PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

FACULTAD DE INGENIERÍA

MAESTRÍA EN REDES DE COMUNICACIONES

"DISEÑO E IMPLEMENTACIÓN DE UN PROTOTIPO DE SISTEMA DE GESTIÓN DE LUMINARIAS LED PARA LA EMPRESA ELÉCTRICA QUITO UTILIZANDO UNA RED DE SENSORES INALÁMBRICOS CON 6LoWPAN"

PATRICIO RODRIGO ARELLANO VARGAS

TRABAJO PREVIO LA OBTENCION DEL TÍTULO DE MASTER EN REDES DE COMUNICACIONES

Quito – noviembre de 2015

AUTORÍA

Yo, PATRICIO RODRIGO ARELLANO VARGAS, portador de la cédula de ciudadanía No. 1706996442, declaro bajo juramento que la presente investigación es de total responsabilidad del autor, y que se ha respetado las diferentes fuentes de información realizando las citas correspondientes.

_____

Ing. Patricio Rodrigo Arellano Vargas

CC. 1706996442

DEDICATORIA

Dedico este trabajo de titulación a mi familia, la que me acogió desde pequeño y la que he formado junto a mis maravillosas esposa e hija. Ustedes son la razón de mis esfuerzos y mi afán de superación. El amor inagotable que me entregan cada día es mi combustible para avanzar y superar los obstáculos que se presentan en el camino.

AGRADECIMIENTO

Agradezco sobre todo a Dios por sentirlo tan cercano en los peores momentos de mi vida y aún más en los mejores momentos, por ser esa fuente inagotable de amor.

A mi madre y abuelita por apoyarme en todo desde cuando tengo uso de razón, a mis hermanos por sus palabras de apoyo y amor.

Un capítulo especial de agradecimiento lo constituye mi maravillosa esposa y madre de la más grande responsabilidad que Dios nos confió, nuestra amada hija. Ustedes significan la razón de mi constante motivación para superarme y alcanzar cada vez metas más grandes. Solo puedo decir que Dios les pague todo eso que hacen por mí y que trataré de corresponderles en todo lo que esté a mi alcance.

También es importante expresar mi agradecimiento a la Pontificia Universidad Católica del Ecuador y en particular al Ing. Carlos Egas, mi director de este proyecto, quien me supo guiar durante el desarrollo del mismo. Gracias por su apoyo y tiempo sin el cual no hubiese sido posible finalizarlo.

Gracias a mis correctores el Ing. Juan Francisco Chafla y la Ing. María Soledad Jiménez, por su colaboración y apoyo en la terminación del documento escrito.

Finalmente dejo constancia de mi agradecimiento a todos mis amigos, colegas y compañeros de este hermoso camino llamado vida. Gracias de todo corazón.

CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El presente trabajo se enfoca en presentar el diseño y evaluación, utilizando una plataforma de desarrollo provista por el fabricante (Libelium), de un sistema de gestión de luminarias públicas LED[1] para la Empresa Eléctrica Quito utilizando 6LoWPAN[2], así como la implementación de un prototipo en laboratorio del sistema.

Para cumplir con el objetivo del proyecto se plantea diseñar una red de sensores inalámbricos que recolectan la información pertinente y la envían a un Gateway. Los sensores usan el protocolo 6LoWPAN sobre la capa de enlace del 802.15.4[3] para crear una red mallada que interconecta todos los dispositivos de la red con el Gateway[4].

El Gateway recibe los datos de los nodos terminales (paquetes 6LoWPAN), cambia las cabeceras IP[5] a IPv4 y mantiene la capa de transporte UDP[6], luego envía la información a la computadora de túnel IPv4/IPv6[7] que cambia la cabecera nuevamente a IPv6 y la envía al

[1] LED: *Light Emitting Diode*, Diodo emisor de luz, es un componente opto electrónico pasivo que emite luz.
[2] 6LoWPAN: *IPv6 over Lowpower Wireless Personal Area Networks*, es un estándar que posibilita el uso de IPv6 sobre redes basadas en el estándar IEEE 802.15.4.
[3] 802.15.4: es un estándar que define el nivel físico y el control de acceso al medio de redes inalámbricas de área personal con tasas bajas de transmisión de datos.
[4] Gateway: es el dispositivo que permite interconectar redes de computadoras con protocolo de comunicaciones y arquitecturas diferentes a todos los niveles de comunicación.
[5] IP: *Internet Protocol*, es un protocolo de comunicación de datos digitales clasificado funcionalmente en la Capa de Red según el modelo internacional OSI.
[6] UDP: *User Datagram Protocol*, es un protocolo del nivel de transporte basado en el intercambio de datagramas (Encapsulado de capa 4 Modelo OSI).
[7] IPv6: *Internet Protocol Version 6*, es una versión del protocolo Internet Protocol (IP), definida en el RFC 2460 y diseñada para reemplazar a Internet Protocol version 4 (IPv4) RFC 791, que actualmente está implementado en la gran mayoría de dispositivos que acceden a Internet.

servidor.

Cada luminaria representa un nodo de la WSN[8] y contiene un sensor de apagado-encendido de la misma, es decir una entrada digital que recebe una muestra de la señal que alimenta a la luminaria.

El uso de 6LoWPAN nos permite varias ventajas:

- Es un estándar abierto y confiable.
- Integración transparente con internet y uso de su infraestructura.
- Mantenimiento de la red.
- Escalabilidad global.
- Flujo de datos end-to-end.

De lo anterior podemos concluir que el uso de 6LoWPAN es válido para el proyecto pues sus ventajas son útiles para el escenario planteado y nos permitirá comprobar que es útil para este tipo de soluciones.

[8] WSN: *Wireless Sensor Network,* son sensores autónomos especialmente distribuidos con el fin de monitorear condiciones físicas o ambientales y transmitir los datos cooperativamente a través de la red a una ubicación principal.

CAPÍTULO 1

1. INTRODUCCIÓN

1.1 Antecedentes

Las luminarias que se han venido utilizando regularmente en Quito y la mayor parte del país son de vapor de sodio, este tipo de luminaria desperdicia aproximadamente el 46% de luz puesto que el foco de emisión irradia en varias direcciones y luego se direcciona gracias a la parábola que las contiene. Las luminarias LED emiten directamente la luz sin necesidad de la parábola logrando que un mayor porcentaje de la luz se pueda aprovechar, tal como se muestra en la figura 1.1.

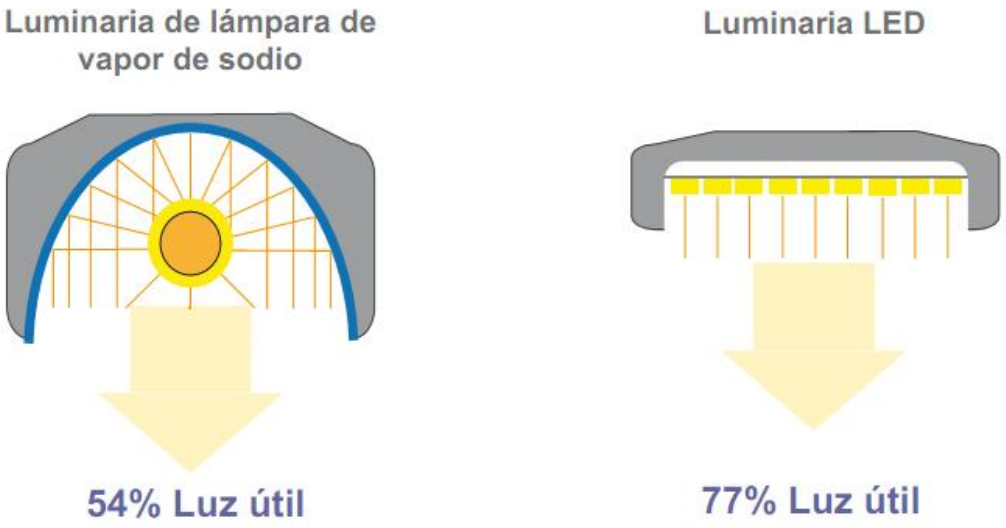


Figura 1.1 Comparación entre lámpara de vapor de sodio y LED[1]
Fuente: "I Congreso EI: Iluminación LED en edificios inteligentes - CASADOMO."

Las lámparas LED tienen varias ventajas comparadas con las lámparas de vapor de sodio:

- El promedio de vida útil de una lámpara de sodio[9] es de 7000 a 10000 horas dependiendo si son de baja o de alta presión respectivamente, mientras que cuando hablamos de lámparas LED su vida útil es de 50000 horas.
- Las lámparas LED tienen un 25% por ciento más de luminosidad.
- La contaminación del medio ambiente es nula.
- La respuesta es más rápida.
- La calidad de luz es mejor.
- Emiten mucho menos calor.

Aun cuando el consumo de energía de las lámparas LED es menor a las convencionales, el uso de un sistema de gestión de las mismas permitiría un mayor ahorro de energía pues se podría detectar las luminarias que, por ejemplo, permanecen encendidas durante el día.

1.1.1 Breve historia del alumbrado público

Para el año 1879 Thomas Alva Edison había inventado la bombilla incandescente, la misma que emitía luz utilizando el efecto Joule, es decir, calentaba un filamento metálico para cumplir con su propósito.

[9]Lámpara de vapor de sodio: es un tipo de lámpara de descarga de gas que usa vapor de sodio para producir luz.

Con esta disposición, el rendimiento era demasiado bajo puesto que se desperdiciaba más del 90% de energía en forma de calor o de radiación no perceptible. Esto permitió, entre otras cosas, el inicio de los sistemas de alumbrado público eléctrico.

Posteriormente se fueron sucediendo distintas tecnologías que permitieron hacer más eficiente el sistema. Cabe mencionar como un punto importante que las lámparas de descarga de alta densidad incrementaron notablemente el rendimiento (hasta cuatro veces superior), la primera que apareció fue la de Vapor de Mercurio en los años 30. Se les denomina lámparas de descarga puesto que la luz emitida se consigue por excitación de un gas sometido a descargas eléctricas entre dos electrodos. Las lámparas de descarga se clasifican de acuerdo al tipo de gas que contienen y la presión a la que se encuentre (alta o baja).

Precisamente en los años 30 aparecieron las lámparas fluorescentes tubulares, con una mejor eficiencia lumínica y un índice de reproducción cromática superior (IRC[10])[1], la desventaja era que la vida útil era inferior a la de las lámparas de Vapor de Mercurio.

Para la misma década se empezó a utilizar las lámparas de Vapor de Sodio de baja presión, que tenían la mayor eficacia existente, aunque una baja reproducción cromática debido a su color (amarillo monocromático) por lo que se usa en los casos donde el color de la luz

[10] IRC: es una medida de la capacidad que una fuente luminosa tiene para reproducir fielmente los colores de varios objetos en comparación con una fuente de luz natural o ideal.

no es importante (autopistas, túneles, etc.).

Las lámparas de Vapor Mezcla, son un intermedio entre las de Vapor de Mercurio de alta presión y las incandescentes. Normalmente traían un recubrimiento fluorescente. Su uso se fue generalizando puesto que el filamento actuaba como estabilizador de corriente.

Para los años 60 se presentó un auge en el desarrollo de las lámparas de descarga de alta intensidad (HID*, High Intensity Discharge*), pasando de las de Vapor de Sodio de alta presión a las de Halogenuros metálicos. Estas últimas, a pesar de tener una menor eficiencia y vida útil, producen luz blanca con un elevado índice de reproducción cromática.

La primera década del siglo XXI ha introducido en el mercado la tecnología LED (*Light-Emitting Diode*) diseñada como un componente electrónico, que desde entonces ha tenido una evolución constante. Actualmente tienen un rendimiento lumínico de hasta 150 lúmenes por vatio.

La tecnología LED ha iniciado una nueva revolución en el campo del alumbrado público, mientras que las tecnologías anteriores tuvieron una evolución lenta, los LED en menos de una década han cambiado notablemente en su rendimiento. Entre tanto vemos que las tecnologías tradicionales se encuentran en una etapa madura de su evolución.

En cuanto a Quito específicamente podemos notar la evolución del alumbrado público en las siguientes etapas:

- Farolas a gas en 1900.
- Farolas con bujías de luz incandescente 1920.
- Luminarias tipo sol con lámpara incandescentes de alta potencia 1950.
- Primeras luminarias de descarga 1970.
- Luminarias de descarga de alta potencia 2000.
- Luminarias LED 2012.

1.2 Justificación

La Empresa Eléctrica Quito ha iniciado el proyecto de cambio de las luminarias convencionales a las de LED, de acuerdo a la evolución de la tecnología en alumbrado público y con el propósito de reducir el consumo de energía eléctrica y por tanto disminuir costos de funcionamiento de las mismas.

Las luminarias LED son más amigables con el medio ambiente al no contener mercurio, lo que hace que su reciclaje sea más sencillo y menos contaminante. Su bajo consumo se traduce en menor emisión de CO_2 y azufre, disminuyendo la huella de carbono por el uso de luminarias.

En el centro histórico de la ciudad de Quito se han instalado numerosas luminarias y actualmente se encuentran funcionando a su capacidad total y controladas únicamente por foto celdas en cuanto al mecanismo

de encendido y apagado de las mismas.

Frente a este escenario, se justifica la necesidad de un sistema de gestión de las luminarias, que permita ciertas bondades en la operación de las mismas, tales como:

- Control del estado de las luminarias en cuanto a encendido/apagado.
- Detectar una luminaria con falla y poder ordenar su reemplazo con mayor celeridad.

Uno de los principales beneficios de implementar un sistema de gestión de las luminarias es la de lograr el máximo de duración de las mismas, contribuyendo a la disminución de contaminación producido por el cambio continuo de las mismas.

Si tomamos en cuenta que la duración de una luminaria LED es mucho mayor a las que se vienen utilizando en nuestra ciudad, la implementación de un sistema que nos permita llevar un control de las mismas se ve plenamente justificada.

En la actualidad en la EEQ no existe un sistema que permita llevar el control de las luminarias públicas. Algunas permanecen encendidas durante el día, otras no se encienden en la noche y por último algunas se han quemado por el uso regular. Estas novedades se conocen por medio de los usuarios que las reportan o por medio del personal de la Empresa involucrado en estas tareas. El uso de un sistema automatizado para detectar el estado de las luminarias será de gran

utilidad para la Empresa puesto que de esta forma se solucionarán las novedades que se presentan en su funcionamiento cotidiano de una manera oportuna.

1.3 Objetivos

1.3.1 Objetivo general

Diseñar y construir un prototipo de un sistema de gestión de luminarias LED para la Empresa Eléctrica Quito utilizando una red de sensores inalámbricos (WSN) con 6LoWPAN para permitir una administración eficiente de las mismas.

1.3.2 Objetivos específicos

1 Analizar el estado del arte respecto al protocolo IEEE802.15.4 y los sensores disponibles que mejor se ajusten para la consecución del proyecto.

2 Analizar el proceso de comunicación de los sensores en relación a la utilización de 6LoWPAN y comparar con el sistema que usa IPv4, identificando ventajas y desventajas.

3 Diseñar y construir un prototipo de un sistema de sensores inalámbricos, utilizando dispositivos de estándar de comunicaciones 6LoWPAN, que formen una red local de monitoreo y registren el encendido-apagado de las luminarias. Para esto se pretende utilizar las herramientas de desarrollo y los dispositivos del fabricante de Libelium.

4 Diseñar e implementar una aplicación de software para PC que demuestre que se reciben los datos de los nodos, almacene las variables y procese la información. Se presentará en cuanto a las luminarias:

4.1 El estado en tiempo real, es decir encendida o apagada;

4.2 El total del tiempo de uso.

5 Analizar la factibilidad de la implementación del sistema diseñado en las luminarias LED instaladas por la Empresa Eléctrica Quito, estableciendo el costo del mismo. Para esto se analizará la situación actual de la instalación de las luminarias, proponiendo una solución específica tomando en cuenta todos los requerimientos tanto para la instalación de los sensores como para la transmisión de los datos.

CAPÍTULO 2

2. MARCO TEÓRICO

2.1 Redes de sensores inalámbricos WSN[11]

2.1.1 Definición

Una red de sensores inalámbricos (WSN, *Wireless Sensor Network*) es en particular un tipo de red que se compone de varios elementos autónomos denominados "nodos" interconectados entre sí inalámbricamente y cuyo propósito principal suele ser: recolectar mediciones de ciertos parámetros (temperatura, humedad, luminosidad, etc.) del lugar donde se encuentran ubicados, realizar algún tipo de procesamiento no muy intensivo y por último transmitir esta información hacia un dispositivo central que finalmente analiza los datos que permitirán obtener información importante en el área de estudio para la que se desplegó la red.

[11] Una red de sensores es una red de pequeños nodo equipados con sensores, que colaboran en una tarea común.

2.1.2 Componentes de una red WSN

Una red WSN está compuesta básicamente de [2]:

- Sensores: Obtienen del medioambiente la información requerida (temperatura, luz, sonido, humedad, etc.) y la transforman en señales eléctricas.
- Nodos (motes): Toman la información de los sensores, dependiendo del diseño realizan un procesamiento no muy intensivo y la transmiten a la estación base.
- Estación base: Es la encargada de recoger la información que le envían todos los nodos de la red.
- Gateway: Permite la interconexión entre la red de sensores y la red TCP/IP.
- Red Inalámbrica: Comúnmente basada en el estándar IEEE 802.15.4, en el presente proyecto se utiliza 6LoWPAN.

En la figura 2.1 observamos las redes de adquisición y de distribución de datos son sus diferentes variantes.

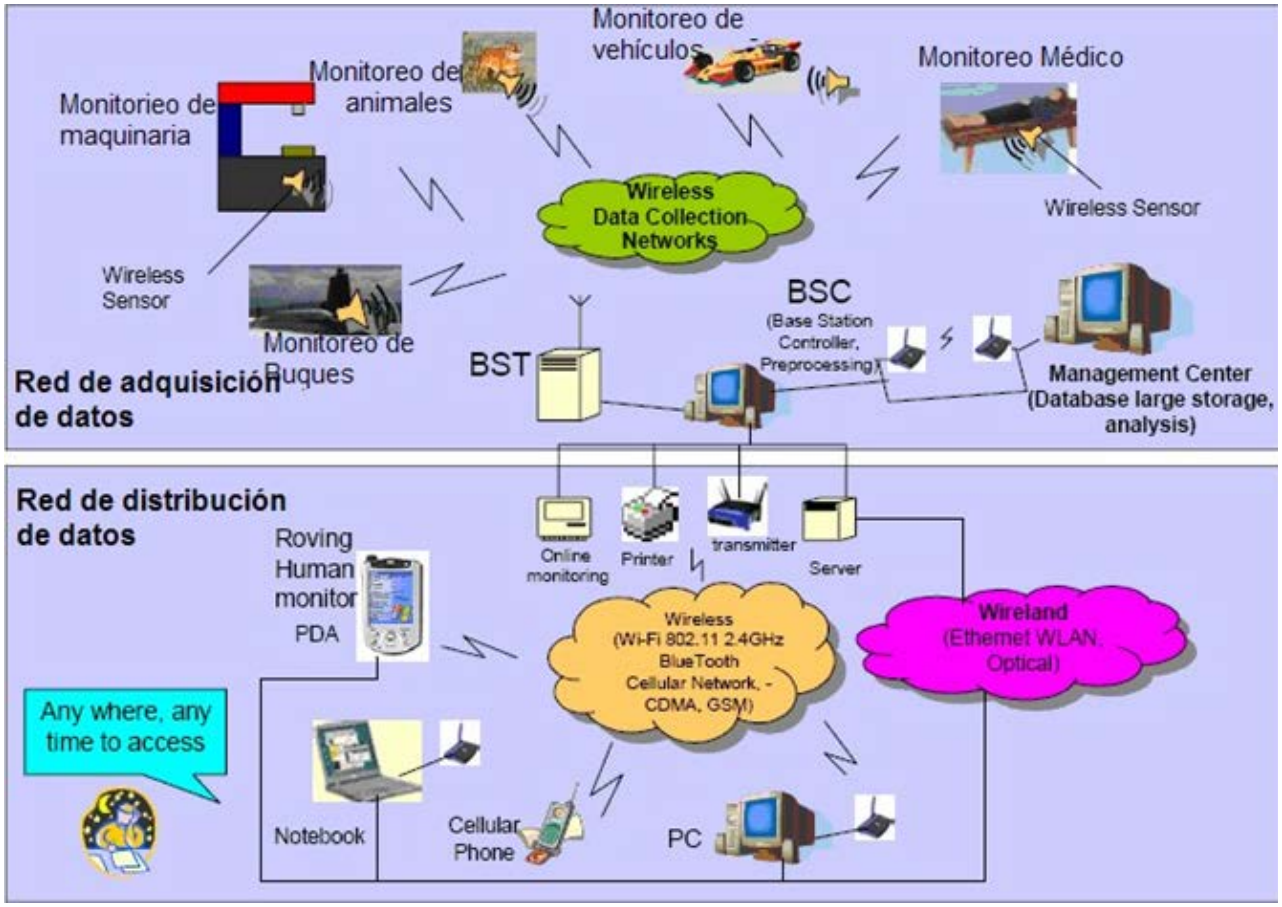


Figura 2.1: Redes de adquisición y distribución de datos de una WSN
Fuente: F. Zhao and L. Guibas, "Wireless Sensor Networks," 2004[3]

2.1.2.1 Sensores

Los sensores obtienen del medio ambiente la información requerida por la aplicación desplegada. Tenemos en la actualidad diversos tipos de sensores:

- De aceleración: la obtienen del objeto al que están unidos. Nos sirven para detectar impactos, medir vibraciones o conocer una inclinación.

