. Procesamiento mota 2 Riot
Fuente: PALATE, Byron. 2016

Por último en la figura 32-3 se muestra el tiempo de procesamiento de la mota 3, la cual fue de: 3051010 µs

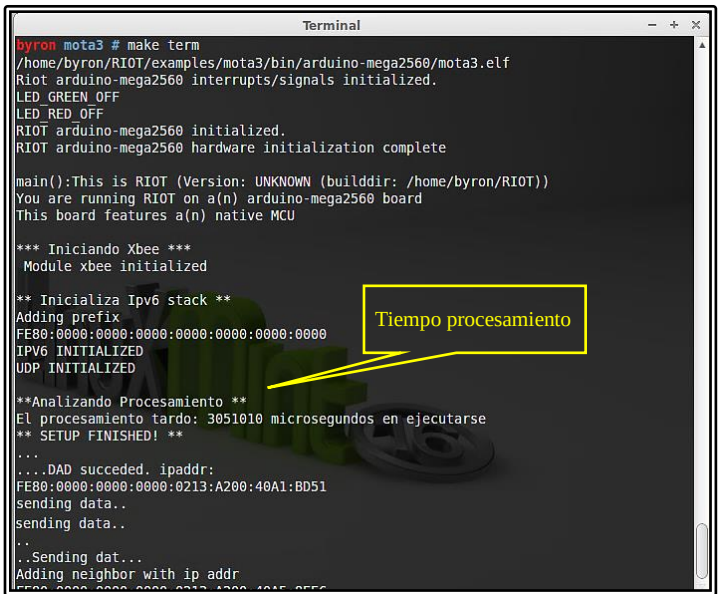


Figura 32-3. Procesamiento mota 3 Riot
Fuente: PALATE, Byron. 2016

Como se podía esperar, los tiempos de procesamientos no son los mismos, esto debido a que al igual que Contiki, el módulo de comunicación es el que produce estas variaciones de tiempos al iniciarse.

Como en el caso de Contiki los valores de procesamientos de las motas están en un rango similar entre ellas, por lo cual se podría obtener un procesamiento promedio solo entre ellas, el cual sería 3105006 µs.

1.20.3 Consumo Energético

Para medir el consumo real de la motas se utilizó un multímetro digital, en este caso un Proskit MT-1210 el cual posee un marco de precisión del 99% según sus características técnicas. Para alimentar a nuestra placa podríamos hacerlo desde el conector de 3.5mm o el USB, pero tendríamos que romper el cable para poder poner el multímetro en serie con la fuente y así poder medir el consumo real.

Debido a esto alimentaremos nuestro sistema por los pines Vin y Gnd que proporciona la placa, y para esto solo necesitaríamos un par de cables de protoboard, el esquema que se estableció se lo puede apreciar en la figura 32-3.

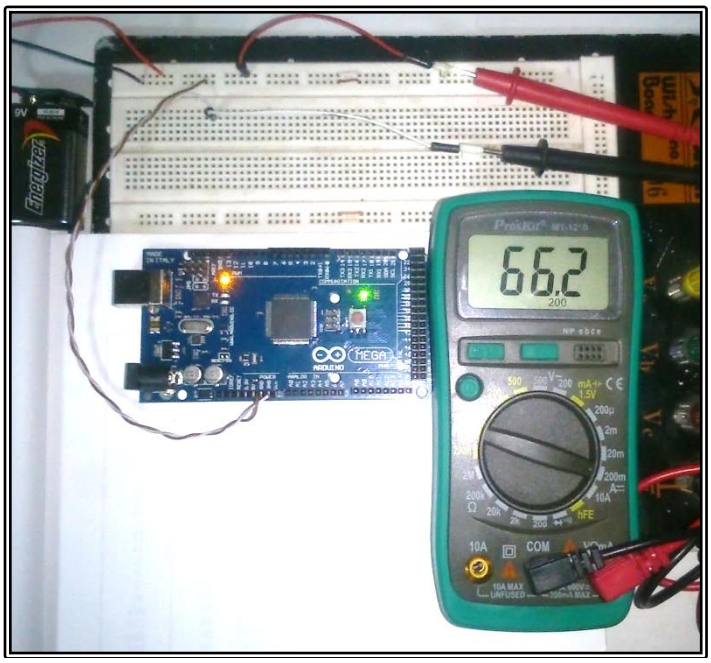


Figura 33-3. Esquema de medición energética
Fuente: PALATE, Byron. 2016

El esquema se realizó con dos fuentes de alimentación las cuales se encuentran entre los rangos permitidos por arduino, el primero es de 5V que es el voltaje con el que entra en funcionamiento la placa, el segundo es de 9V debido a que el voltaje máximo que soporta es de 12V, pero lo recomendable es no llegar a su límite de voltaje.

Se realizaron dos tipos de mediciones:

El consumo del sistema embebido Arduino por el simple hecho de estar activo, al tratarse de una medición estática su resultado es mucho más cómodo debido a que no existen cambios de estado ni fluctuaciones.

El consumo de todo el conjunto hardware en funcionamiento, este resultado tuvo unas variaciones debido a su módulo de comunicación que se encuentran en modo sleep para el ahorro de energía.

1.20.3.1 Contiki-OS

Consumo de Arduino Mega 2560

En la figura 34-3 se muestra la medición del consumo energético de la placa arduino con una fuente de 5 y 9V respectivamente.

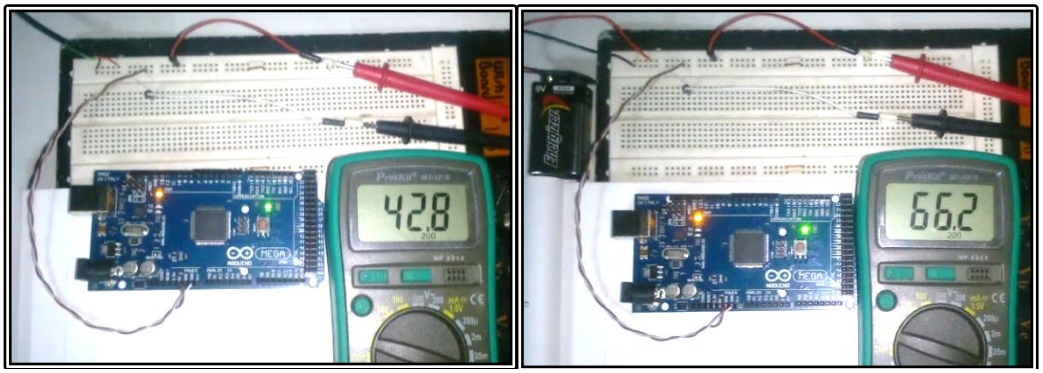


Figura 34-3. Consumo energético arduino-Contiki
Fuente: PALATE, Byron. 2016

En la tabla 2-3 se puede observar los resultados obtenidos de la placa arduino cuando se encuentra funcionando con el programa de Contiki, existiendo un descenso considerable al pasar de trabajar de los 9V a los 5V de alimentación, por lo que se puede deducir que la tensión de entrada afecta directamente al consumo que tendrá la placa.

Tabla 2-3: Consumo energético arduino-Contiki

Mota	Voltios	mA
Coordinador/sensor	5V	42.8
	9V	66.2

Realizado por:PALATE, Byron, 2016.
Fuente:PALATE, Byron. 2016.

Consumo energético de todo el conjunto

En la tabla 3-3 se representan todos los datos obtenidos en la medición, estos resultados se expresan en miliamperios (mA), y su consumo energético en operación aumenta considerablemente debido a que el shield xbee por si solo consume aproximadamente 1.9 mA, y no existe forma para reducir su consumo energético, según los datos técnicos del módulo xbee S1 su consumo en operación es casi 47 mA y en modo sleep es casi 0.9 mA.

Tabla 3-3: Consumo energético motas-Contiki

Mota	Voltios	Operación	Sleep
Coordinador	5V	90.9	47.8
	9V	113.2	70.1
Mota 1	5V	92.3	50.1
	9V	114.9	73.1
Mota 2	5V	92.6	50.6
	9V	115.1	73.4
Mota 3	5V	92.4	50.4
	9V	115.5	73.2

Realizado por: PALATE, Byron, 2016.
Fuente: PALATE, Byron. 2016.

En la figura 35-3 se puede apreciar la medición de la mota sensor 2 en sus dos estados, al izquierdo el modo operación y al derecho el modo sleep, con todos sus componentes tanto de censado y de transmisión, con una fuente de energía de 9V conectados conjuntamente con el multímetro.