- Presencia: pueden ser infrarrojos (en este caso detectan la interrupción de un haz) inductivos o capacitivos.
- Temperatura: transforman los cambios de temperatura en cambios de señales eléctricas. Pueden ser de tres tipos, termistores, RTD[12] y termopares.
- Sonido: Transforman la señal de audio en eléctrica.
- Otros: de humedad del aire, fuerza, proximidad, gas, radiación, fuerza motriz, etc.

En la figura 2.2 se observa algunos tipos de sensores que se pueden utilizar en las redes WSN.

Figura 2.2: Algunos tipos de sensores

Fuente: S. R. MAROTO CANTILLO, "Desarrollo de aplicaciones basadas en WSN," 2010

[12] RTD: *resistance temperature detector,* es un detector de temperatura resistivo, es decir que transforma la variación de temperatura en variación de resistencia de un conductor.

2.1.2.2 Nodos (motes)

Los nodos inalámbricos son conocidos como mote, por el hecho de que pueden llegar a ser bastante pequeños, mote en inglés es mota o lunar y hace referencia al tamaño del mismo. Captan la información del sensor, la procesan y la transmiten inalámbricamente. El parámetro principal de un mote no es únicamente su tamaño (no es suficiente con miniaturizar un computador personal), los requisitos son un tamaño reducido, un consumo muy bajo de energía y un costo también reducido. Además debe ser capaz de transmitir eficazmente los datos.

El nodo es un elemento con capacidad de procesamiento, memoria y una interfaz de comunicación. El hardware estándar de un nodo incluye un transceptor, el procesador, la memoria, la batería y la interfaz para conectar los sensores.[4]

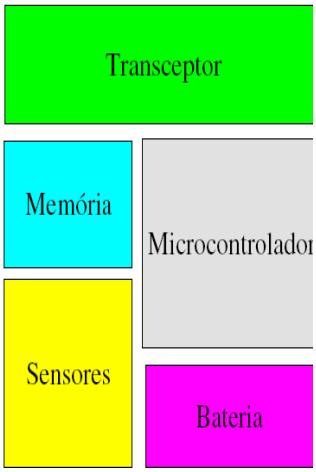


Figura 2.3: Esquema de hardware de un mote

Fuente: S. R. MAROTO CANTILLO, "Desarrollo de aplicaciones basadas en WSN," 2010

La capacidad de procesamiento está relacionada con el microprocesador que incluya dentro de su *hardware*. Posee una memoria interna en el microcontrolador y la comunicación se realiza a través del transceptor. Por otro lado, la fuente de alimentación varía de acuerdo a la tecnología con la que se ha fabricado la batería.

2.1.2.2.1 Nodos comerciales

- **Arduino**

Es una plataforma electrónica de código abierto basada en hardware y software “fácil de usar”, permite a los computadores capturar señales y controlar más cosas del mundo físico que un computador personal típico. El fabricante de Arduino se caracteriza por comercializar plataformas de hardware de uso múltiple y que se complementan con módulos que aportan más características.

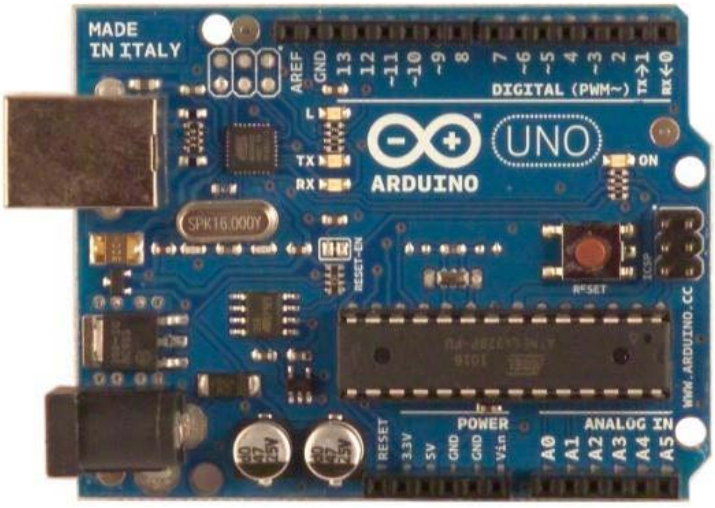


Figura 2.4: Arduino UNO[5]

Fuente: Arduino - ArduinoBoardUno.”,

https://www.arduino.cc/en/Main/ArduinoBoardUno.

Arduino UNO es una placa que tiene un puerto USB que sirve tanto para extraer la información como para cargar la batería.

Característica	Valor
Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz

Tabla 2.1 Características de Arduino UNO[5]

Fuente: Arduino - ArduinoBoardUno.",
https://www.arduino.cc/en/Main/ArduinoBoardUno.

Arduino tiene en el mercado varias opciones que se pueden ajustar al proyecto que tengamos que desarrollar, entre ellos placas, kits completos y accesorios.

- **Sensinode**

Se trata de un proveedor reconocido de soluciones para el IoT (*Internet of Things*). Se enfoca en la automatización de edificios, salud inalámbrica, logística, y otras aplicaciones más. Uno de los productos más reconocidos es el NanoStack, que es una plataforma y sistema de

radio independientes que forman un stack de comunicaciones para redes de sensores inalámbricas basadas en IP.

Sus características más destacadas son[6]:

RF interface support

2.4 GHz (IEEE 802.15.4)

Sub-1GHz (IEEE 802.15.4g)

Supported IEEE and IETF standards

6LoWPAN (ND, HC, RPL), UDPv6,

ICMPv6, TCP

Custom porting available

Self-healing Mesh network

Self-configurable

Support for Multicast forwarding

128-bit AES security support

Network processor and library versions

Support for 6LoWPAN Bootstrap and Link-local operation modes

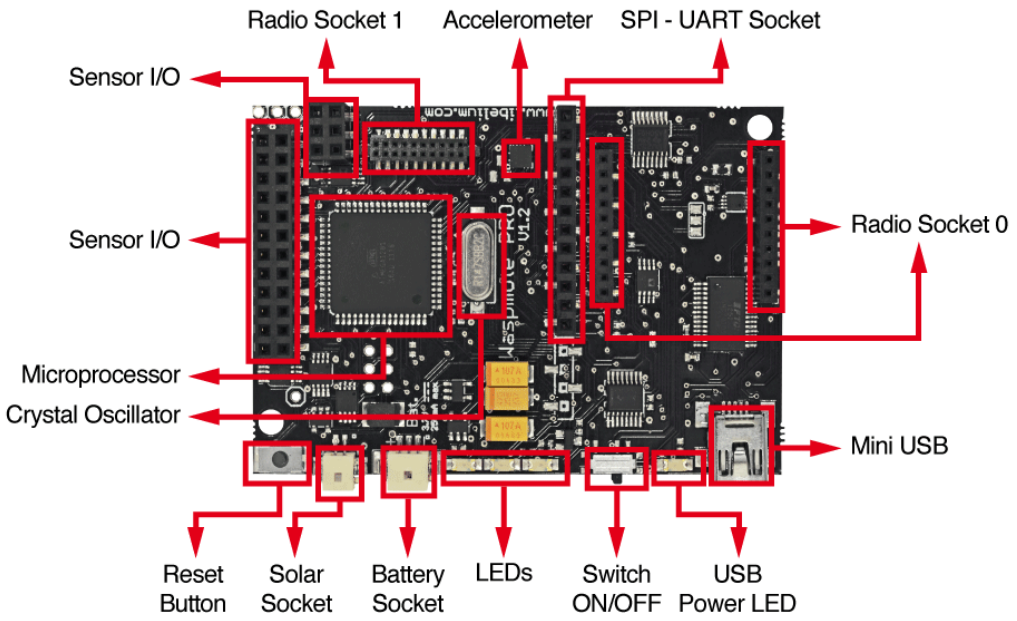
- **Libelium**

"Diseña y fabrica tecnología de hardware para la implementación de redes sensoriales inalámbricas y redes malladas para que los integradores de sistemas puedan llevar a cabo soluciones fiables a usuarios finales."[7]

Figura 2.5: Logo de Libelium[7]

Fuente: "Libelium - Connecting Sensors to the Cloud.", http://www.libelium.com/.

Su principal producto es el *Waspmote*, el mismo que puede integrar a más de 50 sensores diferentes.



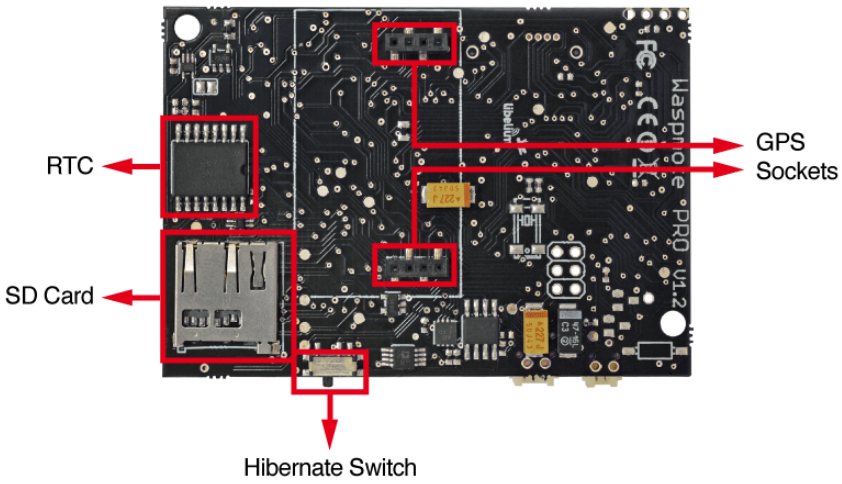


Figura 2.6: Placa base del *Waspmote*[8]

Fuente: "Waspmote Mote Runner - 6LoWPAN Development Platform - IPv6 for the Internet of Things and Sensors | Libelium.", http://www.libelium.com/products/waspmote-mote-runner-6lowpan/.

El *waspmote* es una plataforma orientada a desarrolladores. Puede trabajar con distintos protocolos (*ZigBee*, *LoRa*, *Bluetooth*, *GPRS, 6LoWPAN*) y frecuencia (2.4GHz, 868MHz, 900MHz) siendo capaz de mantenerse conectado a distancias de hasta 22km. Tiene un modo de hibernación de 0.06µA lo que permite ahorrar batería cuando no está transmitiendo.

Las características técnicas del *Waspmote* se muestran en la tabla 2.2.

Característica	Valor
Microcontrolador:	ATmega1281
Frecuencia:	14MHz
SRAM:	8KB
EEPROM:	4KB
FLASH:	128KB
Peso:	20gr
Dimensiones:	73.5 x 51 x 13 mm
Rango de Temperatura:	[-10ºC, +65ºC]
Reloj:	RTC (32KHz)
Consumo ON:	15mA
Consumo Sleep:	55uA
Entradas	7 Analógicas, 8 Digitales (I/O), 1SPI, 2 UARTs, 1 I2C, 1USB
Tensión de Batería	3,3 V – 4,2 V
Carga USB	5 V – 100 mA
Carga Placa Solar	6 – 12 V – 280 mA
Tensión Batería auxiliar	3 V
Sensores incorporados	Temperatura y Acelerómetro

Tabla 2.2: Características técnicas del *Waspmote*[8]

Fuente: "Waspmote Mote Runner - 6LoWPAN Development Platform - IPv6 for the Internet of Things and Sensors | Libelium.", http://www.libelium.com/products/waspmote-mote-runner-6lowpan/.

2.1.2.3 Estación base

La estación base puede ser simplemente un computador o un sistema embebido. Para nuestro prototipo utilizaremos un computador que será el encargado de recibir la información y procesarla mediante una aplicación de software para presentarla de acuerdo a los requerimientos.

2.1.2.4 Gateway

Para el caso del prototipo que vamos a implementar, el Gateway será un mote estándar con un módulo Ethernet adicional, tal como se muestra en la figura 2.7.

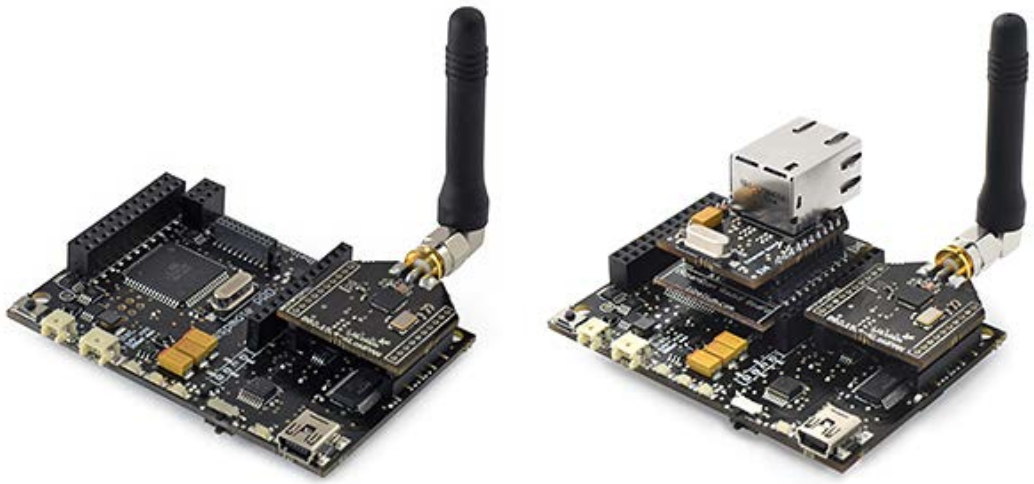


Figura 2.7: Nodo estándar (izquierda) y Gateway (derecha)[8]

Fuente: "Waspmote Mote Runner - 6LoWPAN Development Platform - IPv6 for the Internet of Things and Sensors | Libelium.", http://www.libelium.com/products/waspmote-mote-runner-6lowpan/.

Los nodos finales tienen sensores integrados y son usados para recolectar la información que será enviada al Gateway. Ellos crean una red mallada (*mesh*) entre sí, re-enviando los paquetes de otros nodos para que la información llegue al Gateway. Los nodos finales están equipados con un módulo 6LoWPAN, los sensores y la batería.

Los Gateway toman la información enviada por los nodos finales y la envían a través del túnel servidor IPv4/IPv6, usando su interface

Ethernet IPv4. Cada nodo Gateway está equipado con un módulo 6LoWPAN, un interface Ethernet IPv4 y la batería.

2.1.2.5 Red inalámbrica

La red inalámbrica de la estructura WSN en el caso de los motes Libelium es una red 6LoWPAN, puesto que estos son los módulos de radio que utilizaremos en la implementación del prototipo.

2.1.3 Características de una WSN

Las características que distinguen a una red WSN dentro de las redes inalámbricas son:[9]

- Movilidad de los nodos: Los nodos pueden o no estar en movimiento.
- Tamaño de la red: El número de sensores en la red puede ser mucho más grande que el de una típica red inalámbrica.
- Densidad de la red: La cantidad de nodos que cubran un área, puede depender, a más de la necesidad implícita de recabar información, del requerimiento de disponibilidad de la red. En las aplicaciones militares, por ejemplo, se requiere una alta disponibilidad de la red con un grado de seguridad alto y que exista redundancia de la información.

- Limitación de energía: este tipo de red, puede tener que operar en ambientes complicados, con poca o ninguna supervisión humana. Por esta razón el consumo de energía debe ser mínimo. Debido a que la única fuente de energía es una batería, la red tiene un tiempo de vida limitado, por lo que, se necesita un conjunto de protocolos muy eficientes a nivel de capa de red, de enlace de datos e incluso física para obtener un manejo eficiente de la energía.
- Fusión de datos e información: debido al ancho de banda y energía limitadas, la cantidad de bits y de información aumenta en los nodos intermedios. Para lograr fusionar los datos, se requiere usar más paquetes dentro de uno solo antes de su transmisión. Es decir, se reduce el ancho de banda utilizando encabezados redundantes en los paquetes y de paso se reduce el retardo al acceder al medio para transmitir.
- Distribución del tráfico: el patrón de tráfico varía de acuerdo a la aplicación de la red. En temas ambientales se genera un tráfico ocasional de pequeños paquetes, que requiere de poco ancho de banda. Otras aplicaciones, como por ejemplo la detección de intrusos en el ámbito militar, requiere transmisión en tiempo real.
- Escalabilidad: al no tener un Access point tradicional, la incorporación y eliminación de nodos debe ser un proceso sencillo y transparente al usuario.

- Seguridad: las redes son vulnerables a ataques pasivos o activos, pudiendo el atacante emular un nodo legítimo y capturar paquetes. Por esta razón se utilizan técnicas de encriptado como AES.

2.1.4 Topologías de las redes WSN

Existen básicamente tres clases de topologías que se pueden usar para implementar una red WSN y son: estrella, malla o una híbrida entre las dos.[9]

- Topología estrella: en esta arquitectura la información recolectada da únicamente un salto pues todos los nodos están en comunicación directa con el Gateway. En la figura 2.8 se observa la topología estrella.

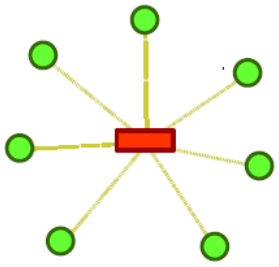


Figura 2.8: Topología en estrella

Fuente: S. R. MAROTO CANTILLO, "Desarrollo de aplicaciones basadas en WSN," 2010

Los nodos finales no intercambian información entre ellos y si se requiriese lo hacen a través del gateway.

- Topología en malla: este tipo de arquitectura es un sistema multisalto donde cada nodo puede enviar y recibir información de otro y de la puerta de enlace. En teoría con esta disposición la red podría tener una extensión ilimitada. En esta topología, la tolerancia a fallos aumenta pues cada nodo tiene distintos caminos para enviar o recibir información. Si un nodo falla la red se reconfigura alrededor del nodo fallido automáticamente. La figura 2.9 muestra la topología en malla.

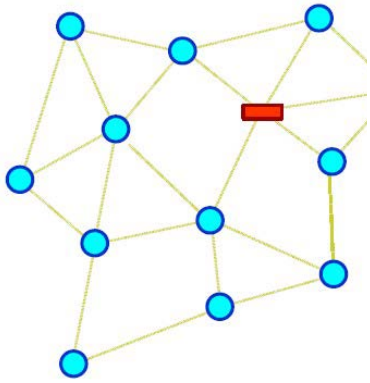


Figura 2.9 Topología en malla

Fuente: S. R. MAROTO CANTILLO, "Desarrollo de aplicaciones basadas en WSN," 2010.

- Topología híbrida: una solución intermedia a la hora de establecer una red WSN es utilizar routers. En este caso los nodos necesitan establecer comunicación directa con un router y de esta forma se encaminan los datos. En la figura 2.10 observamos las diferentes topologías.

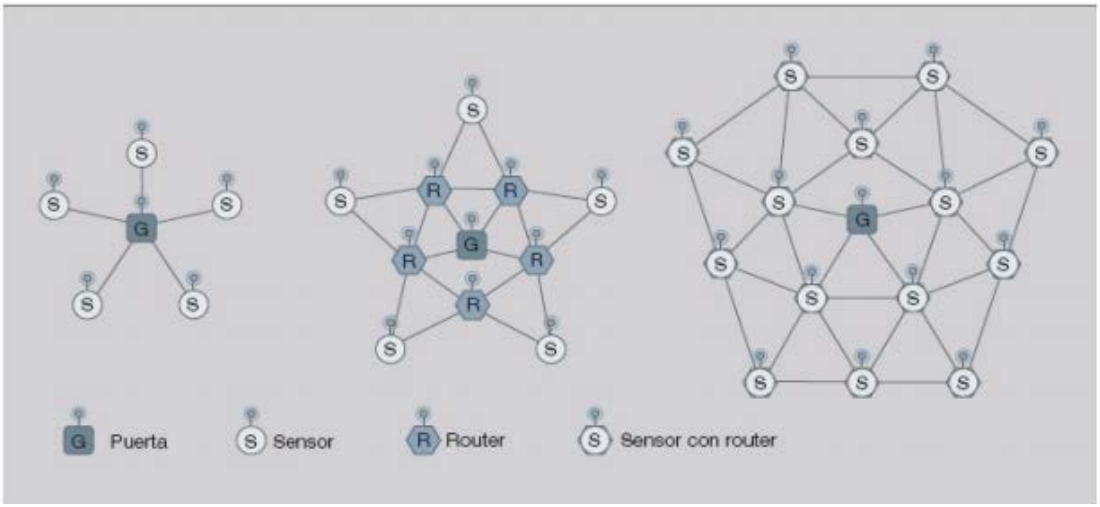


Figura 2.10 Topologías de las redes WSN

Fuente: S. Chatterjea, P. Havinga, and S. Dulman, "Introduction to Wireless Sensor Networks"

2.1.5 Aplicaciones de las redes WSN

De acuerdo al ámbito de la aplicación, estas se dividen en:

2.1.5.1 Militar

Varios autores coinciden en indicar que el origen de este tipo de redes se debe específicamente a las aplicaciones militares. La idea principal con estas aplicaciones es la de tener un conocimiento del campo de batalla en tiempo real que permita tomar decisiones adecuadas. Como un predecesor de las redes de sensores modernas se tiene a *Sosus* (*Sound Surveillance System*), que era una red de boyas sumergidas instaladas por los Estados Unidos para detectar a los submarinos

durante la Guerra Fría mediante sensores de sonido. La investigación en redes de sensores comenzó cerca de 1980 con el proyecto de DSN (*Distributed Sensor Networks*) de la agencia militar de investigación avanzada de Estados Unidos DARPA (*Defense Advanced Research Projects Agency*).