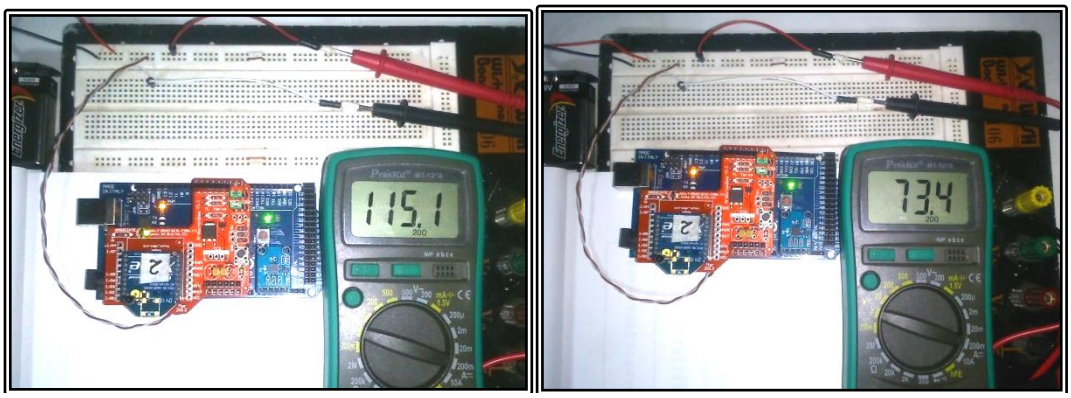


Figura 35-3. Consumo mota 2 Contiki
Fuente: PALATE, Byron. 2016

Cuando la mota se encuentra en modo operación el led del shield de comunicación permanece encendido, en cambio al momento de pasar al modo sleep este mismo diodo permanece apagado.

Existe una clara diferencia de consumo de energía entre el modo de operación y el modo sleep, esto debido a que en el modo sleep se desactivan algunas características tanto del radio de transmisión como del sistema embebido.
El modo de operación consume casi un 55% más que el modo sleep debido a que necesita censar y transmitir la información mediante su radio hacia la mota coordinador.

1.20.3.2 RIOT-OS

Consumo de Arduino Mega 2560

En la figura 36-3 se muestra la medición del consumo de energía de la placa arduino con una fuente de 5 y 9V respectivamente, cuando se encuentra con la plataforma Riot.

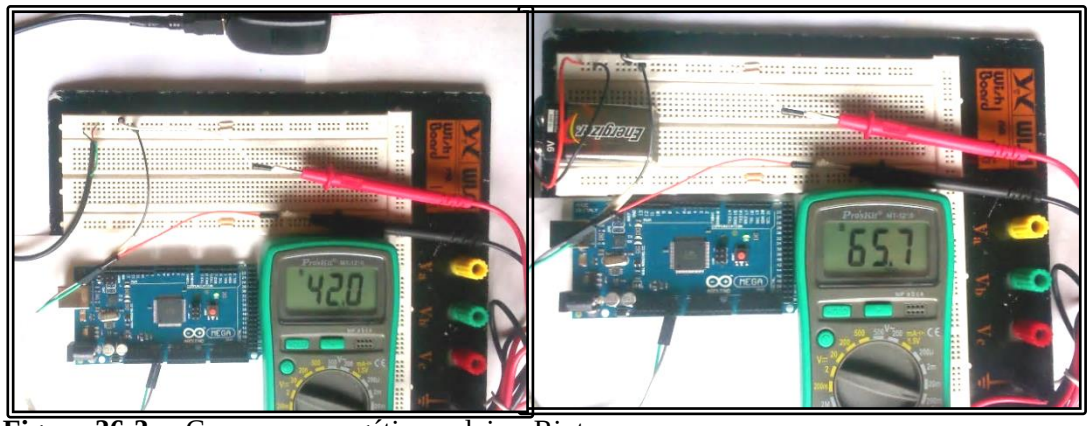


Figura 36-3. Consumo energético arduino-Riot
Fuente: PALATE, Byron. 2016

En los resultados de la tabla 4-3 se puede observar el consumo del sistema embebido arduino-mega2560 cuando se encuentra corriendo el programa de Riot, su consumo es muy bajo debido a que no cuenta con ningún sensor y tampoco ningún módulo de transmisión, su consumo es por simple funcionamiento.

Tabla 4-3: Consumo energético arduino-Riot

Mota	Voltios	mA
Coordinador/sensor	5V	42.0
	9V	65.7

Realizado por: PALATE, Byron, 2016.
Fuente: PALATE, Byron. 2016.

Consumo energético de todo el conjunto

En la tabla 5-3 se dan a conocer los resultados de la medición del consumo energético de la placa arduino ejecutando la plataforma Riot, estos resultados se expresan en miliamperios (mA), al igual que la medición de Contiki, esta se lo realizo con todos los elementos juntos, por esto existe tanta variación del consumo energético.

Tabla 5-3: Consumo energético motas-Riot

Mota	Voltios	Operación	Sleep
Coordinador	5V	90.2	47.2
	9V	112.0	68.9
Mota 1	5V	91.5	49.5
	9V	114.1	72.3
Mota 2	5V	92.0	50.0
	9V	114.3	72.8
Mota 3	5V	91.5	49.7
	9V	115.0	72.7

Realizado por: PALATE, Byron, 2016.
Fuente: PALATE, Byron. 2016.

En la figura 37-3 se puede apreciar la medición energética de la mota sensor 2 con todos sus componentes tanto de censado y de transmisión, al lado izquierdo en modo de operación (LED Shield ON), al derecho en modo sleep (LED Shield OFF) con una fuente de energía de 9V conectados al multímetro.

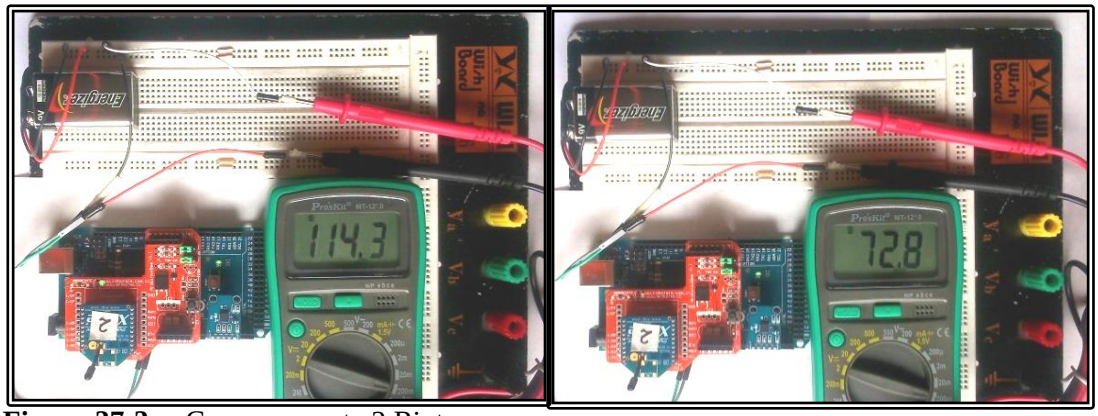


Figura 37-3. Consumo mota 2 Riot
Fuente: PALATE, Byron. 2016

1.21 Comparación entre las plataformas 6LowPan

Debido a los factores medidos anteriormente, podemos comparar estas plataformas Contiki-OS y Riot-OS, y derivar principios fundamentales para el IoT.

Contiki sigue un concepto modular cerca del enfoque por capas, su programación es puramente orientada a eventos, donde se utiliza una estrategia FIFO (First in, Firstout). Riot, por otro lado, utiliza un programador, lo que garantiza una distribución justa de tiempo de procesamiento.
Contiki utiliza un subconjunto del lenguaje de programación C, donde no se pueden usar algunas palabras clave, mientras que Riot por su parte, apoya un verdadero multi-threading, que está escrito en C estándar, y ofrece soporte para una amplia gama de diferentes lenguajes de programación y script.

Debido a estos compromisos de diseño, Contiki carecen de un desarrollador clave y amigable: estándar de programación C y C++, estándar multi-threading, y de apoyo en tiempo real.

Un sistema operativo aprovechando una implementación exitosa, a gran escala de los dispositivos IoT debe apoyar estas funciones. Desde la perspectiva del desarrollador, esto significa un potente API independiente del hardware. (Baccelli E., et al., 2013)

Tabla 6-3: Características principales de Contiki y Riot.

OS	Min. Mem.	Soporte C	Soporte C++	Multi-threadding	Modularidad	Tiempo Real
Contiki	<2kB	○	✖	○	○	○
Riot	~1,5kB	✓	✓	✓	✓	✓

Realizado por: PALATE, Byron, 2016.
Fuente: Riot-os.org.

- (✓) APOYO COMPLETO.
- (○) APOYO PARCIAL.
- (✖) NO Soporta.

La tabla 6-3 compara las plataformas con requerimientos mínimos de memoria para una aplicación básica, soporte para lenguajes de programación, multi-threading, modularidad y comportamiento en tiempo real.

1.21.1 Consumo de Memoria

En la figura 38-3 se muestran los porcentajes del consumo de memoria de la mota sensor de cada plataforma con respecto a la memoria total de arduino que es 8192 bytes.

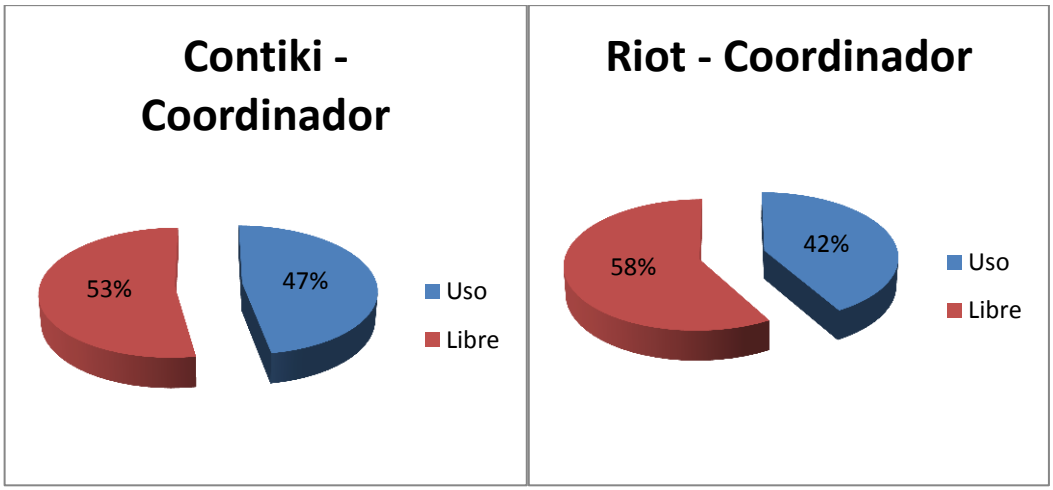


Figura 38-3. Estadística memoria coordinador
Fuente: PALATE, Byron. 2016

En la figura 39-3 se muestran los porcentajes del consumo de memoria de las motas sensores de cada plataforma con respecto a la memoria total de arduino que es 8192 bytes.

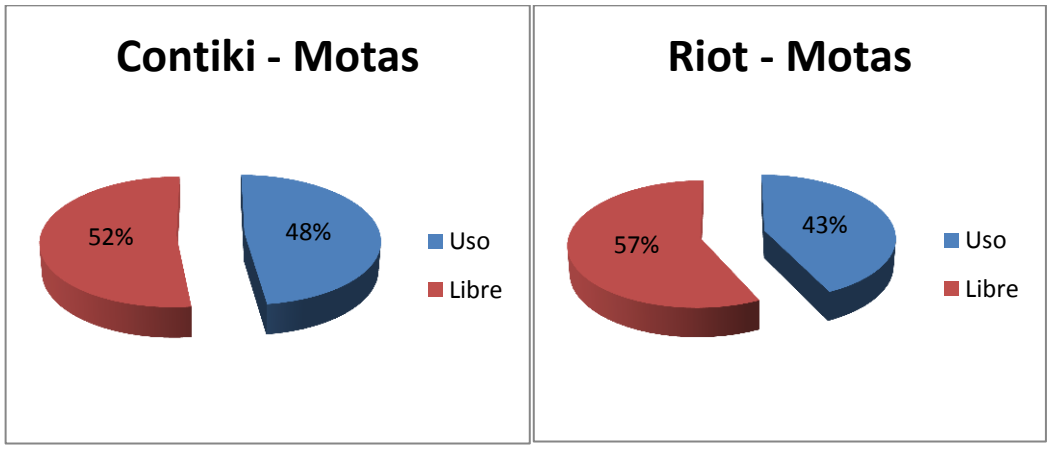


Figura 39-3. Estadística memoria motas sensores
Fuente: PALATE, Byron. 2016

1.21.2 Velocidad de Procesamiento

En la figura 40.3 se muestra la comparación de las velocidades de procesamiento de la mota coordinador entre las plataformas evaluadas.

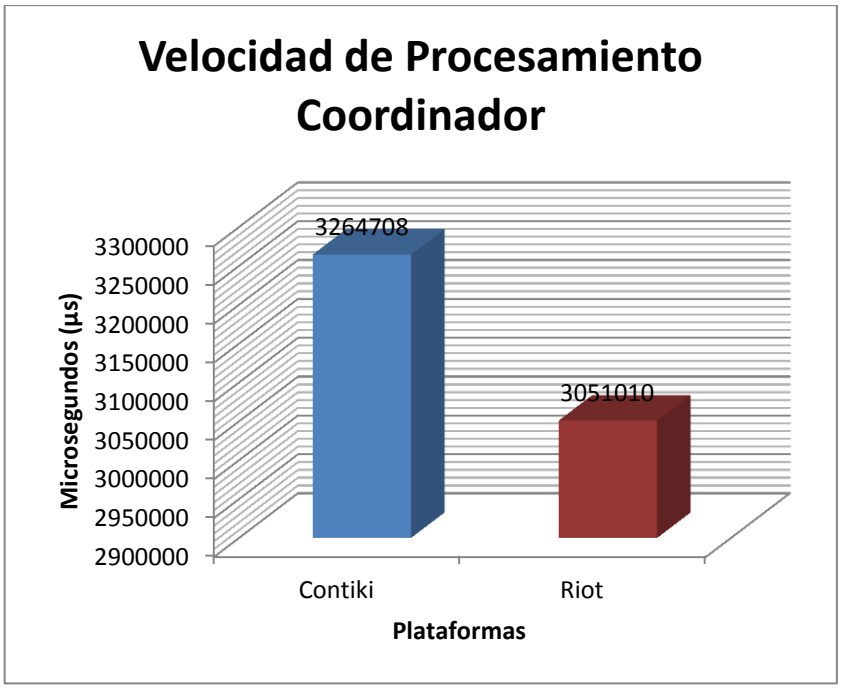


Figura 40-3. Estadística procesamiento coordinador
Fuente: PALATE, Byron. 2016

En la figura 41-3 se muestra la velocidad de procesamiento de las motas sensores en cada una de las plataformas.

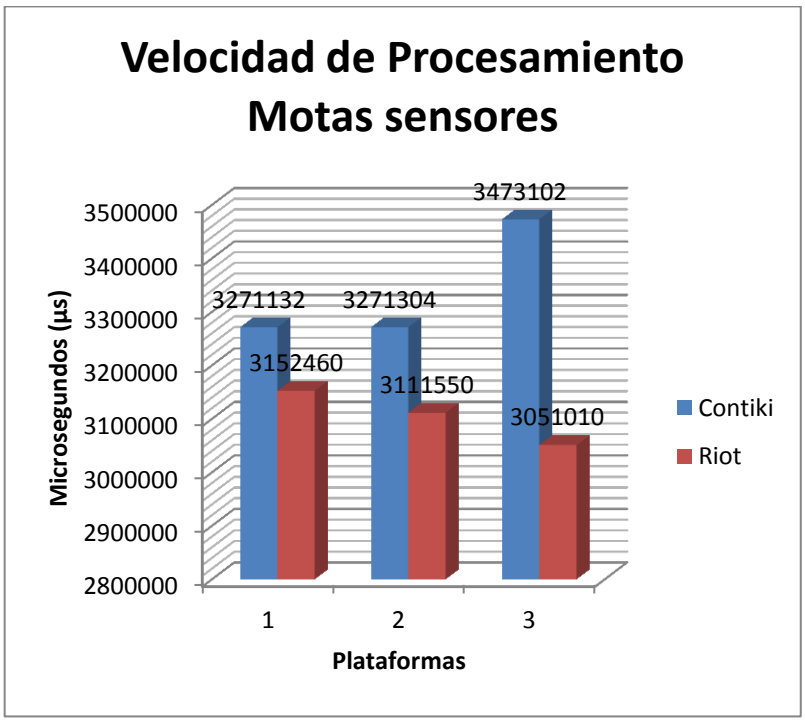


Figura 41-3. Estadística procesamiento motas sensores
Fuente: PALATE, Byron. 2016

En la figura 42-3 se puede observar una clara diferencia entre las velocidades de procesamiento promedio que se obtuvo de cada una de las plataformas.