2.1.5.2 Medio ambiente

Las aplicaciones medioambientales son variadas, se usa este tipo de redes para detectar incendios forestales, deslaves, inundaciones y estudio de los animales en su hábitat natural, por nombrar unas pocas.

Se puede mencionar el caso de conservación de la fauna en la isla de Great Duck en Estados Unidos. En esta aplicación se pretende detectar la presencia de intrusos así como también poder estudiar el comportamiento de una especie de aves en particular.

En el 2002 el laboratorio de investigaciones de Intel en Berkeley en colaboración con el College of the Atlantic in Bar Harbor y la University of California en Berkeley, lanzaron un proyecto para desplegar esta red WSN. Concretamente el proyecto constó inicialmente de una red de 150 nodos que monitorean las condiciones ambientales de los nidos/refugios de las aves marinas Petreles (familia de los Albatros) y también de los alrededores.

Con esta implementación, un grupo de Biólogos del College of the Atlantic, pudieron observar la actividad de los Petreles en la isla, desde sus oficinas usando un enlace satelital. Los parámetros que registraron

fueron la cantidad de luz, la temperatura y la humedad. En la figura 2.11 se observa la ubicación de los sensores en la Isla de Great Duck.

Figura 2.11 Ubicación típica de los sensores en Great Duck[10]
Fuente: W. Staff, "The Ultimate on-the-fly Network," Dec-2003, http://www.wired.com/2003/12/network-2/.

2.1.5.3 Salud

En este ámbito se utilizan redes WSN para monitorear pacientes, diagnosticar enfermedades y administrar medicina entre otras. El cuidado de los adultos mayores es una aplicación en este campo que poco a poco se va haciendo popular. Estos pacientes requieren de un seguimiento exhaustivo, que incomoda tanto al paciente como al personal a cargo del mismo.

En este caso la red de sensores ubicados en puntos estratégicos del domicilio del adulto, en objetos de uso cotidiano e incluso en el propio

cuerpo, permite realizar el monitoreo en tiempo real de los signos vitales y el comportamiento del paciente.

Aparte de esto, el cuidado médico en hospitales también es aprovechado por esta tecnología.

El proyecto CodeBlue, desarrollado por la Universidad de Harvard implementa sensores para monitorear signos vitales como los latidos cardiacos, concentración de oxígeno en la sangre, datos EKG de los electrocardiogramas, etc.

Toda la información recolectada por los sensores se envía a un PDA o un computador portátil para su procesamiento.

2.1.5.4 Aplicaciones civiles

IrisNet es una arquitectura desarrollada por Intel en el 2002, teniendo como objetivo la gestión de redes mundiales de sensores de diversos tipos, permitiendo el acceso a estos sensores de una forma eficiente. Conforme la red es más grande, se requiere más ancho de banda, IrisNet ha permitido la implementación de distintas aplicaciones, tales como:

- Manejo de parqueaderos.
- Vigilancia de niños y adultos mayores con video.
- Monitoreo de redes de computadores.
- Observatorios terrestres y marítimos.
- Seguridad física.

2.2 Protocolos de comunicaciones

2.2.1 El estándar IEEE 802.15.4 y ZigBee

IEEE 802.15.4 fue creado para el desarrollo de las redes inalámbricas de baja velocidad y mínimo consumo de potencia. En este tipo de aplicaciones la velocidad de transferencia es baja aunque permite a los nodos de la red ser energizados usando baterías.

ZigBee es un grupo industrial formado para establecer las especificaciones que permitan aplicaciones inalámbricas fiables, económicas y de bajo consumo de energía basadas en la norma IEEE 802.15.4. En concreto ZigBee es la ampliación de 802.15.4 más difundida. El grupo ZigBee está compuesto por seis promotores principales (Honeywell, Samsung, Mitsubishi, Invensys, Motorola y Philips) y más de 80 participantes.[11]

2.2.1.1 **Características de ZigBee** [12]

- Menor consumo de potencia comparado con otros tipos de WPAN y que permite usar equipos alimentados por batería.
- Buena parte del tiempo los nodos permanecen en estado sleep con un ciclo efectivo de transmisión menor a 0.1%.
- Potencia de transmisión de 1mW.
- Alcance de 10 a 75 m. El típico es menor a 50m.
- Se pueden usar hasta 65534 nodos en una red.

- Tres bandas de comunicación: 868MHz (20Kbps), 915MHz (40Kbps) y 2.4GHz (250Kbps). La velocidad de transmisión típica es menor a 20Kbps.
- Usa CSMA-CA (*Carrier Sense Multiple Access Collision Avoidance*) para acceso al canal.
- Define las capas de red y de soporte a las aplicaciones.
- La Alianza ZigBee promueve el uso de las especificaciones y certifica los equipos.
- Existen varios estándares que se utilizan en redes de corto alcance, de acuerdo a la aplicación para la que están diseñados. En la actualidad ZigBee es el más aceptado para las redes de sensores de deben operar con batería. Sin embargo, estamos observando el aumento vertiginoso de redes implementadas con 6LoWPAN, el mismo que se analizará más adelante.

2.2.1.2 Tipos de tráfico

El tráfico se clasifica en tres tipos bien definidos:

- Datos periódicos (continuo): La tasa de datos está definida por la aplicación y su transmisión es constante. El ejemplo de este tipo de tráfico puede ser un sensor de temperatura que transmite sus lecturas cada 5 segundos.
- Datos intermitentes (por evento): La tasa de datos es variable y está controlada básicamente por estímulos externos. Por ejemplo

en un sistema domótico de control de luces, los interruptores transmiten únicamente en el cambio de posición, mientras esto no suceda se encuentran en el modo sleep con un consumo mínimo de energía.

- Datos periódicos usando ranuras de tiempo garantizadas (GTS, *Guaranteed Time Slot*): se usa para aplicaciones que necesitan comunicarse con prioridad de uso por el canal, pues son aplicaciones de baja latencia. GTS garantiza la comunicación dentro de un periodo de tiempo T por un cierto Δt utilizando lo que se conoce como súper trama. Para cumplir este objetivo se utiliza una forma de trabajo conocida como “con baliza” (*beacon*) implementando multiplexación en el tiempo.

2.2.1.3 Arquitectura

En la figura 2.12 observamos las capas de los protocolos 802.15.4 y ZigBee. Las dos capas inferiores, la capa física (PHY) y la capa de acceso al medio (MAC) están definidas en el estándar 802.15.4. Las capas de red (NWK) y de aplicación (APL) son definidas en ZigBee.

2.2.1.3.1 La capa Física

La capa física define aspectos como la potencia y la sensibilidad del transmisor.

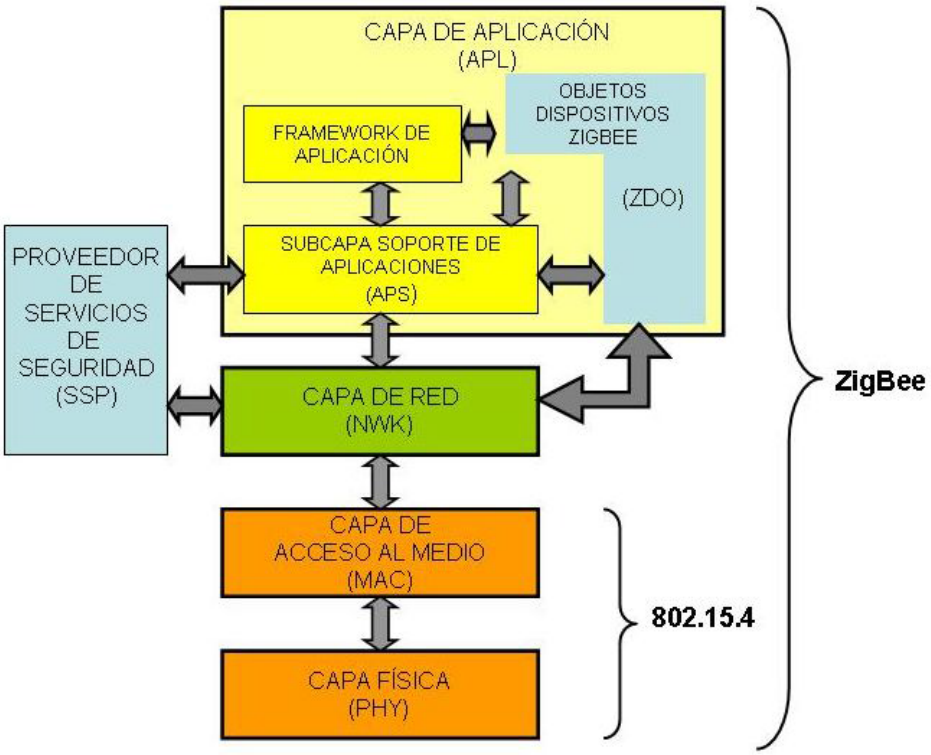


Figura 2.12: Capas de 802.5.14 y ZigBee[12]

Fuente: J. Dignani, "Análisis del protocolo ZigBee," 2012

El nivel físico PHY (*Physical Layer Protocol*) provee la transmisión de datos y además una interfaz con la entidad de gestión de nivel físico a través de la que se permite acceder a los servicios de gestión de nivel físico. La entidad de gestión se encarga de almacenar una base de datos con la información de redes relacionadas.

En la primera versión de 802.15.4 se definían 27 canales, distribuidos de la siguiente manera: 1 en la banda de 868MHz, 10 en la banda de 915MHz y 16 en la banda de 2.4GHz.

2.2.1.3.1.1 **Evaluación de canal libre** [13]

El mecanismo de acceso al medio CSMA-CA implica el que la capa MAC solicite a la capa PHY que evalúe el canal para comprobar que está libre. La evaluación del canal libre o CCA (*Clear Channel Assessment*) forma parte del área de administración de PHY. Existen tres modos de operación del CCA:

- Modo 1: previo a usar un canal, el dispositivo mide el nivel de energía en el mismo, obteniendo el valor medio de un intervalo correspondiente a ocho símbolos. Con este proceso se puede determinar un canal ocupado.

  La sensibilidad del receptor es la potencia mínima para que la señal pueda ser detectada y demodulada con un error menor al 1%. El estándar 802.15.4 permite una diferencia de 10dB entre la sensibilidad del receptor y el nivel mínimo de energía detectable y el rango de medición que exige es de 40dB. Por ejemplo si la sensibilidad es de -60dBm, debe poder detectar señales desde -50dBM y el intervalo de medición sería desde -50 a -10dBm.
- Modo 2: otra opción para determinar si el canal está ocupado es el sensado de portadora (CS*, Carrier Sense*), es decir se demodula la señal y se determina si esta es compatible con el estándar 802.15.4, y solo si es compatible se considera ocupado.
- Modo 3: una combinación del modo 1 y el modo 2 que incluye la aplicación de los dos criterios o de cualquiera de los dos.

2.2.1.3.2 La capa MAC

El protocolo IEEE802.15.4 establece especificaciones para las capas PHY y MAC y la que se ubique sobre esta última puede ser cualquiera de acuerdo a como lo establezca el protocolo usado. En el caso de ZigBee la capa que se ubica sobre la MAC es la de Red (NWK)

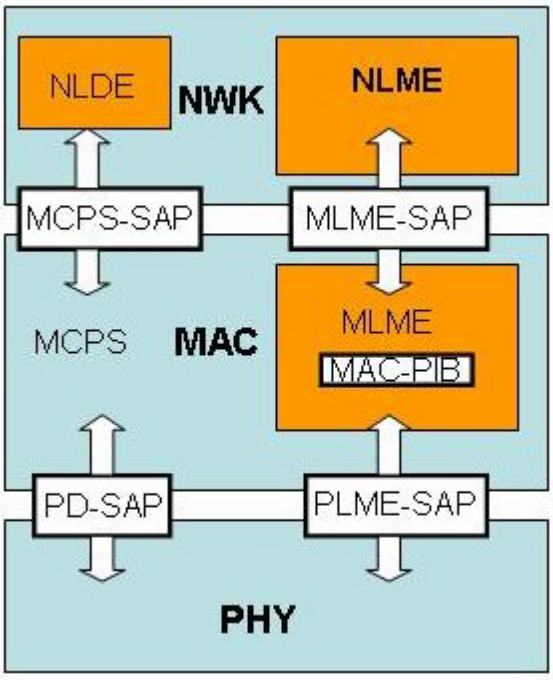


Figura 2.13: Interfaces entre la capa MAC y las capas vecinas[12]

Fuente: J. Dignani, "Análisis del protocolo ZigBee," 2012

En la figura 2.13 observamos la capa MAC y su relación con las capas Física y de Red. Los SAP[13] se encuentran igualmente divididos en los de datos y los de administración. La capa MAC tiene una base de datos

[13] SAP: *Service Access Point*, se dividen en datos y administración.

propia conocida como MAC-PIB (MAC – *Pan Information Base*). Esta base de datos almacena información de administración tal como:

- Duración de espera de "*Acknowledgment*" (comprobación del dato recibido).
- Permisos asociados.
- Respuesta automática de Datos.
- Opciones de extensión de vida de batería.
- Periodos de extensión de vida de batería.
- Carga de señal útil.
- Largo de carga de señal útil.
- Orden de señales.
- Tiempos de transmisión de señales.
- Número de secuencia de las señales.
- Coordinador de direcciones largas.
- Coordinador de direcciones pequeñas.
- Número de secuencia del dato.
- Opciones de permiso de GTS.
- Máximos intentos para CSMA.
- Identificador PAN.
- Direcciones cortas.
- Orden de "Superframe" (súper trama).
- Tiempo de transacción persistente.

Las funciones de la capa MAC son las siguientes:

- Puede generar balizas cuando se trata de un coordinador.
- Usa CSMA-CA como método que permite compartir el canal.
- Permite el manejo, sincronización y GTS en el modo de balizas.
- Permite un enlace seguro entre las MAC de dos dispositivos.
- Provee los servicios de asociación y des asociación.
- Provee un mecanismo de seguridad cuyo nivel estará determinado por las capas superiores.

2.2.1.3.2.1 Operación de la PAN usando balizas

El uso de las balizas es la forma como se garantiza la comunicación cuando la aplicación así lo requiera, es así como se puede disponer de ranuras de tiempo garantizadas (GTS). Para este efecto se han creado tramas especiales MAC conocidas como tramas de baliza y el uso de una estructura especial llamada súper trama que está separada por las tramas de baliza. En la figura 2.14 se observa la estructura de una súper trama.

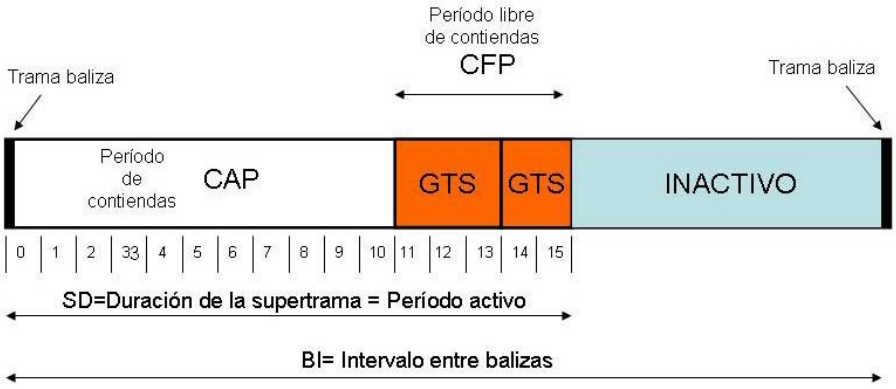


Figura 2.14: Estructura de las súper trama[12]

Fuente: J. Dignani, "Análisis del protocolo ZigBee," 2012

La súper trama se divide en tres partes, el período de contiendas CAP, el período libre de contiendas CFP y el período inactivo. Cuando los nodos desean transmitir en el período CAP, necesariamente tendrán que usar CSMA-CA para poder acceder a un canal que está disponible para cualquier dispositivo en igualdad de condiciones. El primero dispositivo en encontrarlo lo usa y lo tiene disponible hasta que termine su transmisión, si encuentra el canal ocupado entrará en una período de espera aleatorio (*back off*) y tratará de usarlo de nuevo. Las tramas MAC de comando se deben transmitir en el CAP. Por lo tanto no existe una garantía de que el dispositivo pueda transmitir en el CAP en el momento que lo necesite puesto que está en competencia con otros dispositivos.

En cambio cuando un dispositivo usa el CFP, tiene garantizada una

ranura de tiempo en la que no es necesario utilizar CSMA-CA, de hecho este período es el que se creó para que lo puedan utilizar las aplicaciones de baja latencia.

El período activo, como se muestra en la figura 2.14, está constituido por el CAP y el CFP y se divide en 16 ranuras de tiempo de igual tamaño. La baliza se muestra en la primera ranura y pueden existir hasta 7 GTS en el CFP. Cada GTS puede ocupar una o más ranuras.

La súper trama puede tener un período de inactividad, en el cual los dispositivos pueden apagar sus transceptores de radio para ahorrar energía. A estos nodos con el transceptor apagado se los conoce como nodos en modo *sleep*. El coordinador es quien determina los períodos de la súper trama dando valores a las constantes que definen el intervalo entre balizas (BI) y la duración de la súper trama (SD).

2.2.1.3.2.2 Espacio entre tramas

El espacio entre tramas es un tiempo de espera que hace el transmisor entre tramas de modo que el receptor tenga la oportunidad de procesarlas. A este tiempo se le conoce como IFS (*Interframe spacing*). Según el largo del MPDU se utiliza un IFS corto (*SIFS, Short IFS*) o uno largo (*LIFS, Long IFS*). Existen dos tipos de comunicación, con confirmación (*ACK: Acknowledge*) o sin ella.

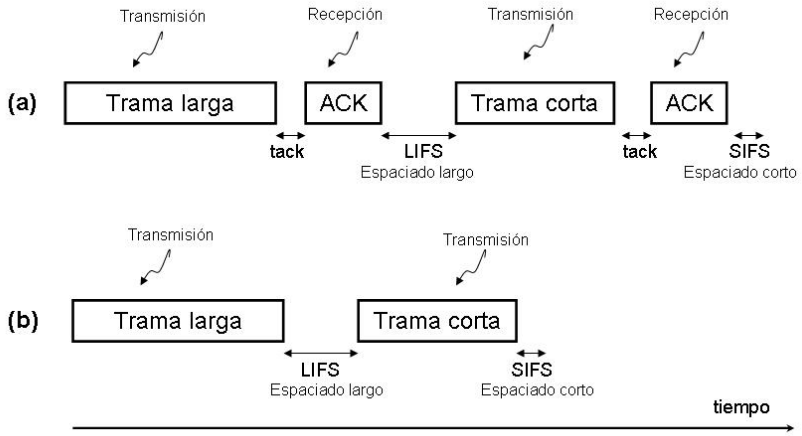


Figura 2.15: Espacio entre tramas: (a) con ACK, (b) sin ACK

Fuente: J. Dignani, "Análisis del protocolo ZigBee," 2012

En la figura 2.15 observamos que en el caso de usar comunicación con ACK, el IFS comienza luego de la recepción del ACK.

2.2.1.3.2.3 CSMA-CA

En general, cuando un dispositivo desea transmitir, debe verificar que el canal no esté siendo utilizado por otro dispositivo. También existen transmisiones que no necesitan realizar esta verificación. Estas son:

- Transmisión de balizas.
- Transmisión en el período CFP
- Transmisión después de haber dado ACK a un comando de pedido de datos.

El uso de CSMA-CA tiene en cuenta si se está utilizando súper trama o no. Cuando utilizamos súper trama el tiempo activo se divide en 16

ranuras de tiempo iguales, y por tanto el *back off* debe ser alineado de tal forma que no caiga en el CAP, esto recibe el nombre de CSMA-CA ranurado. En cambio cuando no utilizamos súper trama, no es necesario sincronizar el *back off* y hablamos de CSMA-CA no ranurado.

2.2.1.3.2.4 Nodo oculto y nodo expuesto

La presencia de un nodo oculto presenta problemas para el algoritmo de CSMA-CA.

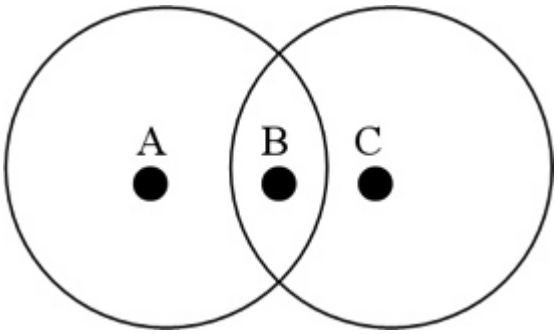


Figura 2.16: Nodo oculto[14]

Fuente: "Introducción a Zigbee y las redes de sensores inalámbricas." http://www.javierlongares.com/arte-en-8-bits/introduccion-a-zigbee-y-las-redes-de-sensores-inalambricas/

En la figura 2.16 se observa el problema del nodo oculto. El nodo B está dentro del rango de alcance de A y de C, pero ellos están fuera de alcance entre sí. Cuando A transmite algo a B, C no se entera y viceversa. Si en algún momento A y C transmiten simultáneamente y en el mismo canal, se creará una colisión de paquetes en B. Debido a que

en la MAC de 802.15.4 no hay un mecanismo de *handshake* que soporte RTS/CTS (*Request To Send/Clear To Send*), no existe una solución a este problema a nivel de software de MAC.[14]

El problema del nodo expuesto, se da por ejemplo si tuviésemos un nodo D al alcance solo de C. Cuando A y B estén comunicándose, C escucha los mensajes de B y por tanto considera que el canal está ocupado y no envía mensajes a D, aunque sí podría hacerlo. Este problema es menos grave que el del nodo oculto, pues solo influye en la latencia de las comunicaciones.

2.2.2 6LoWPAN

El estándar 6LoWPAN apareció con el objetivo de permitir el transporte de paquetes IPv6 sobre tramas IEEE 802.15.4. El utilizar 6LoWPAN en comparación con las soluciones tradicionales presenta ventajas que pueden ser resumidas en:

- El utilizar IP posibilita el uso de la red existente.
- Las tecnologías IP están funcionando en la actualidad y son bastante conocidas.
- Existen herramientas de diagnóstico y gestión de redes IP.
- Los dispositivos IP pueden conectarse directamente a la red sin necesidad de gateways o proxies, es decir estableciendo conexiones *end-to-end*.

- Es un estándar abierto.

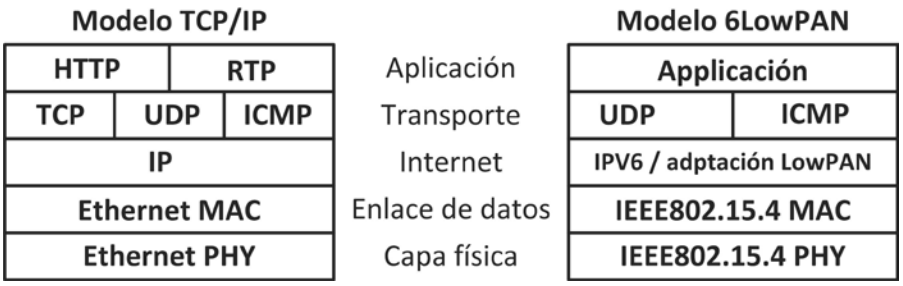


Figura 2.17: Modelos TCP/IP y 6LoWPAN[15]

Fuente: Ó. Vadillo Gutiérrez and others, "Provisión de servicios de la Internet de las Cosas sobre redes de sensores basadas en 6LoWPAN," 2014.

En la figura 2.17 se observa la comparación en los modelos TCP/IP y 6LoWPAN.

2.2.2.1 Dispositivos

En las redes 6LoWPAN podemos encontrar distintos tipos de dispositivos, como son:[15]

- El edge router, que se encarga de enrutar el tráfico entre la red IPv6 y la LoWPAN. Para que los protocolos IPv6 y 6LoWPAN interactúen se deben comprimir y descomprimir las cabeceras. También utiliza una versión modificada del protocolo de descubrimiento de vecinos ND (*Neighbor Discovery*) de IPv6.

- Los routers, que son nodos con capacidad de encaminamiento, es decir que pueden enviar paquetes al resto de nodos que están en su radio de alcance, como consecuencia están siempre encendidos.
- Los host, que son los dispositivos de menor capacidad de procesamiento y se usan como recolectores de datos o actuadores. No es necesario que estén siempre activos.

2.2.2.2 Topologías

En la figura 2.18 se observan las distintas topologías de LoWPAN.

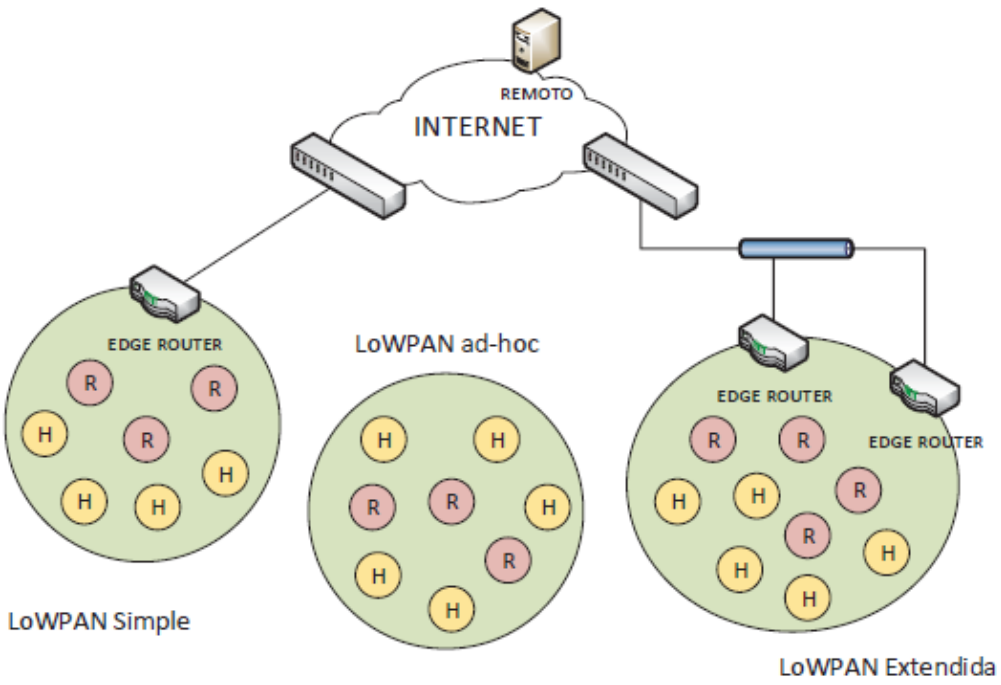


Figura 2.18: Topologías LoWPAN[15]

Fuente: Ó. Vadillo Gutiérrez and others, "Provisión de servicios de la Internet de las Cosas sobre redes de sensores basadas en 6LoWPAN," 2014.

Las topologías de LoWPAN se clasifican en:

- Simple: incluye un solo edge router.
- Extendida: tiene varios router de borde que comparten un enlace común. Se utiliza cuando se tienen muchos nodos en la red.
- Ad-hoc: no tiene router de borde pues no existe la necesidad de enrutar fuera de la red.

2.2.2.3 Estructura del paquete

El formato del paquete de 6LoWPAN se muestra en la figura 2.19. Puesto que el estándar encapsula paquetes IPv6 sobre tramas de enlace IEEE 802.15.4, se define la capa de adaptación que se encarga de realizar el proceso y de añadir cabeceras adicionales y también ofrece mecanismos como *mesh-under*, fragmentación o compresión de cabeceras de capas superiores.

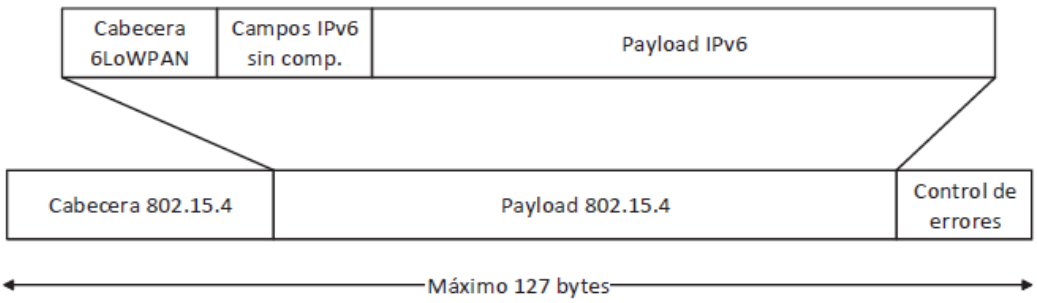


Figura 2.19: Formato del paquete 6LoWPAN[15]

Fuente: Ó. Vadillo Gutiérrez and others, "Provisión de servicios de la Internet de las Cosas sobre redes de sensores basadas en 6LoWPAN," 2014.

Las cabeceras de adaptación inician con un campo de 8 bits conocido como *dispatch byte*, el mismo que sirve para identificar el tipo de cabecera, tal como se muestra en la figura 2.20.

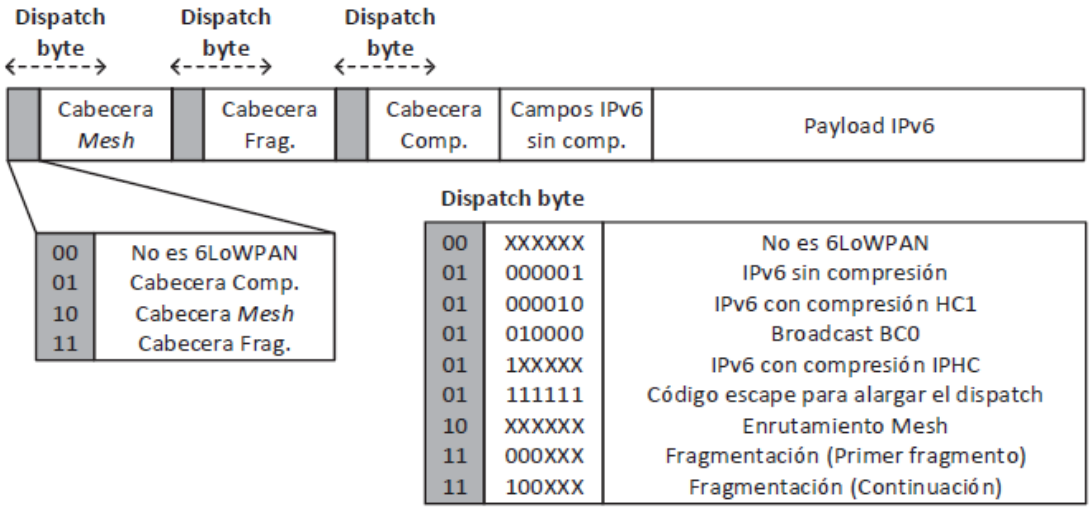


Figura 2.20: Cabecera 6LoWPAN[15]

Fuente: Ó. Vadillo Gutiérrez and others, "Provisión de servicios de la Internet de las Cosas sobre redes de sensores basadas en 6LoWPAN," 2014.

2.2.2.4 Direccionamiento mallado

Cuando se establece comunicación entre nodos de la misma red LoWPAN, la capa de adaptación se encarga del enrutamiento. Para esta situación se añade una cabecera llamada de malla o *mesh header*, la misma que contiene direcciones de capa MAC de origen y destino y el número de saltos. Los nodos se encargan de formar la trama MAC usando sus tablas de rutas con el objetivo de que llegue al destino con el menor número de saltos.

En la figura 2.21 se observa el formato de la cabecera.

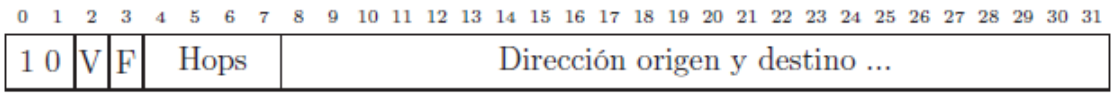


Figura 2.21: Cabecera mesh[15]

Fuente: Fuente: Ó. Vadillo Gutiérrez and others, "Provisión de servicios de la Internet de las Cosas sobre redes de sensores basadas en 6LoWPAN," 2014.

Los bits V y F indican el tipo de direcciones (64 o 16 bits) que se usan para el origen y destino. El campo *Hops* que se decrementa en cada salto, cuando llega a cero se descarta. La dirección de origen y destino que contienen las direcciones MAC.

Para tener la capacidad de difusión en la red, se antepone la cabecera de *broadcast* a la cabecera *mesh-under*. La cabecera *broadcast* se denomina BC0 y se muestra en la figura 2.22.

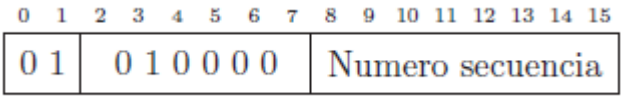


Figura 2.22: Cabecera broadcast BC0[15]

Fuente: Ó. Vadillo Gutiérrez and others, "Provisión de servicios de la Internet de las Cosas sobre redes de sensores basadas en 6LoWPAN," 2014.

El número de secuencia se incrementa por el origen del mensaje *broadcast* cada vez que envía un paquete.

2.2.2.5 Fragmentación y re ensamblado

Puesto que 6LoWPAN debe soportar la transmisión de tramas MAC IPv6 con una MTU (*Maximun Transmission Unit*) mínima de 1280 bytes y el tamaño de la trama de IEEE 802.15.4 es de 127 bytes, todos los paquetes 6LoWPAN que superen el tamaño de *payload* de esta última deberán ser fragmentados para su transmisión. En las figuras 2.23 y 2.24 se observa el formato de las tramas y cabeceras definido para implementar la fragmentación.

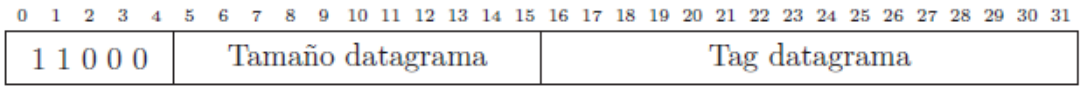


Figura 2.23: Cabecera del primer fragmento

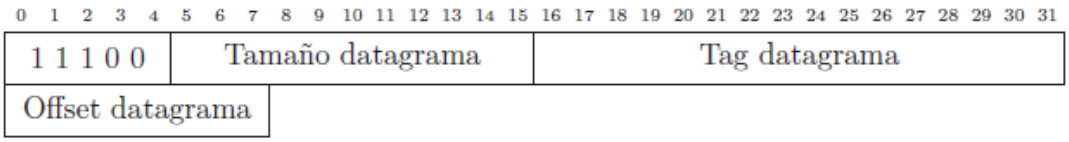


Figura 2.24: Cabecera de continuación de fragmentos[15]

Fuente: Ó. Vadillo Gutiérrez and others, "Provisión de servicios de la Internet de las Cosas sobre redes de sensores basadas en 6LoWPAN," 2014.

Para implementar el re ensamblaje del paquete se incluye el tamaño total del mismo en todos los fragmentos y además el desplazamiento

(*offset*) expresado en múltiplos de 8 bytes, además se indica la sección del paquete. Todos los fragmentos contienen la misma etiqueta para identificar al paquete original.

2.2.2.6 Compresión de cabeceras

La encapsulación directa de un paquete IPv6 sobre una trama IEEE802.15.4 no permite un rendimiento adecuado, pues la relación entre carga útil y cabeceras es muy baja. Esto se puede observar en la figura 2.25.

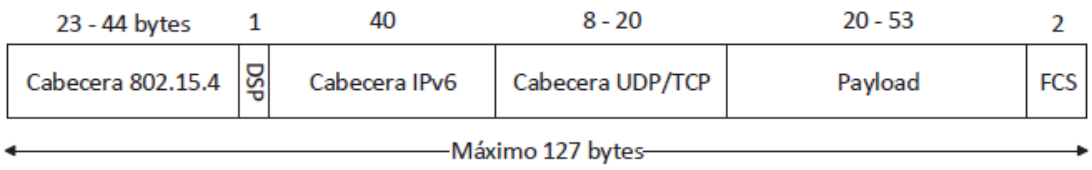


Figura 2.25: Encapsulamiento IPv6 sobre IEEE 802.15.14 sin compresión[15]

Fuente: Ó. Vadillo Gutiérrez and others, "Provisión de servicios de la Internet de las Cosas sobre redes de sensores basadas en 6LoWPAN," 2014.

La cabecera MAC de 802.15.4 ocupa 23 bytes usando direccionamiento EUI-64 y si se implementa seguridad AES-CCM*-128 se deben adicionar 21 bytes, llegando a ocupar 44 bytes. El FCS (*Frame Check Sequence*) que implementa corrección de errores usa 2 bytes.

El DSP (*Dispatch byte*) es un paquete de IPv6 sin compresión.

La cabecera de red IPv6 tiene un tamaño fijo (sin cabeceras de extensión) de 40 bytes.

La cabecera UDP ocupa 8 bytes y la TCP 20.

Considerando que la trama MAC tiene un tamaño se 127 bytes se puede obtener el payload a nivel de transporte y por tanto la eficiencia resultante en la transmisión de un paquete TCP y UDP sobre 6LoWPAN como se observa en la tabla 2.3.

	UDP		TCP	
	# Bytes	Efic. (%)	# Bytes	Efic. (%)
Sin seguridad	53	41.7	41	32.3
AES-CCM-128	32	25.2	20	15.7

Tabla 2.3: Tamaño de payload y eficiencia a nivel de transporte sin compresión[15]

Fuente: Ó. Vadillo Gutiérrez and others, "Provisión de servicios de la Internet de las Cosas sobre redes de sensores basadas en 6LoWPAN," 2014.

Al analizar los datos de la tabla resulta evidente que IPv6 no se diseñó para trabajar en redes IEEE 802.15.4, razón por la cual 6LoWPAN implementa mecanismos de compresión para los paquetes IPv6. Estos mecanismos se centran en comprimir la cabecera de red de IPv6 y la de transporte UDP, TCP no se ha tomado en cuenta puesto que es inusual usarlo en redes LoWPAN.

2.2.2.7 Compresión HC1-HC2

El primer método de compresión se fundamenta en la eliminación de redundancia de la cabecera IPv6 (HC1) usando la información de la cabecera MAC. Adicionalmente se usa compresión para la cabecera UDP (HC2). En la figura 2.27 se observa el formato de compresión utilizado.

Los campos que se incluyen en la cabecera IPv6 (tipo de tráfico, etiqueta de flujo, próxima cabecera, etc.) pueden ser eliminados por no ser estrictamente necesarios, a excepción del número de saltos. En ciertas ocasiones incluso los campos de direccionamiento se pueden comprimir, por ejemplo cuando el tráfico es interno a la red es suficiente con conocer las direcciones MAC de los nodos. Para mantener un registro de los procesos de compresión que se emplearon se añade una cabecera HC1 que se observa en la figura 2.27.

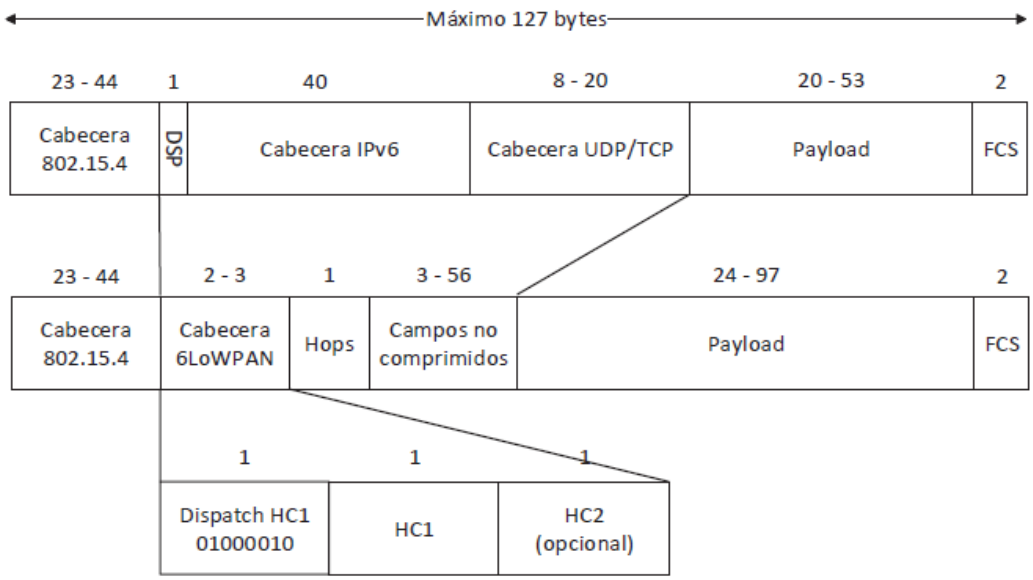


Figura 2.26: Compresión de cabeceras HC1-HC2[15]

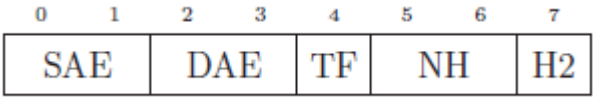


Figura 2.27: Cabecera HC1[15]

Fuente: Ó. Vadillo Gutiérrez and others, "Provisión de servicios de la Internet de las Cosas sobre redes de sensores basadas en 6LoWPAN," 2014.

SAE (*Source Address Encoding*): son los bits que indican el tipo de compresión de la dirección IPv6. El primero indica si se suprimió el prefijo de red y el segundo el identificador del dispositivo.

DAE (*Destination Address Encoding*): igual que el SAE pero con la dirección de destino.

TF (*Traffic Class and Flow Label*): Un bit que indica si se eliminaron los campos de Clase de tráfico y Etiqueta de flujo de la cabecera IPv6.

NH (*Next Header*): Dos bits que indican el tipo de cabecera que viene sobre IPv6. 00 desconocida, 01 UDP, 10 ICMPv6, 11 TCP.