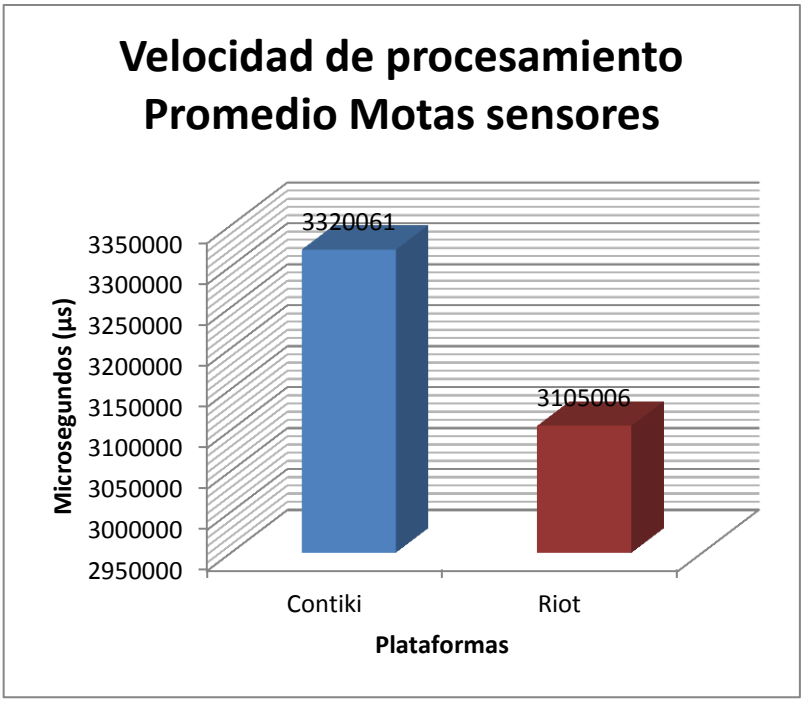


Figura 42-3. Estadística procesamiento promedio de motas sensores
Fuente: PALATE, Byron. 2016

1.21.3 Consumo de Energía

En la figura 43-3 se puede observar el consumo energético de la placa arduino.

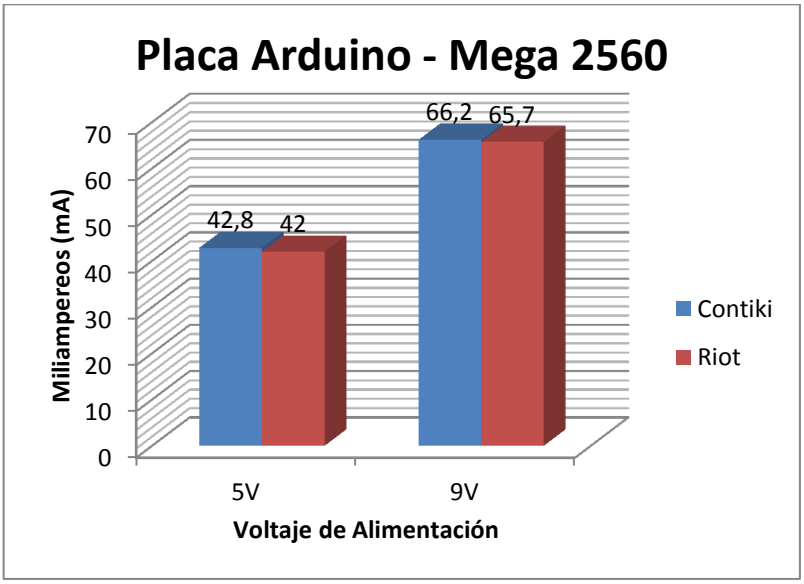


Figura 43-3. Estadística consumo arduino mega 2560 R3
Fuente: PALATE, Byron. 2016

En la figura 44.3 se muestra el consumo energético de la mota coordinador en el modo de operación.

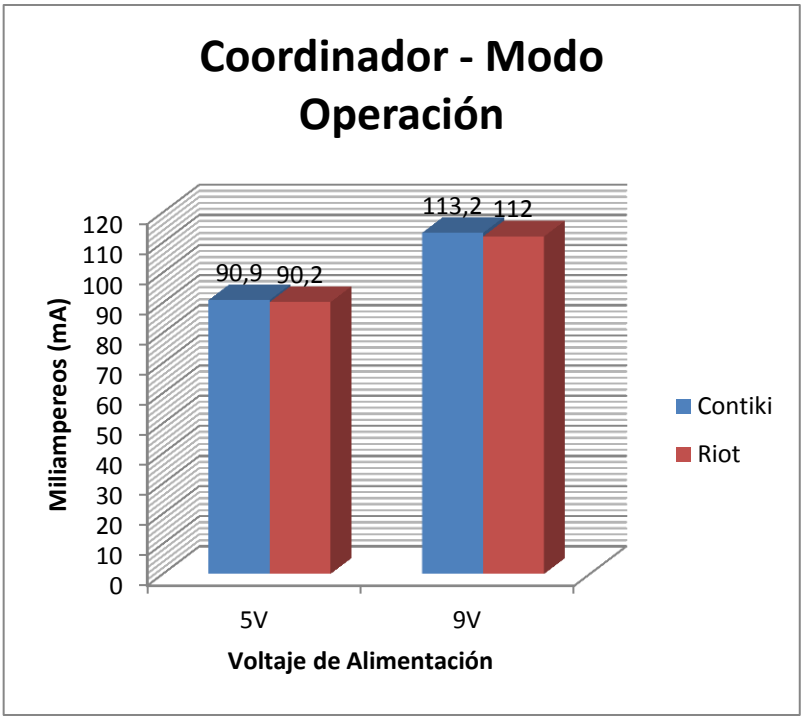


Figura 44-3. Estadística consumo coordinador operación
Fuente: PALATE, Byron. 2016

En la figura 45-3 se muestra el consumo energético de la mota coordinador en el modo sleep.

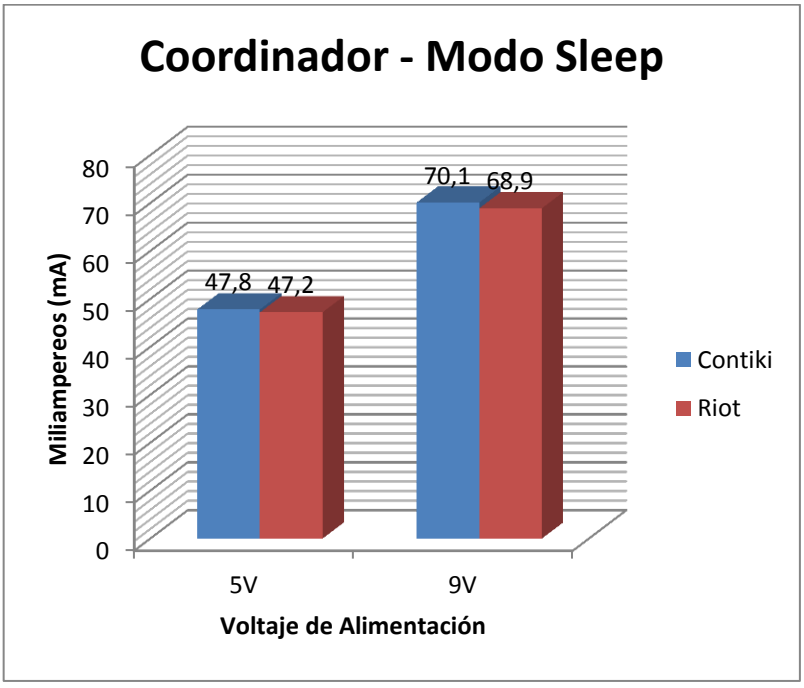


Figura 45-3. Estadística consumo coordinador sleep
Fuente: PALATE, Byron. 2016

En la figura 46.3 se puede observar el consumo energético de la mota sensor 1 en el modo de operación.

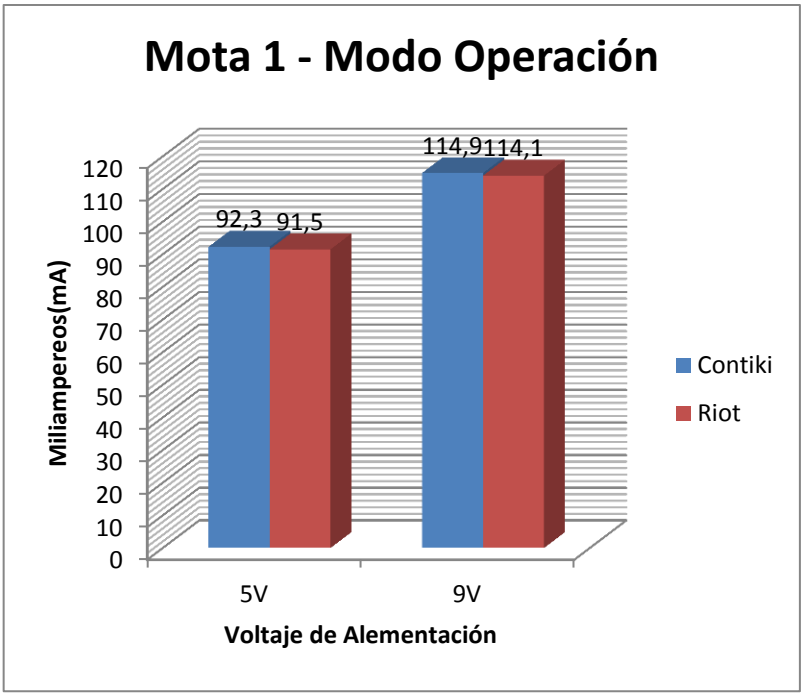


Figura 46-3. Estadística consumo mota1 operación
Fuente: PALATE, Byron. 2016

En la figura 47.3 se puede observar el consumo energético de la mota sensor 1 en el modo sleep.