H2: Un bit que indica el uso de compresión HC2 para el transporte de datos UDP, al mismo tiempo NH debe ser 01 UDP.

Cuando se utiliza la compresión HC2 se añade una cabecera con la información necesaria. En la figura 2.28 se observa la cabecera HC2.

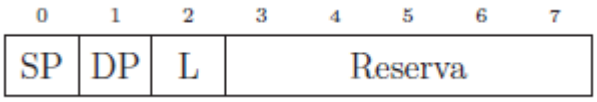


Figura 2.28: Cabecera HC2[15]

Fuente: Ó. Vadillo Gutiérrez and others, "Provisión de servicios de la Internet de las Cosas sobre redes de sensores basadas en 6LoWPAN," 2014.

SP (*Source Port*): Indica si el puerto de la fuente de la cabecera UDP se comprime de 16 a 4 bits.

DP (*Destination Port*): Igual que SP pero con el puerto destino.

L (*Length*): Si se suprime el campo de longitud de la cabecera UDP. Se puede obtener a partir del de la cabecera IPv6.

Reserva: Sin uso por ahora.

Los campos que no se comprimen van a continuación del campo de saltos. Los de IPv6 siguen el orden de la cabecera HC1 y los de UDP los de la cabecera HC2 (si se utiliza).

En la tabla 2.4 se observa el resultado de aplicar la compresión HC1-HC2.

	UDP		TCP	
	# Bytes	Efic. (%)	# Bytes	Efic. (%)
Sin seguridad	97	76.4	79	62.2
AES-CCM-128	76	59.8	58	45.7

Tabla 2.4: Tamaño de payload y eficiencia a nivel de transporte con compresión HC1-HC2[15]

Fuente: Ó. Vadillo Gutiérrez and others, "Provisión de servicios de la Internet de las Cosas sobre redes de sensores basadas en 6LoWPAN," 2014.

Para el caso de mejor eficiencia, usando UDP y sin seguridad a nivel de MAC, se obtiene 76.4%, lo cual quiere decir que de 4 bytes que se transmiten 3 son datos útiles a nivel de transporte. Se debe tomar en cuenta que este caso se presenta cuando se eliminan las direcciones IPv6, pero cuando un nodo se debe comunicar con uno externo, aparte de las direcciones MAC se debe conocer el prefijo de la red para poder encaminar el paquete, para poder comprimir los prefijos de red, apareció el mecanismo de compresión basada en contexto que se analiza a continuación.

2.2.2.8 Compresión basada en contexto IPHC-NHC

Este mecanismo reemplaza al HC1-HC2 aunque pueden funcionar en paralelo. Para la implementación, los últimos 5 bits del dispatch byte se usan para la cabecera IPHC. A diferencia de HC1-HC2, a continuación de las cabeceras agregadas se ubican los campos no comprimidos. En

HC1-HC2 se juntaban las cabeceras y luego iban todos los campos no comprimidos. En la figura 2.29 se observa el formato de compresión de las cabeceras en IPHC-NHC.

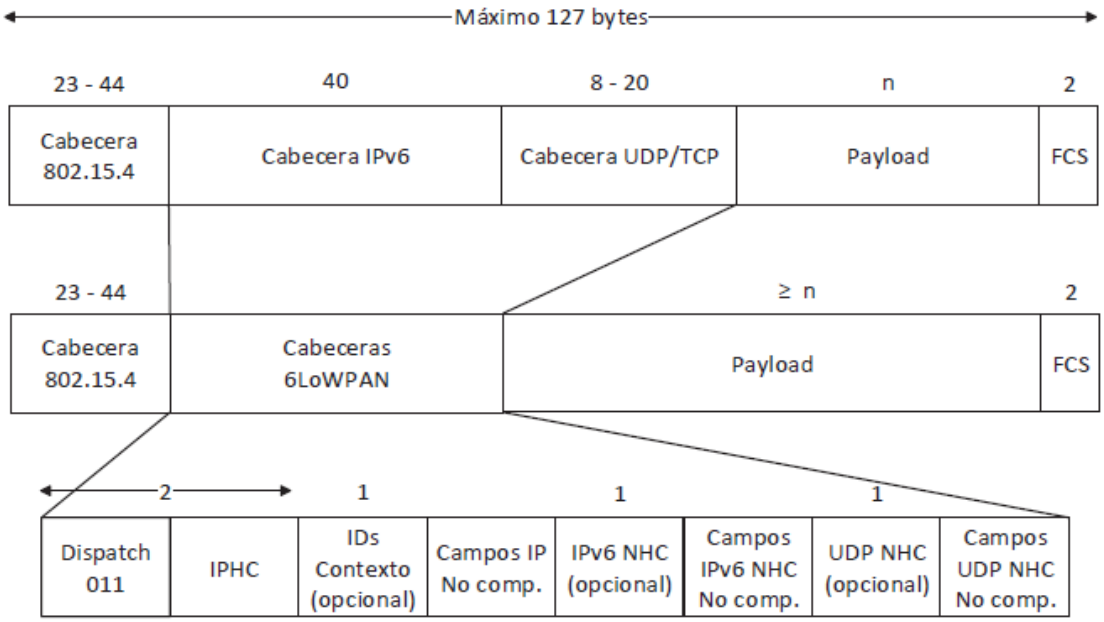


Figura 2.29: Formato de cabeceras compresión IPHC-NHC

Fuente: Ó. Vadillo Gutiérrez and others, "Provisión de servicios de la Internet de las Cosas sobre redes de sensores basadas en 6LoWPAN," 2014.

La cabecera IPHC contiene la información de cómo se comprimió la cabecera IPv6, el formato de esta cabecera se muestra en la figura 2.30.

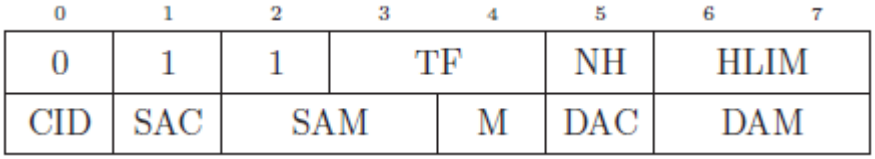


Figura 2.30: Formato de la cabecera IPHC[15]

Fuente: Ó. Vadillo Gutiérrez and others, "Provisión de servicios de la Internet de las Cosas sobre redes de sensores basadas en 6LoWPAN," 2014.

TF (*Traffic Class*, *Flow Label*): Indica si se ha suprimido el campo Clase de tráfico y/o Etiqueta de flujo de la cabecera IPv6.

NH (*Next Header*): Indica si se suprime el campo NH de la cabecera IPv6.

HLIM (*Hop Limit*): Indica si el campo Hops de la cabecera IPv6 se ha comprimido. Además se limita el contador a un valor máximo de 1, 64 o 255.

CID (*Context Identifier Extension*): Indica el uso del campo de extensión de identificador de contexto (opcional).

SAC (*Source Address Compression*): Indica si se va a usar compresión de la dirección de origen IPv6 stateful (basada en contexto) o stateless.

SAM (*Source Address Mode*): De acuerdo al SAC existen cuatro modos de compresión por cada tipo stateful o stateless, no comprimido, comprimido a 64 bits, comprimido a 16 bits o comprimido completamente.

M (*Multicast Compression*): Indica si la dirección destino IPv6 es multicast.

DAC (*Destination Address Compression*): Igual que SAC pero para el destino.

DAM (*Destination Address Mode*): Igual a SAM pero para el destino. Si el bit M está activado se usa la compresión adecuada a multicast (48,

32 u 8 bits).

La cabecera de extensión de identificador de contexto contiene información del contexto que se usa para la compresión del prefijo de red de las direcciones IPv6. En la figura 2.31 se observa el formato de esta cabecera.

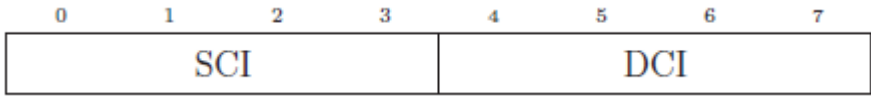


Figura 2.31: Formato de la cabecera IDs Contexto[15]

Fuente: Ó. Vadillo Gutiérrez and others, "Provisión de servicios de la Internet de las Cosas sobre redes de sensores basadas en 6LoWPAN," 2014.

SCI (*Source Context Identifier*): Indica el prefijo que se usa para la compresión basada en contexto de la dirección IPv6. Pueden ser hasta 16 contextos.

DCI (*Destination Context Identifier*): Igual a SCI pero para la dirección de destino.

Los campos IP no comprimidos van en el mismo orden que la cabecera IPv6. La cabecera opcional IPv6 NHC permite implementar mecanismos de extensión de IPv6, para esto del bit NH de la cabecera IPHC debe estar activado como muestra la figura 2.32.

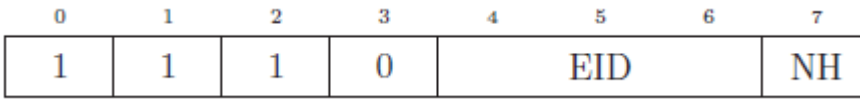


Figura 2.32: Formato de la cabecera NHC

Fuente: Ó. Vadillo Gutiérrez and others, "Provisión de servicios de la Internet de las Cosas sobre redes de sensores basadas en 6LoWPAN," 2014.

EID (*Extension Header ID*): Indica el tipo de información que va en el campo sin compresión de IPv6 NHC: opciones de enrutamiento, fragmento, salto, etc.

NH (*Next Header*): Indica si la cabecera próxima se comprime usando NHC.

Por último la cabecera UDP NHC permite la compresión de la cabecera UDP. El formato de la cabecera se muestra en la figura 2.33.

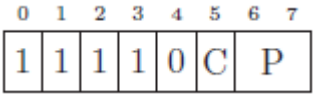


Figura 2.33: Formato de la cabecera UDP NHC[15]

Fuente: Ó. Vadillo Gutiérrez and others, "Provisión de servicios de la Internet de las Cosas sobre redes de sensores basadas en 6LoWPAN," 2014.

C (*Checksum*): Indica si se comprime el campo checksum de la cabecera UDP

P (*Ports*): Indica si los puertos de origen/destino se comprimen de 16 a

4 bits de la misma manera que en HC2.

En comparación con la compresión HC2, el campo de longitud debe ser comprimido siempre.

En la tabla 2.5 se muestran los resultados de la compresión de cabeceras IPHC-NHC cuando se establece comunicación con el exterior de la red LoWPAN. El payload y la eficiencia de byte a nivel de la capa de transporte.

		UDP		TCP	
		# Bytes	Efic. (%)	# Bytes	Efic. (%)
Sin seguridad	*Stateless*	98	77.2	79	62.2
	Stateful	97	76.4	78	61.4
AES-CCM-128	*Stateless*	77	60.6	58	45.7
	Stateful	76	59.8	57	44.9

Tabla 2.5: Tamaño de payload y eficiencia a nivel de transporte con compresión IPHC-NHC[15]

Fuente: Ó. Vadillo Gutiérrez and others, "Provisión de servicios de la Internet de las Cosas sobre redes de sensores basadas en 6LoWPAN," 2014.

2.2.2.9 Neighbor Discovery

El estándar 6LoWPAN define su propio protocolo ND puesto que el de IPv6 no es eficiente para estas redes.

El router de borde se encarga de distribuir el prefijo de red que todos los nodos usarán para generar sus direcciones IPv6, durante el proceso de *bootstrapping* que se observa en la figura 2.34. El host que pretende

formar parte de la red solicita al router información, entre los datos está el prefijo de la red. La solicitud llega a través de un mensaje RS (*Router Solicitation*) que se responde con un RA (*Router Advertisement*). Luego el host solicita el registro a través de un NR (*Node Registration*) en el que indica la dirección de la red con la que quiere identificarse, y que se ha generado por auto-configuración o manualmente. Inmediatamente el router debe comprobar si la dirección no está duplicada y responde con un mensaje NC (*Node Confirmation*).

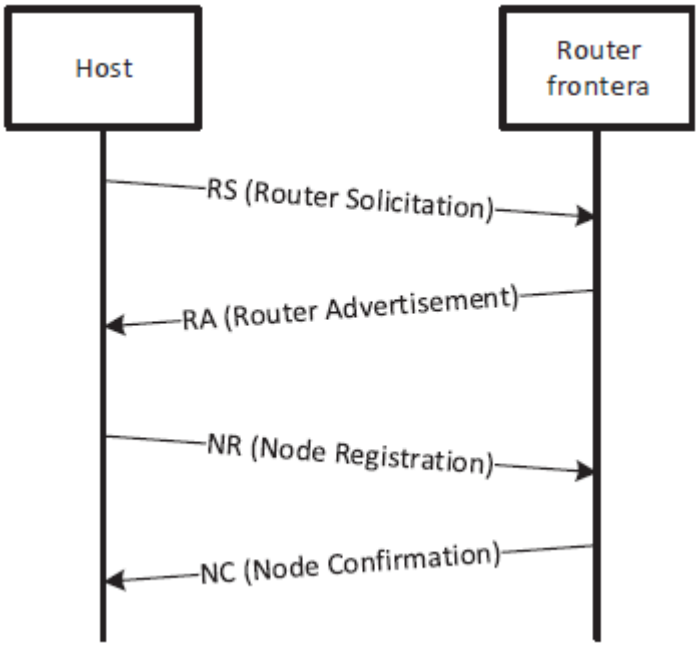


Figura 2.34: Bootstrapping 6LoWPAN-ND[15]

Fuente: Ó. Vadillo Gutiérrez and others, "Provisión de servicios de la Internet de las Cosas sobre redes de sensores basadas en 6LoWPAN," 2014.

Si el host no está al alcance de algún router de borde, un router de red actúa como intermediario en el proceso y propagan el prefijo de red. Es

posible que también estos router deban guardar la información si el host de destino está en modo sleep.

El ND modificado también implementa la eliminación de direcciones basadas en multicast con el fin de contribuir al ahorro de energía de los dispositivos y requerir que el dialogo host-router siempre se inicien por el host para facilitar el estado sleep de los dispositivos.

2.2.2.10 Enrutamiento

Debido a las limitaciones de energía y de procesamiento, el enrutamiento es uno de los puntos más críticos en 6LoWPAN. Existen varios protocolos dedicados a solucionar este aspecto, entre los cuáles tenemos:

- *Mesh-Under*: La capa de adaptación es responsable de manejar el enrutamiento a través de las direcciones de capa MAC. Para implementar esto 6LoWPAN usa la cabecera Mesh. El nodo que envía el paquete escribe en la cabecera de adaptación su dirección MAC y la del dispositivo destino, si la trama MAC llega a un dispositivo intermedio las direcciones de la cabecera se actualizan y se decrementa el número de saltos de la cabecera Mesh antes de ir al siguiente nodo. La desventaja de este enrutamiento es que cuando se pierde un fragmento se necesita retransmitir el paquete desde el nodo principal y mesh-under

tampoco implementa comunicación desde la red LoWPAN al exterior.

- *Router-Over*: La capa de red es responsable de manejar el enrutamiento. Cada salto a nivel MAC se considera igual a nivel IPv6. Si se fragmenta a nivel IPv6, los fragmentos se envían usando la información de las tablas de rutas. La capa de adaptación del siguiente salto comprueba los fragmentos recibidos y los junta para enviarlos a la capa de red. Si se pierde un fragmento se retransmite desde el nodo anterior. La desventaja es que se necesita un alto consumo energético producido por los dominios multicast de los enlaces IPv6.

Las dos opciones tienen desventajas que nos las hacen apropiadas para las redes LoWPAN. Luego de analizar los protocolos AODV (*Ad-hoc On demand Distance Vector*) y DYMO (*Dynamic MANET On.demand*) y concluir que tampoco eran adecuados, la IETF diseñó el RPL.

El objetivo principal de RPL es ofrecer caminos de enrutamiento eficientes para tres tipos de comunicaciones: Multipunto a punto, punto a multipunto y punto a punto. RPL usa tres tipos de mensajes que se encapsulan en paquetes ICMPv6:

- *DODAG Information Object* (DIO): Son enviados por los nodos RPL y sirven para informar sobre el estado del Destination

Oriented DAG (DODAG). Sirven para que un nodo descubra a sus vecinos y sus ranks.

- *Destination Advertisement Object* (DAO): Sirven para propagar la información del direccionamiento hacia los nodos RPL con rango superior. Las tablas de rutas de los nodos padres se actualizan.
- *DODAG Information Solicitation* (DIS): Sirven para solicitar mensajes DIO y descubrir los DODAG disponibles.

Es decir, RPL es un protocolo que permite una configuración eficaz de rutas incluso si la topología cambia, está definido para usarlo en redes de sensores y se utiliza a menudo en forma paralela a 6LoWPAN.

2.2.2.11 Seguridad

En el estándar se proponen distintos mecanismos de seguridad para cada una de las capas, en la MAC es posible usar encriptación basada en el cifrado de bloques AES. En la capa red se puede emplear *Internet Protocol Security* (IPsec) optimizada para 6LoWPAN para garantizar seguridad extremo a extremo. En transporte se recomienda *Transport Layer Security* (TLS), en su versión adaptada a UDP *Datagram Trasnsport Layer Security* (DTLS).

CAPÍTULO 3

3. DISEÑO E IMPLEMENTACIÓN DEL SISTEMA

Para cumplir con los objetivos propuestos en el presente trabajo se procedió a escoger una topología de red que se ajuste a las necesidades propias de la situación que se planea solucionar y por tanto, la aplicación que se implementará.

3.1 Topología de red

Para el caso que se analiza en el presente trabajo, es decir, la gestión de luminarias LED que se encuentran ubicadas en posiciones preestablecidas a ambos lados de la calzada bien sea peatonal o vehicular, la utilización de la topología por defecto de los nodos de Libelium es la adecuada. Los nodos usan el protocolo 6LoWPAN sobre la capa de enlace de 802.15.4 para crear una red mallada que interconecta cada dispositivo de la red con el Gateway (GW).

El Gateway en comparación con el resto de nodos no utiliza el estado *sleep* y se mantiene encendido todo el tiempo, es por esta razón que lo más recomendable es alimentarlo con una fuente que esté disponible siempre.

En la figura 3.1 observamos la topología de la red utilizada para la

implementación del prototipo del sistema.

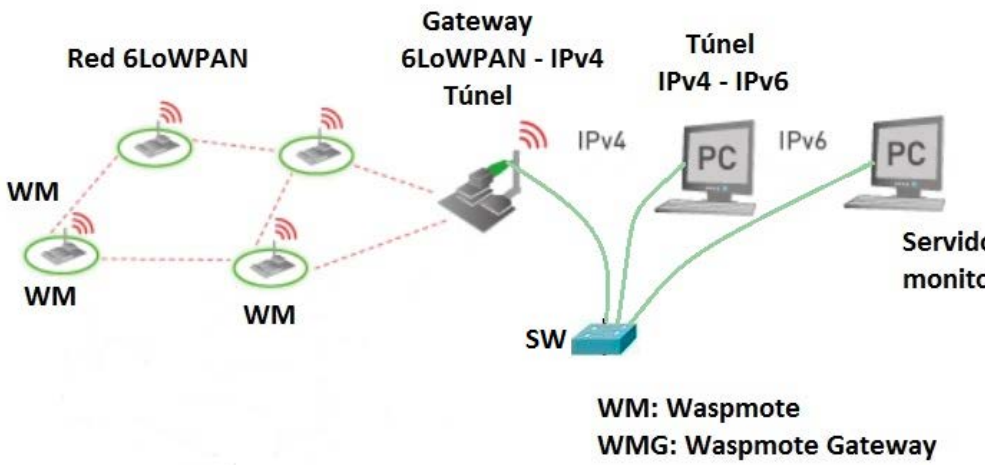


Figura 3.1 Prototipo de red para el sistema propuesto[16]

Fuente: Libelium Comunicaciones Distribuidas S.L., "Waspmote Mote Runner Technical Guide wasp mote," Tech. Guid., p. 85, 2015.

Se entiende que cada nodo inalámbrico está conectado a una luminaria LED y que se encargará de recolectar la información de estado on-off de la misma.

Los nodos de la red 6LoWPAN[14] se conectan con el Gateway que cambia la cabecera IP a IPv4 mientras mantiene la capa de transporte UDP, posterior a este proceso la información es enviada hacia la computadora que implementa el túnel IPv4 – Ipv6 que cambia la cabecera al formato IPv6 correcto y envía los datos hacia el servidor que procesa la información, la almacena y la muestra a través de la

[14] 6LoWPAN: *IPv6 over Low power Wireless Personal Area Networks*

aplicación de software desarrollada específicamente para el prototipo.

3.2 Diseño y justificación

3.2.1 Requerimientos

Uno de los principales objetivos del prototipo es implementar una red de sensores que utilice el protocolo 6LoWPAN. Tomando en cuenta esta característica se utilizarán los motes fabricados por Libelium puesto que permiten esta funcionalidad y proveen de mecanismos amigables con el diseñador, lo que nos permite implementar soluciones de código que cumplan con los requerimientos particulares. El utilizar la plataforma *waspmote* de Libelium implica utilizar una serie de herramientas que vienen incluidas en la misma y que se detallan a continuación.

3.2.1.1 Kit de desarrollo Waspmote Pro V 1.2 de Libelium

Para la implementación del prototipo de sistema propuesto con el fin de solventar las necesidades planteadas se ha utilizado el kit de desarrollo Waspmote Pro V1.2 producido en conjunto por Libelium e IBM.