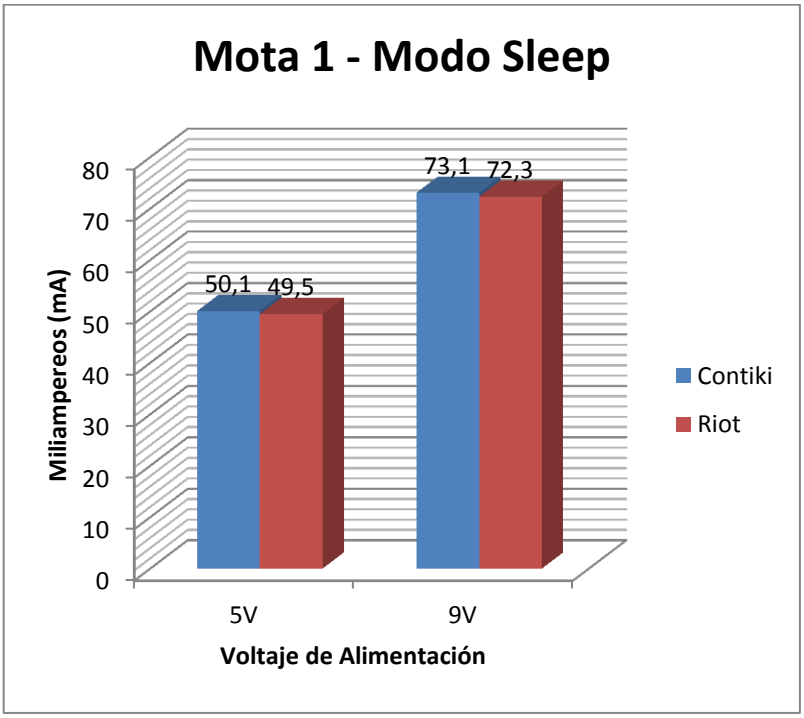


Figura 47-3. Estadística consumo mota1 sleep
Fuente: PALATE, Byron. 2016

En la figura 48-3 se puede observar gráficamente el consumo energético de la mota sensor 2 en el modo de operación.

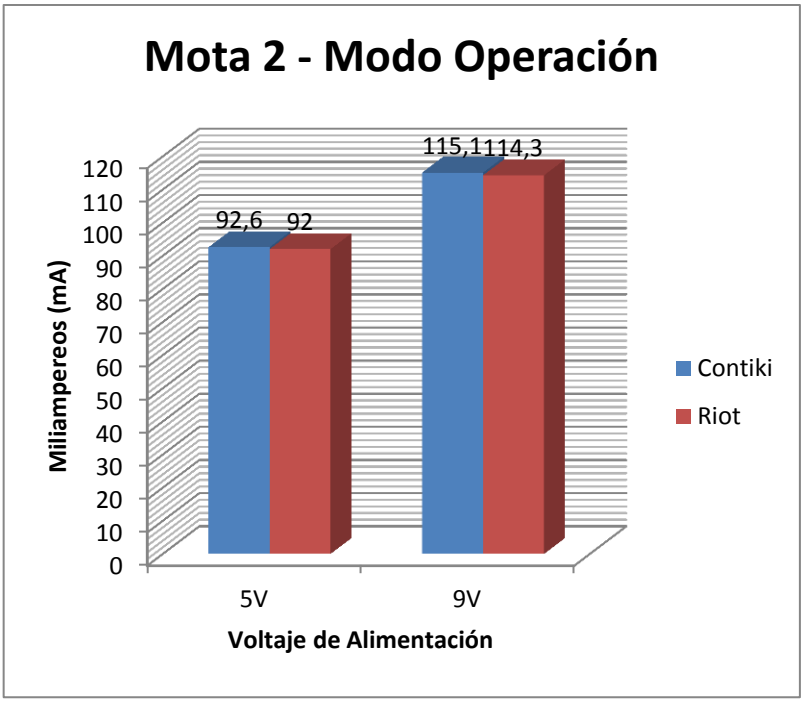


Figura 48-3. Estadística consumo mota2 operación
Fuente: PALATE, Byron. 2016

En la figura 49.3 se puede observar gráficamente el consumo energético de la mota sensor 2 en el modo sleep.

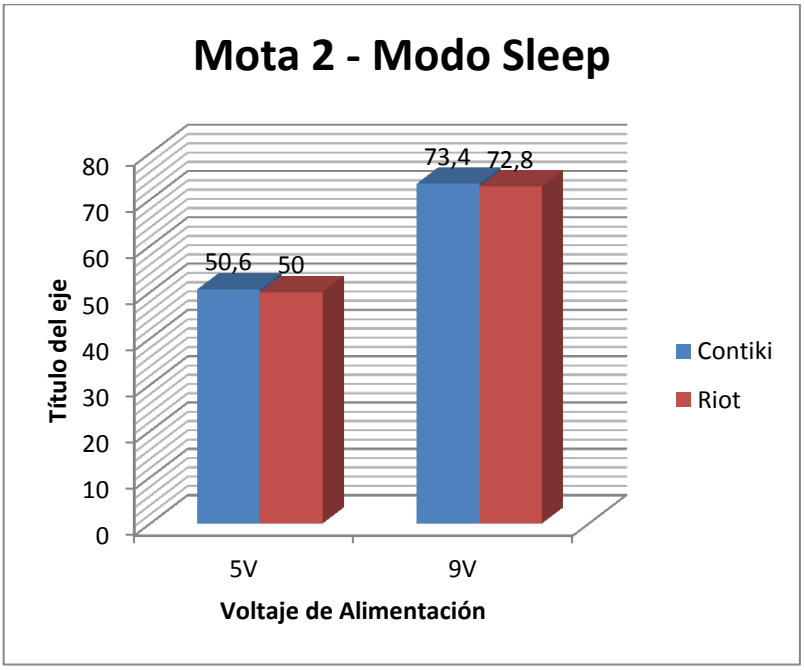


Figura 49-3. Estadística consumo mota2 sleep
Fuente: PALATE, Byron. 2016

En la figura 50.3 se puede observar el consumo energético de la mota sensor 3 en el modo de operación

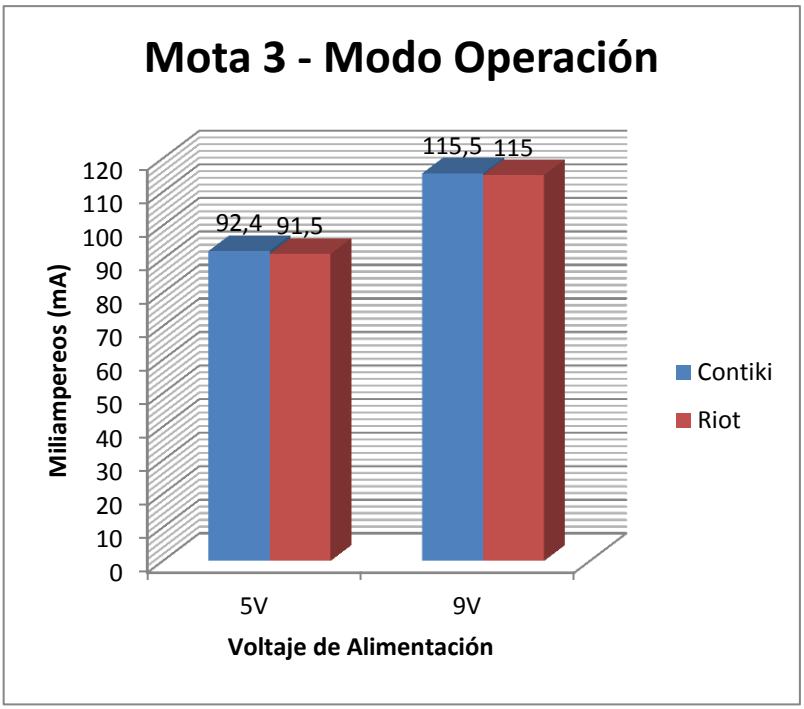


Figura 50-3. Estadística consumo mota3 operación
Fuente: PALATE, Byron. 2016

En la figura 51.3 se puede observar el consumo energético de la mota sensor 3 en el modo sleep.

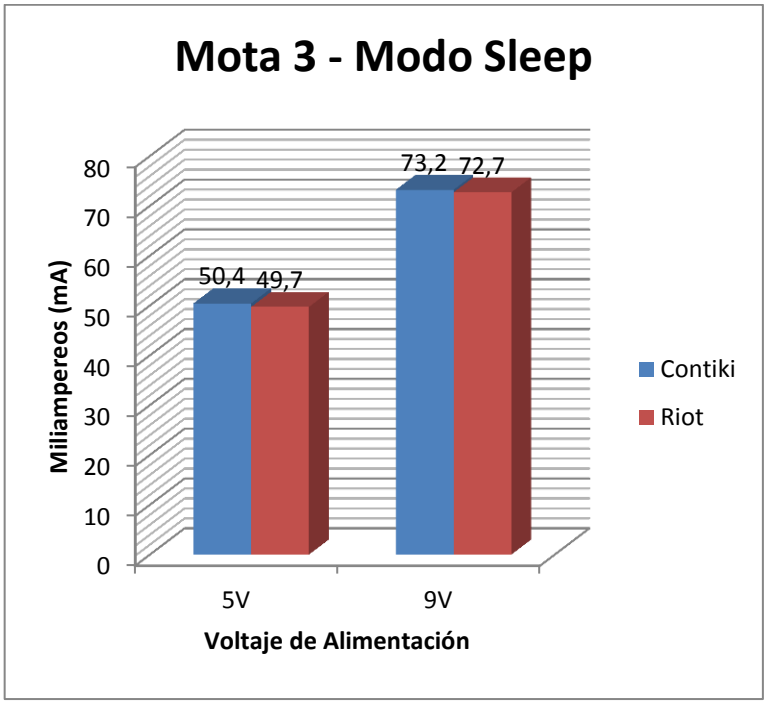


Figura 51-3. Estadística consumo mota3 sleep
Fuente: PALATE, Byron. 2016

1.22 Evaluación y selección de la plataforma más eficaz

Para poder evaluar los parámetros medidos de las plataformas, se ha tomado en cuenta variables entorno a las RFC (Request For Comments) 4919, 6606, que son publicaciones del grupo de trabajo de ingeniería de internet que describen diversos aspectos del funcionamiento del internet y otras redes de computadoras, como protocolos, procedimientos.

Dichos RFC establecen que la mejor opción es aquel sistema que posea la menor cantidad de consumo de recursos, para lo cual se establecerá un porcentaje del 100% de efectividad, y conforme aumente su consumo de recursos su efectividad deberá ir disminuyendo.
Por lo cual se establecen los siguientes valores para poder calificar a las variables medidas en el presente trabajo de titulación.

- Consumo de valor 3: Bajas tasas de consumo de recursos de un sistema embebido que posee bajo recursos de hardware. Efectividad de 100%.

- Consumo de valor 2: Tasas medias de consumo de recursos de un sistema embebido que posee bajo recursos de hardware. Efectividad de 66.66%
  .
- Consumo de valor 1: Altas tasas de consumo de recursos de un sistema embebido que posee bajo recursos de hardware. Efectividad de 33.33%.

- Consumo de valor 0: Tasas extremadamente altas de consumo de recursos de un sistema embebido que posee bajo recursos de hardware. Efectividad de 0%.

La valoración de la plataforma se lo realiza tomando en cuenta los resultados obtenidos en la sección anterior de este trabajo de titulación,

En la Tabla 7-3 se puede observar la valoración de los parámetros según los resultados obtenidos anteriormente en las mediciones.

Tabla 7-3: Valoración de parámetros.

	Contiki		Riot	
Variables	*Valoración*	*%*	*Valoración*	*%*
Min. Consumo Memoria	2	66,66%	3	100%
Min. Velocidad Procesamiento	1	33,33%	3	100%
Min. Consumo Energía	2	66,66%	2	66,66%
Total	5	55,55%	8	88,88%

Realizado por: PALATE, Byron, 2016.
Fuente: PALATE, Byron. 2016.

En la figura 52-3, se muestra el resultado final de la efectividad de las plataformas sobre los resultados obtenidos anteriormente.

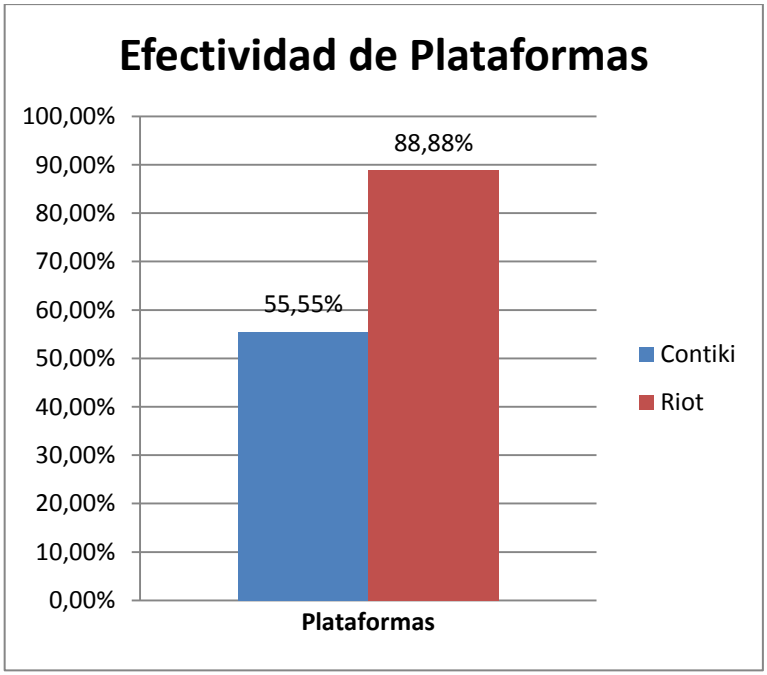


Figura 52-3. Efectividad de las plataformas
Fuente: PALATE, Byron. 2016

Debido a los resultados obtenidos anteriormente, se puede observar que la mejor opción para la implementación del proyecto en el sistema embebido Arduino mega 2560 R3 es la plataforma Riot-Os, debido a que ofrece un 33.33% más de efectividad que la plataforma Contiki.

1.23 Solución a la Sistematización de la problemática

Los parámetros que se determinaron la eficiencia para nuestro estudio de la plataforma, fueron el consumo de memoria, velocidad de procesamiento, y el consumo energético de cada una de las motas.

La tasa de transferencia más adecuada para cada una de las plataformas es de 250 Kbps, debido a que esta tasa es la más óptima para su uso dentro de las redes WSN y 6LowPan.

La efectividad del consumo de memoria en la plataforma Riot es de un 45%, a diferencia del 50% que obtuvo la plataforma Contiki.
La efectividad de velocidad de procesamiento de la plataforma Riot es muy superior al de Contiki, debido a que Riot posee 213698us de diferencia con la plataforma Riot.
La efectividad del consumo de energía es muy similar en ambas plataformas, según los resultados obtenidos solamente existe una variación de unas cuantas decimas de miliamperios, debido a esto su porcentaje de efectividad es la misma.

La principal ventaja de utilizar una plataforma de 6LowPan en el monitoreo estructural de puentes fue la de soportar ejecuciones en tiempo real, debido a su alto nivel de velocidad de procesamiento y bajo consumo de memoria.

CONCLUSIONES

El estándar 6LowPan entregó óptimos resultados al momento de implementar el proyecto en el sistema embebido arduino mega 2560, tanto por su velocidad de procesamiento, como el consumo de memoria por parte del programa, su escalabilidad se ve limitada únicamente por las características físicas del mismo sistema embebido que se implemente.

El software de simulación OMNeT++ nos da una idea clara en tanto al consumo de recursos del sistema, esto gracias a su módulo Mixim y a sus herramientas que presenta este software que ayudan a la programación de las motas según los parámetros establecidos en su archivo de configuración .ino, donde se establece parámetros importantes como el modo de comunicación, distancias, motas, así como la transmisión de datos, comunicación de la pila IPv6.

Al realizar las mediciones de consumo de memoria, velocidad de procesamiento y consumo de batería de la plataforma Contiki-OS y Riot-OS se comprueba que la mayor utilidad para reducir el consumo de recursos del sistema son las soluciones de software en el caso de arduino mega 2560 R3, para valorar su adecuación para entornos de bajo consumo.