“IBM y Libelium han unido esfuerzos para ofrecer una plataforma de desarrollo de IPv6 única para redes de sensores y el Internet de las Cosas (IoT). Mediante la integración de IBM Mote Runner SDK en la parte superior de la plataforma de sensores Libelium Waspmote, se

obtiene una herramienta única y poderosa para los desarrolladores e investigadores interesados en conectividad 6LoWPAN con IPv6 para el Internet de las Cosas” [16]

El kit de desarrollo, que se puede observar en la figura 3.2, contiene los siguientes elementos:

- 6 módulos 6LowPAN (2.4 GHz o 868 MHz)
- 6 baterías de 2300 mAh0
- 6 antenas (2dB para 2.4GHz, 0dB para 868Mhz)
- 1 módulo Ethernet
- 1 Programador Atmel AVR con su respectivo cable USB
- 6 cables mini-USB
- 6 adaptadores USB-110/220V

Figura 3.2 Kit de desarrollo Waspmote Mote Runner para 6LoWPAN[16]

Fuente: Libelium Comunicaciones Distribuidas S.L., “Waspmote Mote Runner Technical Guide wasp mote,” Tech. Guid., p. 85, 2015.

3.2.1.2 Computador con SO Linux Ubuntu 12.04 LTS de 64 bits

El computador que ejecuta el túnel IPv4/IPv6 tiene como sistema operativo la distribución de Linux Ubuntu 12.04 LTS de 64 bits, por recomendación de Libelium para la ejecución de la plataforma de simulación Mote Runner, así como para el desarrollo de aplicaciones para Waspmote Pro V1.2. El Mote Runner SDK nos permite la gestión de los nodos que formarán parte de la red.

3.2.1.3 Software de desarrollo MonoDevelop

Se trata de un IDE diseñado especialmente para C# y otros lenguajes como Nemerle, Boo, y Python en su versión 2.2. Se ha venido mejorando para los desarrolladores del Proyecto Mono[15]. MonoDevelop funciona con GNU/Linux, Windows y Mac, por lo que es considerado un IDE[16] multiplataforma.

3.2.1.4 SDK Mote Runner[17]

La IBM Mote Runner es a la vez una plataforma de ejecución y un

[15] Mono: es el nombre de un proyecto de código abierto iniciado por Ximian y actualmente impulsado por Novell (tras la adquisición de Ximian) para crear un grupo de herramientas libres, basadas en GNU/Linux y compatibles con .NET según lo especificado por el ECMA.

[16] IDE: *Integrated Development Environment*, al ambiente de desarrollo integrado es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software.

[17] SDK: *Software Development Kit*, es un conjunto de herramientas de desarrollo que permiten crear aplicaciones, en este caso involucradas con Mote Runner.

ambiente de desarrollo para redes de sensores inalámbricos (WSN) actualmente bajo el desarrollo en el Laboratorio de Investigaciones de Zurich de IBM. [16]

En la plataforma del mote el firmware Mote Runner provee de la plataforma de ejecución, la misma que incorpora una máquina virtual (VM, *Virtual Machine*[18]) para ejecutar el byte code[19] y un sistema operativo (OS, *Operating System*) para organizar el acceso a los diferentes dispositivos y para administrar las actividades de las aplicaciones.

El byte code es generado compilando una aplicación con el compilador Mote Runner y estas aplicaciones puedes ser escritas en C# o Java. Existe la posibilidad de utilizar un plugin de Eclipse para depurar las aplicaciones y se pueden ejecutar, bien sea en nodos simulados en el PC o en los nodos físicos del kit de desarrollo.

La VM del nodo permite la instalación o eliminación de las aplicaciones en tiempo de ejecución. Para controlar las motas se dispone de una línea de comandos llamada MRSH[20] (*Mote Runner Shell*) y los mensajes que envían las motas pueden ser visualizadas en el servidor WEB. También es posible realizar el desarrollo de aplicaciones off-

[18] VM: *Virtual Machine*, es un software que simula una computadora y puede ejecutar programas como si lo fuese.
[19] Byte code: es un código intermedio más abstracto que el lenguaje de máquina. Cada código de operación tiene la longitud de un byte, de ahí su nombre.
[20] MRSH: *Mote Runner Shell*, es la línea de comandos que nos permitirá ejecutar, grabar y depurar aplicaciones en las motas simuladas o físicas.

mote.

3.2.1.5 Librerías MRv6 de Mote Runner

La versión de 6LoWPAN implementada en Mote Runner es conocida como MRv6 y esta librería debe ser instalada en los nodos que se desea se comunicar usando IPv6. MRv6 es multi-saltos TDMA[21] (*Time Division Multiple Access*).

En la red 6LoWPAN del prototipo se usa la compresión de cabecera IP (IPHC). MRv6 está escrito en C#, empaquetado en *asembly's*[22] y se instala a través de Mote Runner. MRv6 funciona sin problemas cuando las motas envían periódicamente datos a un host remoto y raramente se actualizan.

En el Gateway se instala el MRv6 edge, que es el que permite la administración de la red, las solicitudes de asociación, el nodo que puede transmitir y las rutas de la red. Es posible desde la aplicación MRSH comunicarse con los nodos y recuperar datos desde los mismos.

MRv6 permite también la implementación del túnel y por tanto asignar los paquetes IPv6 a 6LoWPAN y viceversa. El túnel implementa una

[21] TDMA: *Time Division Multiple Access*, multiplexación por división de tiempo, es una técnica que permite la transmisión de señales digitales y cuya idea consiste en ocupar un canal de transmisión a partir de distintas fuentes, de esta manera se logra un mejor aprovechamiento del medio de transmisión.

[22] Los *assembly* pueden ser incluidos directamente por Mote Runner o ser generados por el compilador, son el bytecode que la máquina virtual (VM) puede ejecutar dentro del mote.

interfaz de red virtual de forma que los nodos son identificados utilizando direcciones IPv6.

3.2.1.5.1 Protocolo LIP

Cuando deseamos configurar un nodo por medio de un cable serial o USB se usa el protocolo LIP. En el mismo se usan tramas de datos simples, con una cabecera y carga útil. Los primeros 4 bytes incluyen una dirección de host, los dos siguientes el puerto UDP y el último es el puerto LIP de la mota. Los primeros 64 puertos están reservados para los *assembly's* de direccionamiento.

3.2.1.5.2 Protocolo WLIP

Se trata de la versión inalámbrica de LIP, es de salto simple y por consumir mucha energía en la mota, se lo utiliza para configuraciones básicas como carga de *assembly's* o librerías.

3.2.1.5.3 Limitaciones de MRv6

Debido a las propias limitaciones de hardware de los nodos, el MRv6 también tiene limitaciones que se resumen a continuación:

- MRv6 permite únicamente la transmisión de paquetes UDP dentro de la red 6LoWPAN. Los paquetes (ICMP) *Internet Control Message Protocol* se manejan inmediatamente por el túnel y no se distribuyen al borde de la red por razones de eficiencia.
- TCP y otros protocolos no UDP no son compatibles y tampoco multicast de IPv6.
- En aplicaciones de baja latencia no se debe usar MRv6 debido a que los tiempos para la comunicación de los padres con un nodo hijo en el árbol se asignan a nivel global y no se superponen, es decir, depende del tamaño de la red y por lo tanto se presentan retrasos significativos.
- El protocolo podrá ser desplegado en los rangos de frecuencia de 2.4 GHz o 900Mhz. En 2.4 GHz se utiliza un solo canal.
- La implementación de MRv6 no provee de segmentación de paquetes ni el re ensamblaje y control de flujo. Cuando se necesitan estas características, se las debe implementar por las aplicaciones de capa superior o a nivel de usuario.
- MRv6 está limitado para la configuración automática sin estado o s*tateless*: se utiliza la EUI-64 de 64 bits del identificador único de la mota para formar la dirección IPv6 de 128 bits.
- Las capacidades de enrutamiento son limitadas. El túnel se limita actualmente a una interfaz de red IPv6 de enlace local.

3.2.1.5.4 Utilizando MRv6

Para configurar una red MRv6 se instalan en los nodos la biblioteca de red 6LoWPAN y la o las aplicaciones de usuario. En el nodo Gateway en cambio se instala el *assembly* de borde. Los mensajes WLIP/LIP pasan por el túnel MRv6.

Una vez instalados los *assembly's* MRv6 se puede configurar varios parámetros como los tamaños de búfer, los tiempos, el número de nodos, la profundidad de la red. Algunas configuraciones se pueden hacer en el Gateway y este se encarga de replicarlas en cada nodo que se une a la red y otras configuraciones se tienen que hacer individualmente en cada nodo.

Para iniciar la red se suele utilizar el MRSH (*Mote Runner Shell*) con ciertos comandos que se ingresan a través de la computadora.

Para que se puedan transmitir los paquetes IPv6 a la red MRv6, se debe levantar el túnel. El túnel implementa una interfaz de red virtual en conjunto con Linux para los host conectados a un nodo *edge*, el mismo que mapea paquetes IPv6 a 6LoWPAN y viceversa. El túnel permite que el mapeo ocurra de forma transparente para el host y las aplicaciones de los nodos, de esta forma los host y nodos pueden ser alcanzados e identificados usando direcciones IPv6. El túnel es neceario únicamente si se requiere comunicación con aplicaciones externas basadas en IPv6. El Gateway se encarga de enrutar los

paquetes dentro de la red MRv6 y reenvía los paquetes dirigidos a una dirección externa al túnel.

3.2.1.6 Servidor de monitoreo

El servidor de monitoreo está implementado en un computador con SO Windows 7 conectado dentro de la red IPv6 y aloja una base de datos SQL que es donde los nodos escriben los datos de lectura del estado de encendido o apagado de las luminarias. El servidor además ejecuta la aplicación de gestión de las luminarias diseñada para cubrir con las necesidades propuestas para el presente trabajo, la misma que está escrita en C# y será analizada más adelante.

3.3 Implementación del prototipo

Para implementar el prototipo de red que se plantea para la aplicación de gestión de luminarias LED se deben seguir varios pasos que se explican en los siguientes apartados.

3.3.1 Instalación del firmware en los nodos

El *firmware* es el ambiente de ejecución "*on-mote*" y contiene librerías estándar para el funcionamiento de los nodos. Antes de utilizar los

nodos es necesario cargar el *firmware* en ellos, este proceso se lo realiza con la ayuda del software AVR y con el programador Atmel AVR con su respectivo cable USB que vienen suministrados en el kit. El software AVR que se utiliza en el presente trabajo es AVRdude y se descarga desde el Centro de Software de Ubuntu.

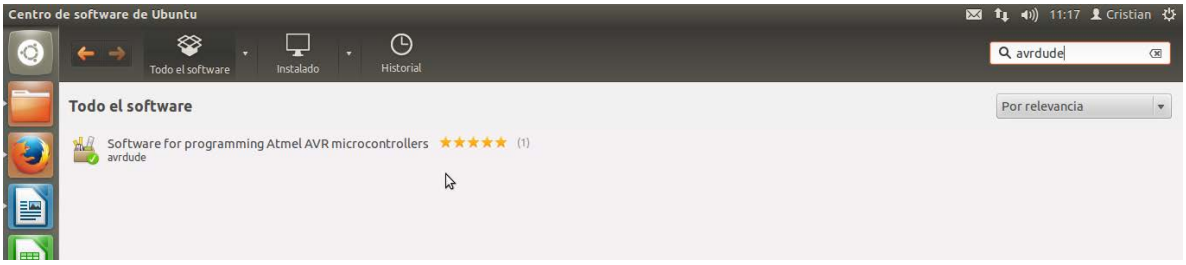


Figura 3.3 AVRdude en el Centro de software de Ubuntu

Para proceder a la instalación del *firmware* se debe localizar el archivo que lo contiene (waspmote.hex). Este archivo se encuentra en el directorio de instalación de Mote Runner y que, por recomendación del fabricante, debe ser en la raíz del sistema de archivos.

En resumen, para instalar el *firmware* en cada nodo se requiere del *hardware* y *software* que se lista a continuación:

- El waspmote pro, es decir el mote.
- El programador AVR con conector ISP
- El *software* de programación, por ejemplo el AVRdude
- El *firmware* Mote Runner que se encuentra en <directorio mote runner>/firmware/waspmote.hex

El proceso para escribir la imagen hex es el siguiente:

1. Conectar el *waspmote* a una fuente de poder (por ejemplo a una PC con el cable USB). Encender el *waspmote* moviendo el switch correspondiente hacia la izquierda como se muestra en la figura 3.4.

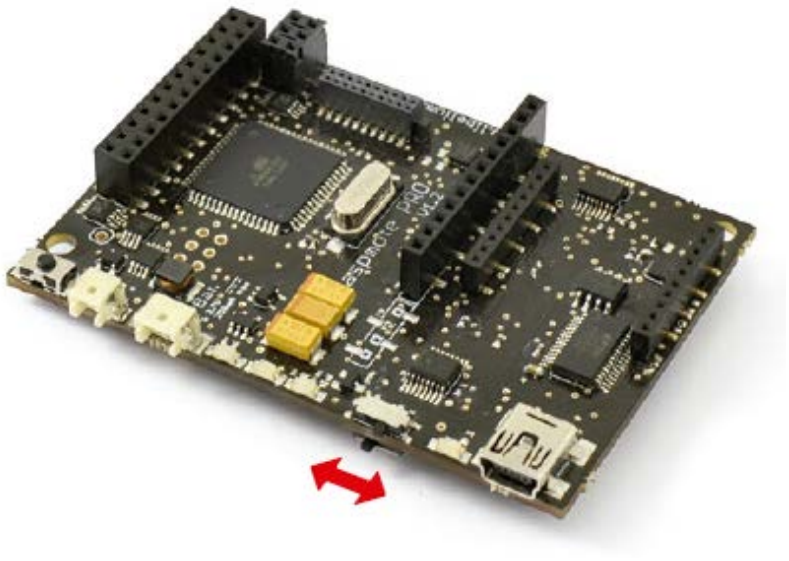


Figura 3.4 Encendido del waspmote[16]

Fuente: Libelium Comunicaciones Distribuidas S.L., "Waspmote Mote Runner Technical Guide wasp mote," Tech. Guid., p. 85, 2015.

2. Conectar el programador AVR al PC con el cable USB respectivo como se muestra en la Figura 3.5.

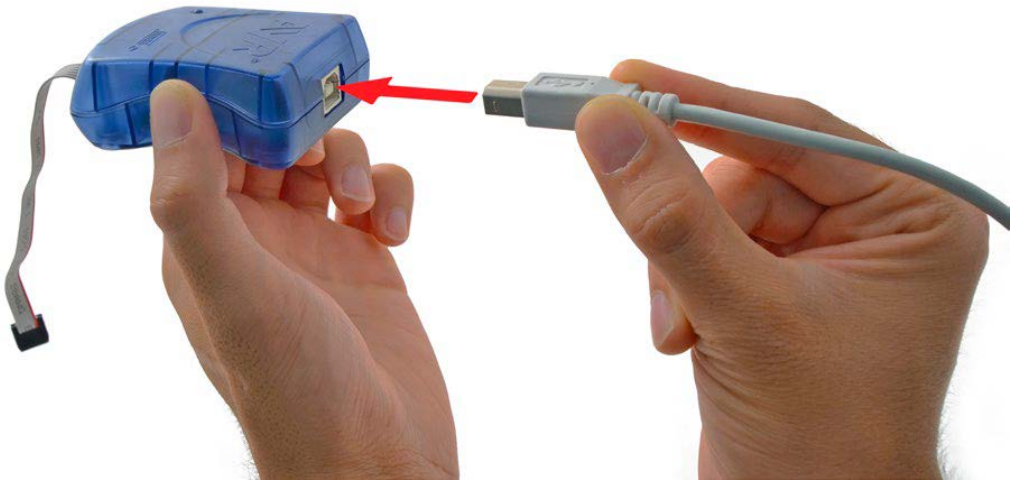


Figura 3.5 Conectando el programador AVR a la PC[16]

Fuente: Libelium Comunicaciones Distribuidas S.L., "Waspmote Mote Runner Technical Guide wasp mote," Tech. Guid., p. 85, 2015.

3. Conectar el programador AVR al puerto ICSP ubicado en la parte posterior del *waspmote* como se muestra en la figura 3.6.

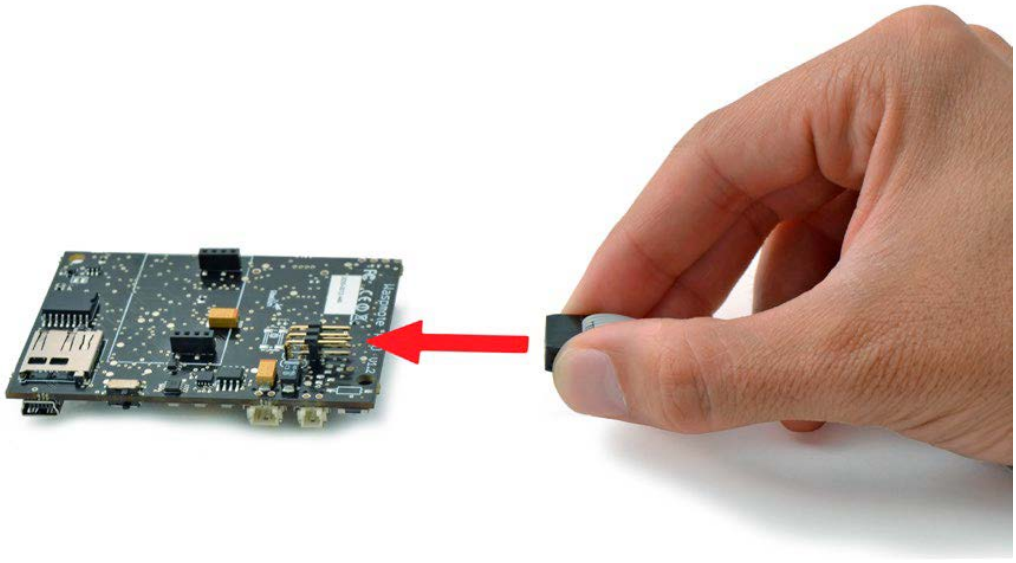


Figura 3.6 Conectando el programador AVR al waspmote[16]

Fuente: Libelium Comunicaciones Distribuidas S.L., "Waspmote Mote Runner Technical Guide wasp mote," Tech. Guid., p. 85, 2015.

4. El último paso corresponde a escribir los comandos específicos en un terminal de Ubuntu.

   Para configurar los “fusibles”:

   sudo avrdude -c avrispmkII -p m1281 -P usb -b 115200 -e -u -U efuse:w:0xFF:m -U hfuse:w:0xD0:m -U lfuse:w:0xFF:m

   Y para cargar la imagen del *firmware*:

   sudo avrdude -c avrispmkII -p m1281 -P usb -b 115200 -U flash:w:/moterunner/firmware/waspmote.hex

Una vez instalado el *firmware* en cada mote se puede instalar los *assembly's* en los mismos, para este proceso ya no es necesario el programador AVR pues es suficiente el uso del cable USB y de comandos específicos dentro del SDK Mote Runner. Los *assembly's* se crean luego de utilizar el compilador mrc (*Mote Runner Compiler*) al código fuente escrito. El mrc genera tres archivos con extensiones .sdx, .sxp y .sba, este último archivo es el que se debe instalar en el mote y que será ejecutado por el mismo.

3.3.2 Servidor MRSH

El *Mote Runner Shell* es un servidor web que provee del soporte necesario para administrar la red 6LoWPAN.

Luego de instalar el SDK Mote Runner es necesario realizar la exportación de las variables de entorno de acuerdo a la figura 3.7.

```
export PATH=~/moterunner/linux64/bin:$PATH
export LD_LIBRARY_PATH=~/moterunner/linux64/bin:$PATH
```


Figura 3.7 Exportación de variables de entorno[16]

Fuente: Libelium Comunicaciones Distribuidas S.L., "Waspmote Mote Runner Technical Guide wasp mote," Tech. Guid., p. 85, 2015.

Con estos comandos nos aseguramos que el servidor MRSH se pueda ejecutar desde cualquier directorio en el terminal de Ubuntu. Para no tener que ejecutar estos comandos cada vez que vayamos a utilizar MRSH es muy útil usar un script y en el presente trabajo se utiliza ExportMoterunner.sh. El contenido del script se observa en la figura 3.8

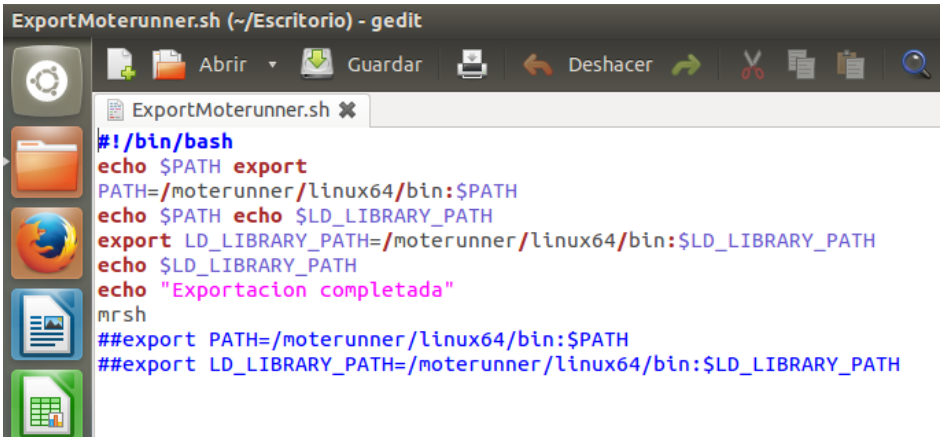


Figura 3.8 Script de exportación de variables de entorno

.Cuando el script se ejecuta en la consola nos muestra un mensaje de “Exportación completada” que indica que el proceso se ejecutó con éxito.