A pesar de que no se ha conseguido un consumo de recursos que realmente pueda considerarse bajo, sí que se han podido registrar mejoras importantes en ellos. Además se ha hecho uso de la configuración sleep del módulo de comunicación que reduce considerablemente el uso de energía de la mota. Todo esto otorga a futuros trabajos una guía sencilla y útil para optimizar la realización de proyectos relacionados con 6LowPan.

El éxito del estudio se logra con la plataforma RIOT-OS alcanzando un porcentaje de efectividad del 88.88% a diferencia de la plataforma Contiki que obtuvo 55.55%. Con todo, es un resultado muy positivo y que aporta una solución real a diseños pensados para trabajar en futuros proyectos relacionados con el internet de las cosas.

RECOMENDACIONES

En el momento de la implementación se debe tomar en cuenta la intemperie a la que va a ser expuesto el sistema, debido a que se debe escoger una buena protección para las motas, caso contrario sufrirán imperfecciones y deberán ser reemplazadas constantemente, esto provocaría que el sistema sea ineficiente para futuros proyectos.

Muy necesario tener conocimiento sobre el sistema operativo Linux, debido a que así el trabajo de instalación de las plataformas se hace mucho más fácil sobre este, además que el repositorio oficial Github que hospeda el código de estas plataformas posee instrucciones o pasos a seguir solamente para este sistema operativo.

Es recomendable que la programación de las motas sea reducida en un mínimo de líneas de código posible, es decir evitar líneas de código innecesarias como redundancias, visualización de varios mensajes, declaración de variables sin usar, ya que esto provocara mayor consumo de memoria y mayor retardo en la velocidad de procesamiento

Tener mucho cuidado con la configuración de los jumpers del shield de comunicación para los módulos xbee, debido a que es la principal razón por la que el sistema embebido puede presentar fallos a corto o largo plazo, se recomienda que antes de comenzar la configuración opte por la lectura de algún manual.

Para la implementación de proyectos a futuro que también se centren en sismos, que necesiten enviar y recibir información en tiempo real, Riot es la mejor alternativa para su desarrollo debido a su gran capacidad de procesamiento y bajo consumo de memoria.

BIBLIOGRAFIAS

BACCELLI E., ET AL. "*RIOT OS: Towardsan OS forthe Internet of Things*". *Computer Communications Workshops (INFOCOM WKSHP)*, [en línea]. 2013. p: 79-80
[Consulta: 12 Diciembre 2015].
https://hal.inria.fr/hal-00945122/document

BACCELLI E., ET AL."*RIOT: One OS to Rule Them All in the IoT*" [en línea]. Centro de investigación saclay - ile de Francia. Research Report N°8176. ISSN 0249-6399. 2012.
[Consulta: 10 Diciembre 2015]
https://hal.inria.fr/hal-00768685v3/document

BAGNULO M., MATTHEWS P., VAN BEIJNUM I. "*Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers*", RFC 6146, IETF, 2011.
[Consulta: 27 Noviembre 2015]
http://www.rfc-editor.org/info/rfc6146

CAMA A., DE LA HOZ E., CAMA D. "*Las redes de sensores inalámbricos y el Internet de las cosas*". 8ª. ed. *INGE CUC* [en línea]. Barranquilla-Colombia. 2012, pp.163-172.
[Consulta: 03 Noviembre 2015]. ISSN 0122-6517.
http://revistascientificas.cuc.edu.co/index.php/ingecuc/article/view/253/232

CRAWFORD M. "*Transmission of IPv6 Packets over Ethernet Networks*", RFC 2464, IETF, Diciembre 1998.
[Consulta: 30 Noviembre 2015]
http://www.rfc-editor.org/info/rfc2464

CUJILEMA G. Y CHERRES P. "*Interoperabilidad de las redes wsn con arquitecturas tcp/ip corporativas mediante 6lowpan*" (**Tesis**). Escuela Superior Politécnica de Chimborazo, Facultad de Informática y Electrónica, Escuela de Ingeniería Electrónica en Telecomunicaciones y Redes. Riobamba-Ecuador. 2015, p.75-85.

[Consulta: 10Diciembre 2015]
http://bibliotecas.espoch.edu.ec/cgi-bin/koha/opac-detail.pl?biblionumber=49649

DUNKELS A., GRÖNVALL B., &VOIGT T.*"Contiki-a lightweight and flexible operating System for tiny networked sensors", 29th Annual IEEE International Conferenceon* [en línea], IEEE. ISSN 0742-1303, 2004. (pp. 455-462).
[Consulta: 07 Diciembre 2015].
http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=9433&filter%3DAND%28p_IS_Number%3A29935%29&pageNumber=4

ECUADOR. INSTITUTO NACIONAL DE ESTADÍSTICAS Y CENSOS (INEC).*Ecuador en cifras.*Quito-Ecuador.2015.
[Consulta: 20 Noviembre 2015]
Disponible en: http://www.ecuadorencifras.gob.ec/

ENERLIS, ERNST AND YOUNG, FERROVIAL AND MADRID NETWORK. "*Libro blanco Smart Cities*". 1a ed. Madrid-España, Imprintia. 2012, p.5
[Consulta: 20 Noviembre 2015]
http://www.innopro.es/pdfs/libro_blanco_smart_cities.pdf

ESPAÑA.ASOCIACIÓN ESPAÑOLA PARA LA CALIDAD(AEC). "*Smart Cities-Ciudades Inteligentes*", Madrid-España 2012, p.1.
[Consulta: 15 Noviembre 2015]
http://www.aec.es/c/document_library/get_file?uuid=c8b4e9a6-1796-4e90-9ddd-4ceb3ac66a81&groupId=10128

ESPINOSA C. "*Adaptación del algoritmo OMEGA para ser utilizado en un prototipo de una red de sensores inalámbrica 6LowPan*" (**Tesis**), Escuela Politécnica Nacional, Facultad de Ingeniería Eléctrica y Electrónica. Ingeniería en Electrónica y Redes de Información. Quito-Ecuador, 2015, p.53
[Consulta: 01 Diciembre 2015]
http://bibdigital.epn.edu.ec/bitstream/15000/11775/1/CD-6533.pdf

EVANS D. "*Internet of Things La próxima evolución de Internet lo está cambiando todo"*, San José-América Central. Grupo de Soluciones Empresariales para Internet (IBSG) de Cisco, Abril 2011, p. 2.
[Consulta: 5 Noviembre 2015]
http://www.cisco.com/web/LA/soluciones/executive/assets/pdf/internet-of-things-iot-ibsg.pdf

EVERIET A., PASTOR J."*Introducción al internet de las cosas construyendo un proyecto de IoT"*.Universidad Rey Juan Carlos. Madrid-España. 2013, p. 5
[Consulta: 5 Noviembre 2015]
https://www.carriots.com/newFrontend/img-carriots/press_room/Construyendo_un_proyecto_de_IOT.pdf

FUNDACIÓN TELEFÓNICA. "*Smart Cities: un primer paso hacia la internet de las cosas*", Madrid-España, Ariel, Informe 16, 2011, p.20
[Consulta: 20 Noviembre 2015]
http://www.socinfo.es/contenido/seminarios/1404smartcities6/01-TelefonicaSMART_CITIES-2011.pdf

HUI J., ED.,THUBERT P."*Compression Formatfor IPv6 Datagrams over IEEE 802.15.4-Based Networks*". RFC 6282. IETF. Septiembre 2011.
[Consulta: 27 Noviembre 2015]
http://www.rfc-editor.org/info/rfc6282

KIM E., KASPAR D., GOMEZ C, BORMANN C."*Problem Statement and Requirements for IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Routing*", RFC 6606. IETF, Agosto 2012.
[Consulta: 09 Diciembre 2015]
http://www.rfc-editor.org/info/rfc6606

KUSHALNAGAR N., MONTENEGRO G.SCHUMACHER C. "*IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals*", RFC 4919, IETF, Agosto 2007.
[Consulta: 09 Diciembre 2015]
http://www.rfc-editor.org/info/rfc4919

MARULANDA J., ET AL. "Monitoreo de salud estructural". 2ª. ed. Universidad del Valle.Cali-Colombia. 2011. p.40-46.
[Consulta: 15 Diciembre 2015]
http://bibliotecadigital.univalle.edu.co/handle/10893/1511

MONTENEGRO G., HUI J., CULLER D. "*Transmission of IPv6 Packets over IEEE 802.15.4 Networks*". RFC 4944. IETF. 2007.
[Consulta: 27 Noviembre 2015]
http://www.rfc-editor.org/info/rfc4944

OMNETPP. "*OMNeT++ Discrete Event Simulator".* Budapest-Hungría, 2015
[Consulta: 15 Enero 2016]
https://omnetpp.org/documentation