Para comprobar que MRSH se está ejecutando, abrimos el navegador Mozilla y en la dirección escribimos *http://localhost:5000*, luego de lo cual deberíamos observar la pantalla de bienvenida al Mote Runner Launchpad, tal como se muestra en la figura 3.9.

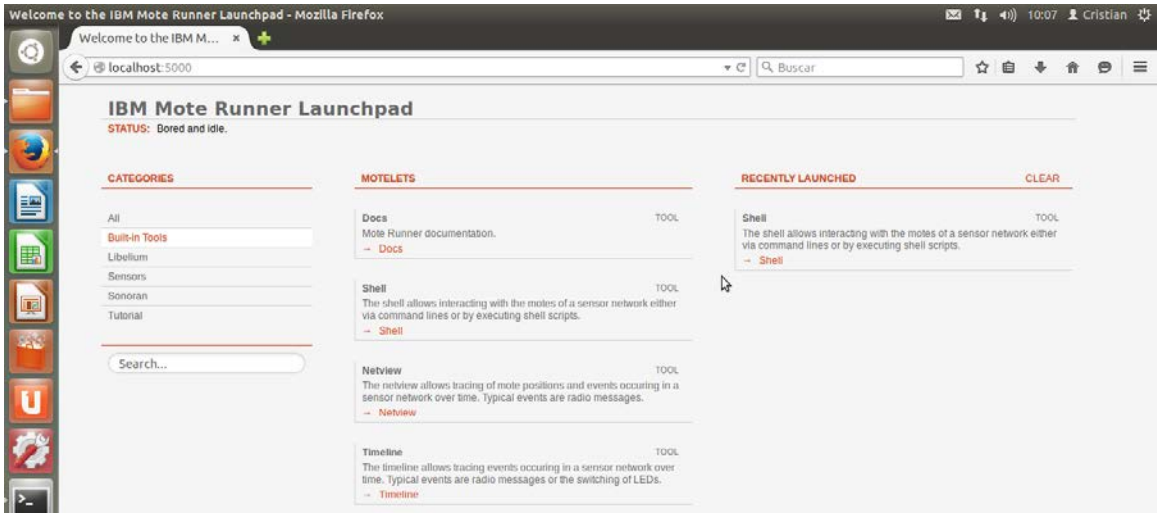


Figura 3.9 Pantalla de bienvenida a Mote Runner Launchpad

Dentro de esta página lo que nos interesa por el momento es el Shell, que es precisamente el que nos permite instalar, ejecutar, depurar y eliminar aplicaciones en los nodos, sean estos físicos o simulados.

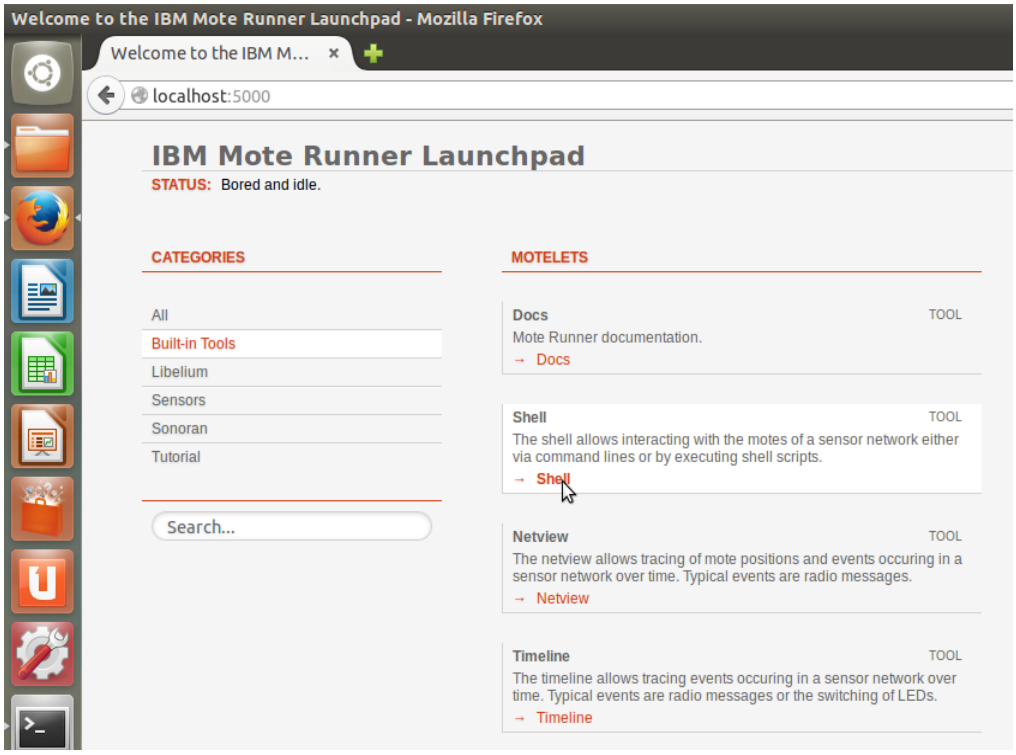


Figura 3.10 Ejecutando el Shell desde la página del Launchpad

En la figura 3.10 observamos la ubicación del link que ejecuta el Shell y en la figura 3.11 el Shell propiamente dicho. Esta es la línea de comandos que nos permite actuar directamente sobre los motes.

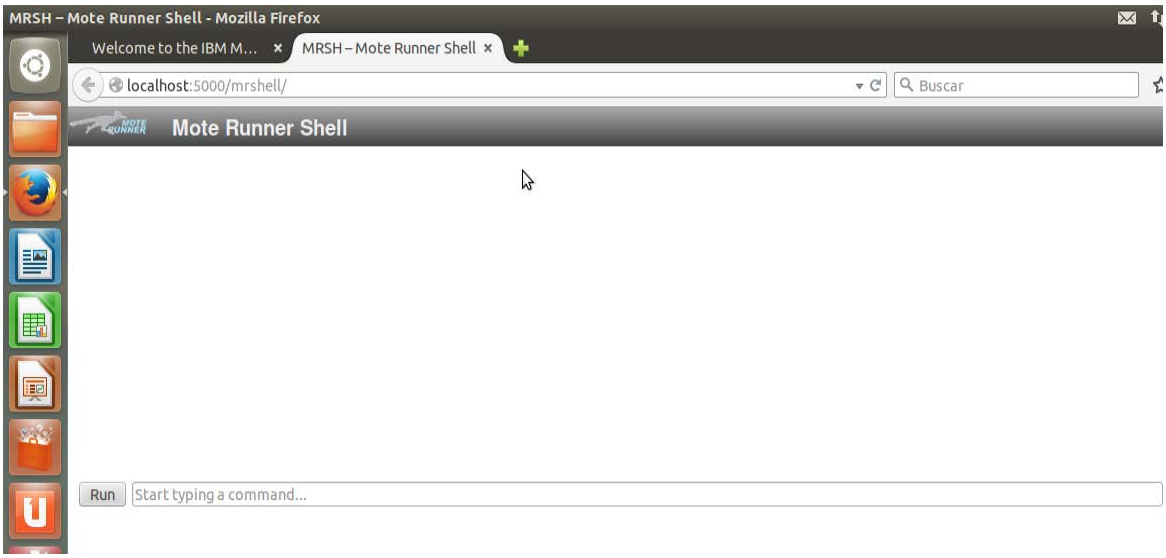


Figura 3.11 Shell de Waspmote Pro v1.2

3.3.3 Librería MRv6 edge para el Gateway

Al mote que se usará como Gateway se le debe conectar el módulo Ethernet que proveerá de la conexión con cable de red al computador que ejecuta el Mote Runner y también se le debe instalar la librería mrv6-edge que es la que lo acredita como Gateway dentro de la red 6LoWPAN.

El proceso de instalación de esta librería depende de la conexión previa del mote usando el cable USB provisto para este efecto. Una vez conectado el mote, se utiliza el Shell para conocer que puerto está usando el sistema para comunicarse con el mote, esta operación se realiza con el comando que observamos en la figura 3.12. Y luego creamos una conexión del servidor con el mote identificado en el puerto

que nos muestra el comando.

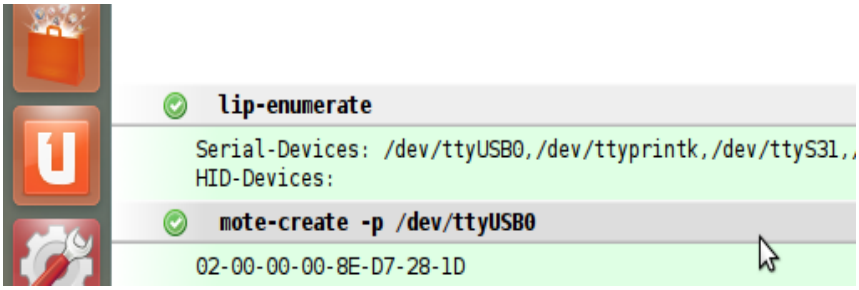


Figura 3.12 Conexión al nodo Gateway conectado vía USB al PC

Luego podemos utilizar el comando MOMA[23] (*Mote Management*) moma-list para ver los programas que tiene instalados el mote como se observa en la figura 3.13.

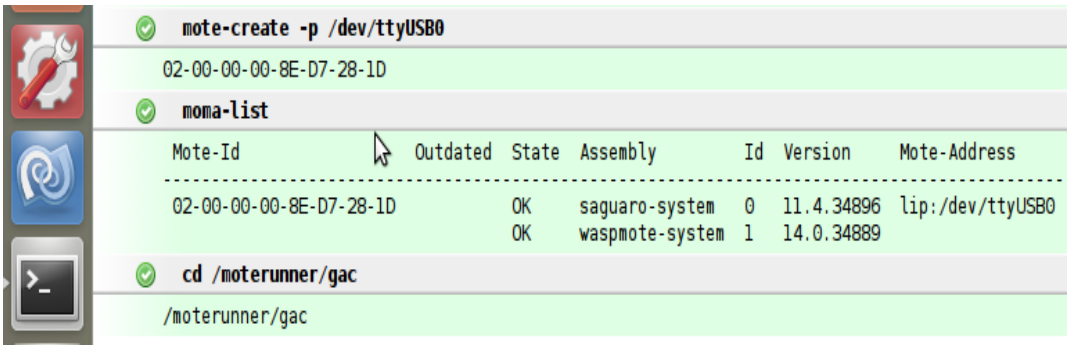


Figura 3.13 Comando moma-list

[23] MOMA: el protocolo MOMA Mote Manager especifica una serie de mensajes de LIP para configurar un sistema Mote Runner. Cualquier aplicación de protocolos de red inalámbrica o por cable pueden enviar mensajes LIP túnel en el sistema para iniciar estas funciones de gestión MOMA

Para instalar la librería mrv6-edge se utiliza el comando moma-load que se encuentra en el directorio /moterunner/gac. La ejecución se observa en la figura 3.14.

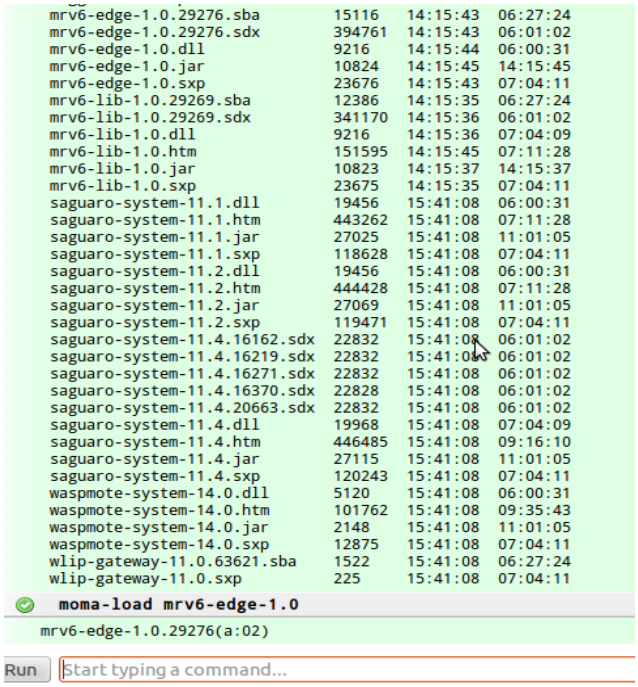


Figura 3.14 Instalación de la librería mrv6-edge

Posterior a la instalación de la librería mrv6-edge en el nodo Gateway, configuramos la dirección IPv4 del mismo, como se observa en la figura 3.15.

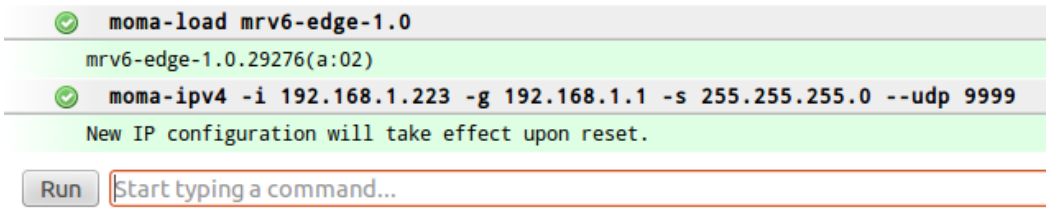


Figura 3.15 Configuración de dirección IPv4 en el Gateway

La dirección del Gateway queda de esta forma configurada a 192.168.1.223 con la máscara de red 255.255.255.0. En el PC en cambio la dirección es 192.168.1.1 y el puerto UDP de comunicación con el Gateway es el 9999. Los cambios de configuración al Gateway se aplican luego del reinicio del mismo.

3.3.4 Librería MRv6 para los nodos inalámbricos

El procedimiento para instalar la librería mrv6-lib en los nodos inalámbricos es el mismo que se utilizó para instalar mrv6-edge en el Gateway. Es decir, conectamos los motes con cable USB uno por uno y usamos el comando lip-enumerate para saber en qué puerto USB fue reconocido el mote y posteriormente el comando mote-create con el puerto que nos indicó el comando anterior como se observa en la figura 3.16.

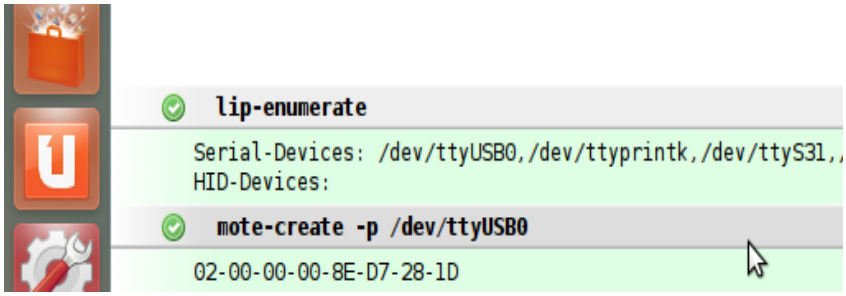


Figura 3.16 Conexión a un nodo inalámbrico

El siguiente paso es instalar la librería mrv6-lib, desde el mismo directorio donde se encuentra mrv6-edge, con el comando moma-load como se muestra en la figura 3.17.

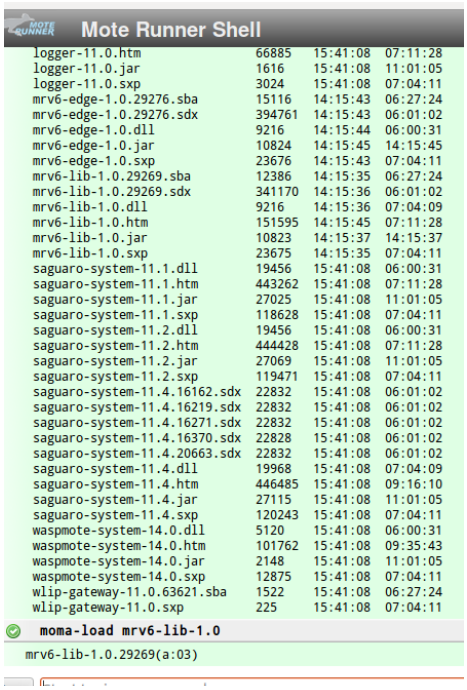


Figura 3.17 Instalación de la librería mrv6-lib en nodo inalámbrico

3.3.5 Configuración del Gateway y de la red 6LoWPAN

Una vez instaladas las librerías pertinentes en los nodos de la red, procedemos a configurar el Gateway para que la red 6LoWPAN pueda intercambiar datos entre sus elementos.

Abrimos una ventana con el terminal de Ubuntu y ejecutamos el script de exportación de variables de entorno, de forma que se ejecute el servidor web MRSH. En la figura 3.18 se observa la ventana del terminal con el resultado de ejecutar el script.

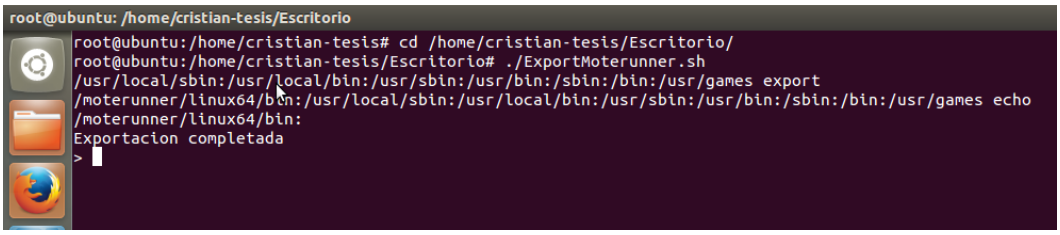


Figura 3.18 Ejecución del script de exportación de variables de entorno

En otra ventana del terminal de Ubuntu se ejecuta el túnel que viene incluido dentro de las herramientas de Mote Runner. El archivo correspondiente al túnel se encuentra en el directorio /moterunner/examples/mrv6/tunnel. En la figura 3.19 se observa el resultado de la ejecución del túnel.

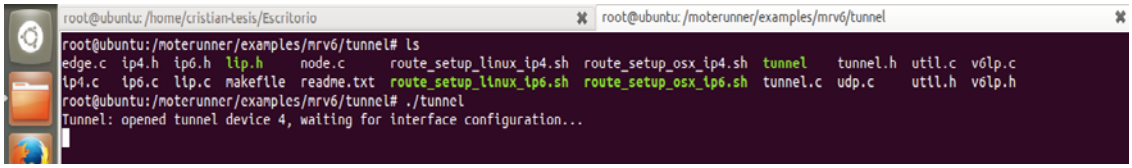


Figura 3.19 Ejecución del túnel

El siguiente paso es conectar el Gateway y la PC con un patchcord UTP cruzado. La PC debe tener configurada la dirección IPv4 que se puntualizó en el Gateway en el apartado 3.3.3 con el comando moma-ipv4 (es decir la dirección 192.168.1.1). Para establecer la conexión del PC con el Gateway por medio del cable UTP, se ejecuta el comando mote-create en el shell de acuerdo a lo que se observa en la figura 3.20.

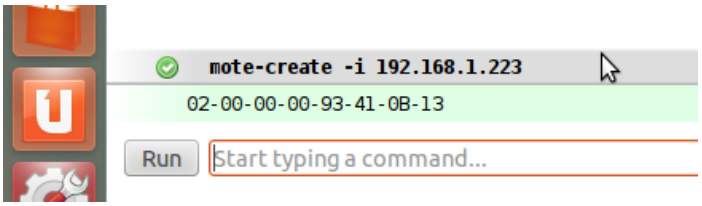


Figura 3.20 Conexión al Gateway por medio de cable UTP

Una vez establecido el enlace con el Gateway se carga el archivo mrv6.js que se encuentra en el directorio /moterunner/examples/mrv6/lib/js usando el comando source, tal como se observa en la figura 3.21.

```
mote-create -i 192.168.1.223
02-00-00-00-93-41-0B-13
cd /moterunner/examples/mrv6/lib/js
/moterunner/examples/mrv6/lib/js
source mrv6.js
null
cd /moterunner/examples/mrv6/src/tmp
/moterunner/examples/mrv6/src/tmp
u0 v6-setup --MAX_DEPTH=12 --NUM_CHILDREN=5 --MAX_CHILDREN=5 --MAX_MOTES=6 --RECV_SAFETY_MILLIS=3 --R24_SLOT_RCV_MILLIS=12 --R24_SLOT_GAP_MILLIS=15 --R24_BEACON_GAP_MILLIS
MRv6
Edge mote configuration:
 Variable                    Value
 ---------------------------------
 R24_SLOT_RCV_MILLIS         12
 R24_SLOT_GAP_MILLIS         15
 R24_BEACON_GAP_MILLIS       20
 R868_SLOT_RCV_MILLIS        100
 R868_SLOT_GAP_MILLIS        40
 R868_BEACON_GAP_MILLIS      40
 RADIO_SELECT                0
 BEACON_INTERVAL_MILLIS      0
 PANID                       12816
 CHANNEL                     1
 LIFESIGN_INTERVAL_CNT       10
 SYNC_INTERVAL_CNT           0
 INFO_INTERVAL_CNT           0
 BEACON_SCAN_RANGE_SECS      120
 BEACON_SCAN_RESTART_SECS    60
 EDGE_BUFFERS_SIZE           768
 EDGE_BUFFERS_CNT            12
```

Figura 3.21 Carga del java script para la configuración de la red 6LoWPAN

Luego de ejecutar el Java Script se usa el comando u0 v6-setup con los parámetros recomendados para la red, tales como el número de canal, máximo número de motas, el tiempo en milisegundos en el que se

envía un mensaje de baliza, el tiempo de slot asignado a cada mota, entre otros parámetros importantes para la configuración del prototipo de red de sensores inalámbricos 6LowPAN.