PATRICIA M. "*Implementación de un sistema de monitorización de co2 mediante redes de sensores inalámbricos en el campus universitario de la ESPE*" (**Tesis**). Escuela Superior Politécnica del Ejército. Departamento de Eléctrica y Electrónica. Carrerade Ingenieríaen Electrónica y Telecomunicaciones Sangolquí-Ecuador. 2015, p.23
[Consulta: 19 Noviembre 2015]
http://repositorio.espe.edu.ec/bitstream/21000/9952/1/T-ESPE-048690.pdf

ROMÁN M. "*Plan de prevención para emergencias por desastres naturales en la provincia de pichincha, su organización y aplicación en la educación básica en la próxima década*". (**Tesis**). República del Ecuador. Instituto de altos estudios nacionales. XXXIII Curso Superior De Seguridad Nacional Y Desarrollo. Quito-Ecuador, 2006, pp.13-14

[Consulta: 03 Noviembre 2015]
http://repositorio.iaen.edu.ec/bitstream/24000/51/1/CD-IAEN-0110.pdf

SÁNCHEZ G. "*Estudio de seguridades en una red extremo a extremo, basada en protocolo ipv6*" (**Tesis**), Escuela Superior Politécnica de Chimborazo, Facultad de Informática y Electrónica. Escuela de Ingeniería Electrónica, Riobamba-Ecuador. 2012. p.19
[Consulta: 30 Noviembre 2015]
http://dspace.espoch.edu.ec/bitstream/123456789/1934/1/38T00287.pdf

SHELBY Z., BORMANN C. "*6LoWPAN: The Wireless embedded Internet*". Reino Unido. John Wiley&Sons. 2011, 43ª. ed.
[Consulta: 24 Noviembre 2015]
http://keshi.iothust.org/2014IoTComm/book/6LoWPAN_The%20Wireless%20Embedded%20Internet.pdf

SHELBY Z., HARTKE K., BORMANN C. *"The Constrained Application Protocol (CoAP)*", RFC 7252, IETF, Junio 2014.
[Consulta: 28 Noviembre 2015]
http://www.rfc-editor.org/info/rfc7252

WINTER T., ED., THUBERT P., ET AL. "*RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks*", RFC 6550, IETF, Marzo 2012
[Consulta: 30 Noviembre 2015]
http://www.rfc-editor.org/info/rfc6550

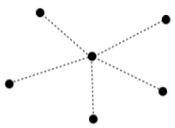
ANEXOS

ANEXO 1

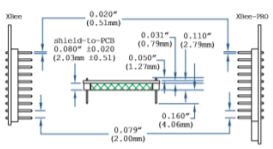
Datasheet Xbee S1 (802.15.4)

SPECIFICATIONS	XBee® 802.15.4	XBee-PRO® 802.15.4
PERFORMANCE		
RF DATA RATE	250 kbps	250 kbps
INDOR/URBAN RANGE	100 ft (30 m)	300 ft (100 m)
OUTDOOR/RF LINE-OF-SIGHT RANGE	300 ft (100 m)	1 mi (1.6 km)
TRANSMIT POWER	1 mW (+0 dBm)	60 mW (+18 dBm)*
RECEIVER SENSITIVITY (1% PER)	-92 dBm	-100 dBm
FEATURES		
SERIAL DATA INTERFACE	3.3V CMOS UART	
CONFIGURATION METHOD	API or AT Commands, local or over-the-air	
FREQUENCY BAND	2.4 GHz	
INTERFERENCE IMMUNITY	DSSS (Direct Sequence Spread Spectrum)	
SERIAL DATA RATE	1200 bps - 250 kbps	
ADC INPUTS	(6) 10-bit ADC inputs	
DIGITAL I/O	8	
ANTENNA OPTIONS	Chip, Wire Whip, U.FL, & RPSMA	
NETWORKING & SECURITY		
ENCRYPTION	128-bit AES	
RELIABLE PACKET DELIVERY	Retries/Acknowledgments	
IDS AND CHANNELS	PAN ID, 64-bit IEEE MAC, 16 Channels	
POWER REQUIREMENTS		
SUPPLY VOLTAGE	2.8 - 3.4VDC	2.8 - 3.4VDC
TRANSMIT CURRENT	45 mA @ 3.3VDC	215 mA @ 3.3VDC
RECEIVE CURRENT	50 mA @ 3.3VDC	55 mA @ 3.3VDC
POWER-DOWN CURRENT	<10 uA @ 25° C	
REGULATORY APPROVALS		
FCC (USA)	OUR-XBEE	OUR-XBEEPRO
IC (CANADA)	4214A-XBEE	4214A-XBEEPRO
ETSI (EUROPE)	Yes	Yes - Max TX 10 mW
C-TICK AUSTRALIA	Yes	
TELEC (JAPAN)	Yes	

802.15.4 – Star

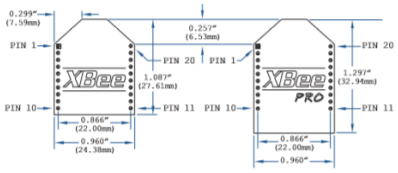


(Side Views)



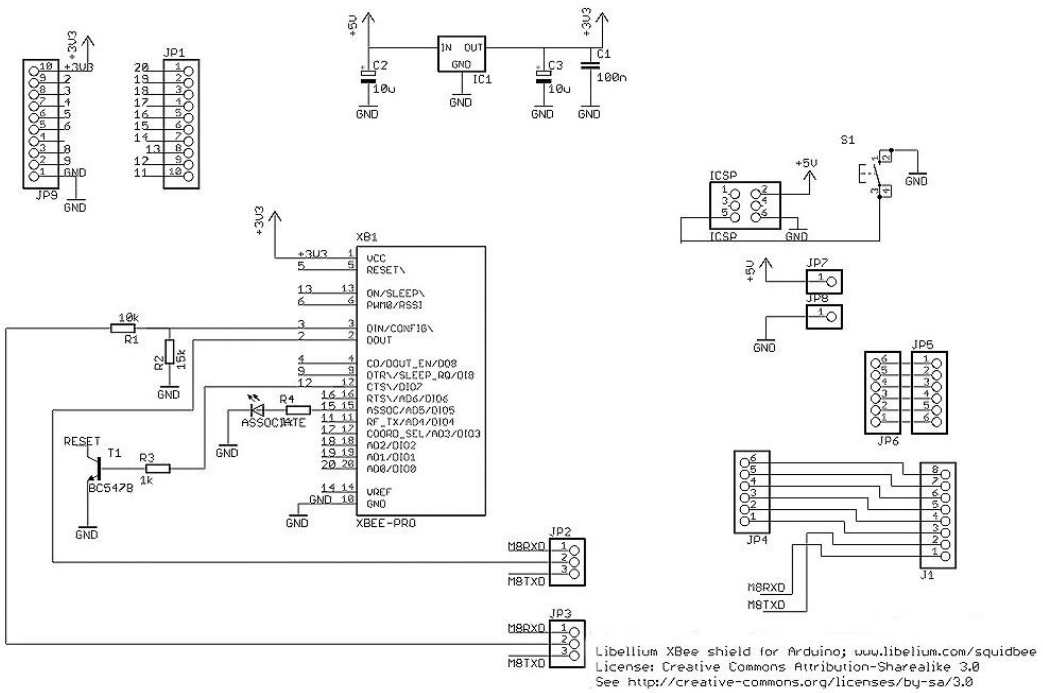
(Top View)

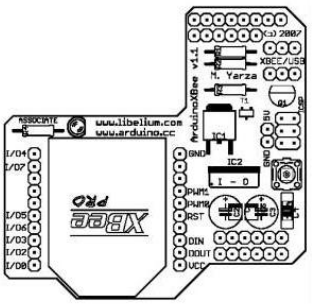
(Top View)

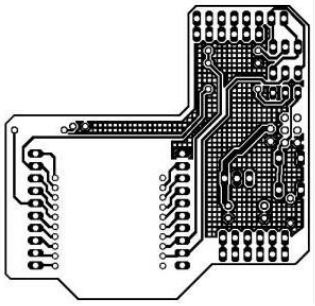


ANEXO 2

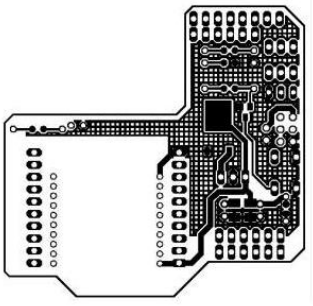
Schematic Arduino Xbee shield







XbeeShieldBottom



XbeeShieldTop

ANEXO 3

Schematic Arduino Mega 2560