Luego de ejecutar el comando de configuración del Gateway, la ventana del terminal de Ubuntu donde ejecutamos el túnel mostrará que el nodo del Gateway ha sido añadido al mismo tal como se muestra en la figura 3.22.

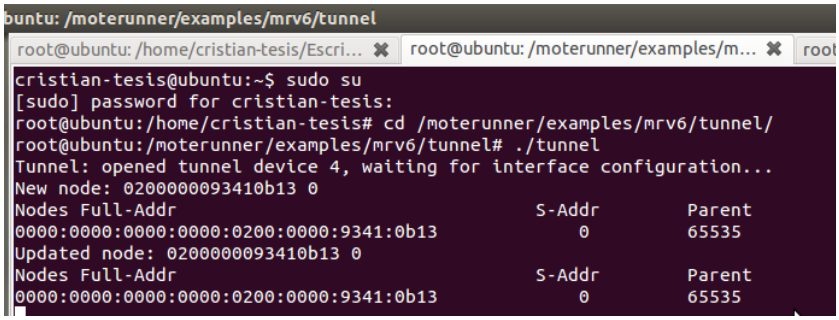


Figura 3.22 El nodo Gateway ha sido añadido al túnel

Por el momento el Gateway tiene únicamente la dirección MAC de 64 bits que lo identifica y no tiene configurada una dirección IPv6.

Para añadir la dirección IPv6 al Gateway debemos ejecutar un script provisto por Mote Runner (route_setup_linux_ip6.sh), pero antes de ejecutarlo, lo abrimos para configurar en el mismo el direccionamiento requerido, tal como se muestra en la figura 3.23.

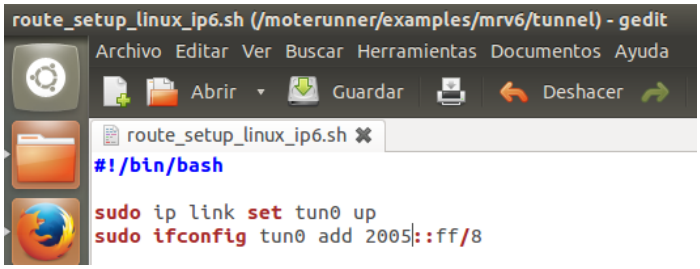


Figura 3.23 Script de configuración del túnel y direccionamiento IPv6

En otra ventana del terminal de Ubuntu se ejecuta el script de configuración del direccionamiento IPv6 para la red 6LoWPAN. Este script está incluido también en las herramientas de Mote Runner y se encuentra en el mismo directorio del túnel. La ejecución del script se observa en la figura 3.24.

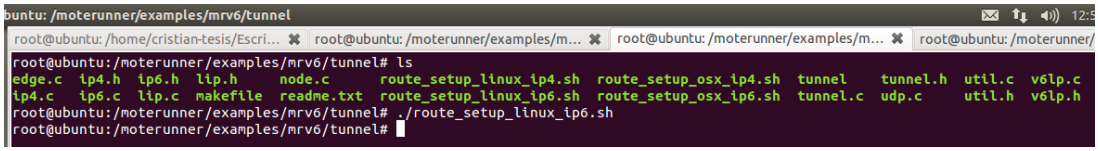


Figura 3.24 Ejecución del script de direccionamiento IPv6 del túnel

Cuando se ejecuta el script tal como se observa en la figura 3.24, la ventana del terminal donde se ejecutó el túnel propiamente dicho actualiza la información del Gateway. Esto se observa en la figura 3.25

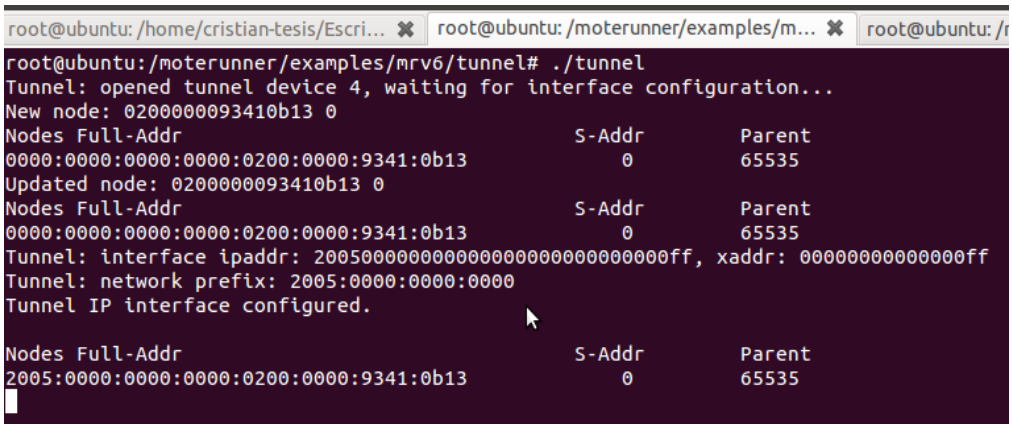


Figura 3.25 Nodo Gateway con la dirección IPv6

Para obtener la dirección IPv6 de 128 bits se toman los 64 bits de la dirección MAC y se completan con 64 del script de configuración. Usualmente en EUI-64[24] a la dirección MAC de 48 bits se le agrega FE:FF hex para completar los 64 bits luego de complementar el séptimo bit.

En la figura 3.25 podemos observar que el nodo Gateway tiene, luego de ejecutar el script de configuración de direccionamiento, una dirección IPv6 de 128 bits.

Una vez configurado completamente el Gateway, se procede a encender los nodos inalámbricos que conforman la red 6LoWPAN del prototipo. En la figura 3.26 se observa cómo se añadieron automáticamente a la red, a través del MRv6 que se encuentra ejecutándose en todos los motes.

[24] EUI-64: El identificador de interfaz de 64 bits se deriva comúnmente de los 48 bits de la dirección MAC. Una dirección MAC se convierte en una dirección EUI-64 de 64 bits insertando FF:FE en el medio.

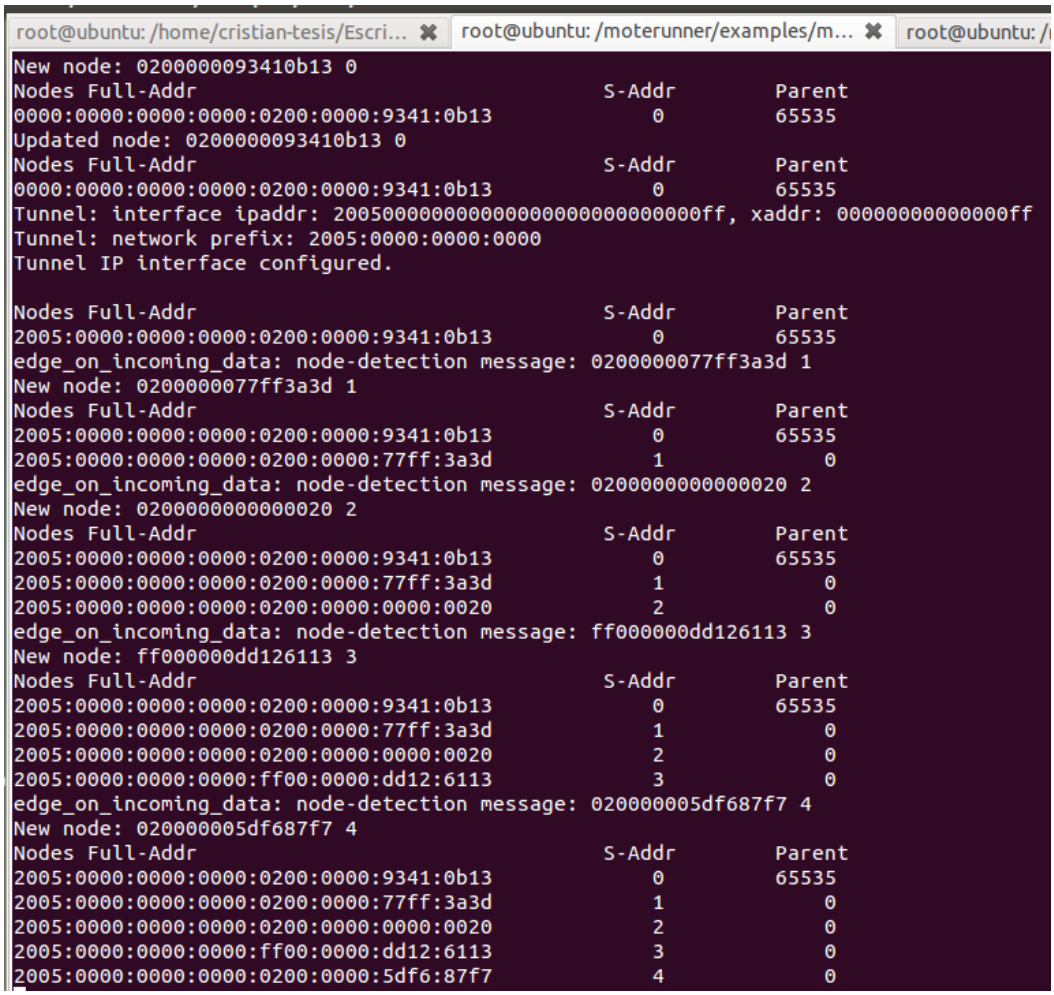


Figura 3.26 Todos los nodos de la red 6LoWPAN con sus direcciones IPv6

En la figura 3.25 se observa también que el nodo Gateway es el padre y que cada nodo se conectó a él. El Gateway por su parte está conectado usando un cable UTP al switch y por medio de este al PC que ejecuta el túnel. Al mismo switch se conecta el Servidor de monitoreo.

Usando el shell del servidor MRSH podemos comprobar la configuración de la red 6LoWPAN por medio de la ejecución del comando v6-connect, tal como se observa en la figura 3.27.

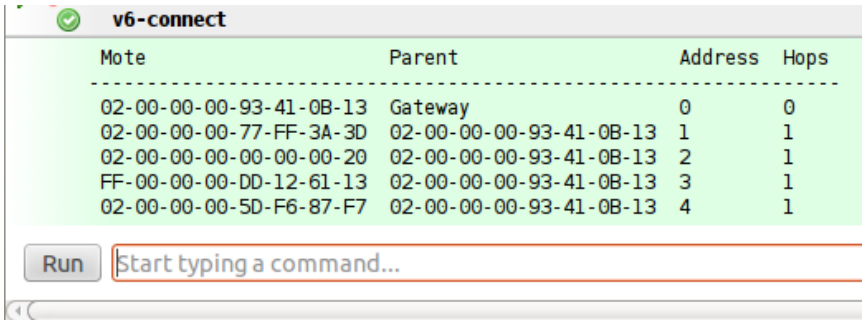


Figura 3.27 Nodos conectados en la red 6LoWPAN shell de MRSH

Una vez que se ha verificado que los nodos inalámbricos se añadieron a la red se procede a comprobar conectividad utilizando el comando ping para tal efecto. En una ventana del terminal de Ubuntu se ejecutan los comandos respectivos con las direcciones de cada uno de los nodos inalámbricos y los resultados se observan en las figuras 3.28, 3.29, 3.30 y 3.31.

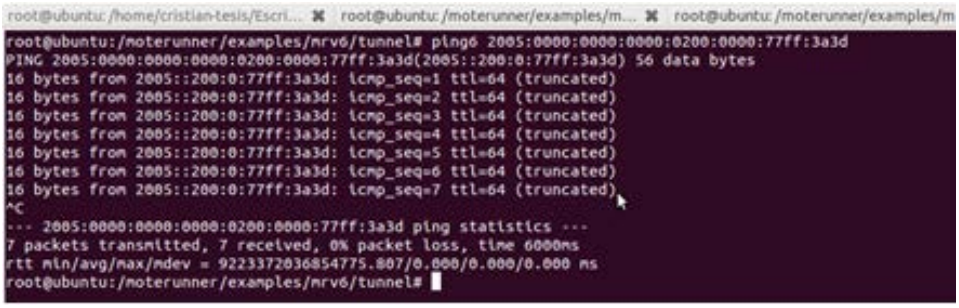


Figura 3.28 Resultado del ping desde la PC al nodo inalámbrico 1

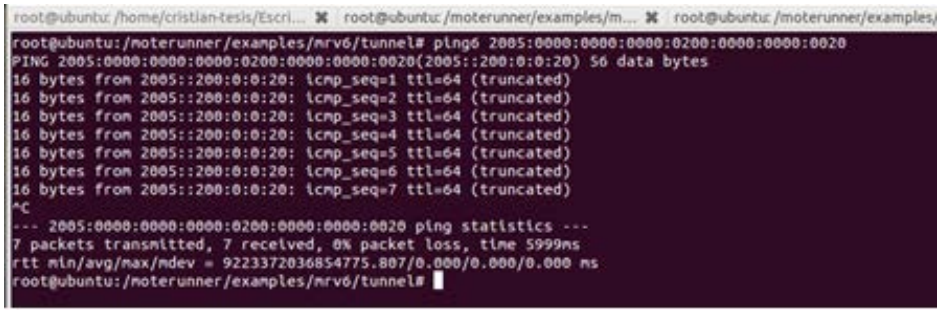


Figura 3.29 Resultado del ping desde la PC al nodo inalámbrico 2

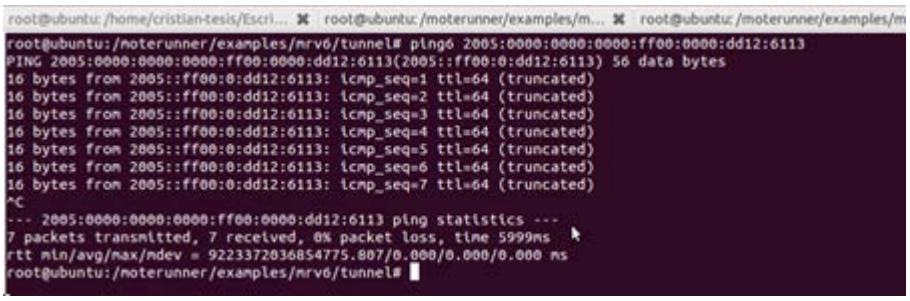


Figura 3.30 Resultado del ping desde la PC al nodo inalámbrico 3

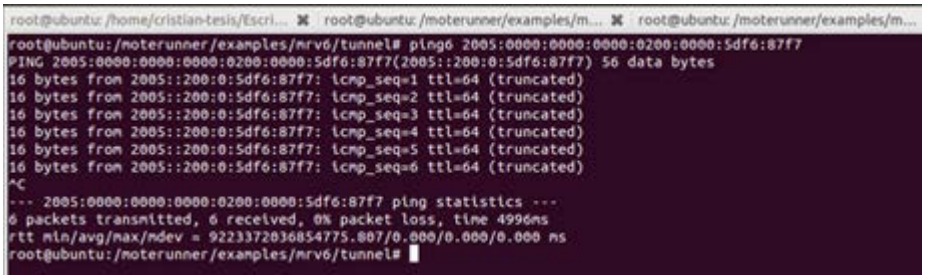


Figura 3.31 Resultado del ping desde la PC al nodo inalámbrico 4

En las figuras con las respuestas al ping se puede observar que a continuación de cada respuesta se incluye la palabra "*truncated*", es

decir truncado. Esto se debe a que los paquetes ICMP[25] sufren el proceso de compresión de cabecera implementado por el protocolo 6LoWPAN.

Una vez realizadas las pruebas anteriores, tenemos implementada la topología de red de acuerdo a los requerimientos planteados en el presente trabajo y lista para realizar las pruebas necesarias a fin de comprobar el funcionamiento del prototipo de sistema de gestión de luminarias LED.

3.3.6 Limitaciones de programación en los nodos

3.3.6.1 Programación en C# y Java para Mote Runner

Las aplicaciones para Mote Runner se pueden escribir usando los lenguajes de programación C# y Java, tomando en cuenta que Mote Runner está diseñado para pequeños sistemas embebidos en los motes y los lenguajes C# y Java son de uso general. Esta diferencia hace que, básicamente se presenten limitaciones de uso de los lenguajes al escribir las aplicaciones para Mote Runner.

El compilador de Mote Runner transforma el *bytecode* del lenguaje general a uno específico conocido como *Common Intermediate Language* (CIL). Si el programa escrito contiene elementos que no son

[25] ICMP: El Protocolo de Mensajes de Control de Internet o ICMP por sus siglas en inglés de Internet Control Message Protocol es el sub protocolo de control y notificación de errores del Protocolo de Internet IP.

admitidos por Mote Runner, se presentarán los mensajes de alerta correspondientes y no se podrán ejecutar.

3.3.6.2 Características no soportadas en Mote Runner

Debido que los motes tienen capacidades limitadas de procesamiento y por tanto Mote Runner es un sistema limitado, es lógico que no sea capaz de soportar todas las características del lenguaje con el que se escriban las aplicaciones. De hecho muchas de ellas no tendrían sentido al utilizarse en una plataforma específica como lo es Mote Runner, lo mismo que sucede, por ejemplo cuando escribimos aplicaciones para teléfonos celulares.

Las restricciones más relevantes a la hora de programar para Mote Runner son las siguientes [16]:

- Tipos de datos double y float.
- Aritmética de enteros de 64 bits o más.
- Matrices multidimensionales.
- Reflexión.
- Las clases internas en C #, sólo se admiten las clases internas en Java.
- Templates.
- Delegados de multidifusión o multicast, solo es compatible el delegado unicast.

- Hilos y sincronización de primitivas.
- Tipo de datos bool o matrices booleanas.
- Tipo de datos String y Arreglos de String (*String Array*).
- Empaquetamiento.
- API's de tiempo de ejecución estándar.
- Enumeraciones.

Las API's de tiempo de ejecución estándar son muy grandes para que puedan funcionar en sistemas embebidos, es por esto que Mote Runner define sus propias API[26] del sistema y cualquier soporte *Language Runtime* ha sido reducido a su mínima expresión. Por lo tanto no existe reflexión, no existen clases para los tipos de valores de empaquetamientos, la clase Object no incluye métodos ni campos. Los arreglos multidimensionales no son compatibles directamente, aunque pueden ser implementadas a través de arreglos de objetos de anidación.

3.3.6.3 Tipos de datos en Mote Runner

Mote Runner implementa una máquina virtual de 16 bits, por lo que la ranura para el stack y los tamaños de objetos, matrices y demás están restringidos precisamente a estos 16 bits. Como excepción la VM

[26] (*Application Programming Interface*), es el conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

(*Virtual Machine*) es compatible con enteros de 32 bits, pero su uso implica una disminución en el rendimiento del mote que puede ser tan crítico como aumentar la lentitud hasta 5 veces que cuando no se usa [16].

El tipo de dato estándar en Java y C# es el int, es decir el entero y está dimensionado como de 32 bits. Es por esta razón que se debe utilizar alguna opción que supla esta deficiencia. Las posibles opciones son:

1. Utilizar enteros cortos en lugar de los estándares.
2. Utilizar un int de 16 bits.

Al utilizar la primera opción, el programador debe utilizar conversiones de tipos de datos puesto que el compilador suele convertir los enteros cortos automáticamente a enteros int. Si por alguna razón alguna de las conversiones es omitida por el programador, la VM no tendrá problema en trabajar con el int pero la operación será ciertamente lenta. El problema principal de esto es que no se puede saber a simple vista con qué tipo de entero está trabajando el mote, puesto que para discriminar el tipo de dato se debería leer el archivo CIL.

La segunda opción es lo que hacen los compiladores de C para sistemas embebidos, pues contienen datos tipo entero de 16 bits. El problema es que los compiladores de C# y Java no tienen esta opción en forma directa sino que el cambio se realiza en la fase de conversión del código mediante el uso del compilador Mote Runner.

3.3.6.4 Excepciones en Mote Runner

Para el manejo de excepciones en Java y C#, Mote Runner tiene ciertas restricciones. En C# la clase raíz para las excepciones es *System Exception*. La clase raíz de todas las excepciones para Mote Runner es la *Remote Exception* que es una subclase de *Runtime Exception* de Java y de *System Exception* de C#. Para la captura de las posibles excepciones en la plataforma Mote Runner se utiliza una captura para la clase *Remote Exception*.

De forma que el manejo de excepciones sea aún mejor, se utiliza un objeto singleton que está disponible para las programas de usuario a través del método estático throwIt en todas las clases de excepciones predefinidas.

En cuanto al objeto singleton, se debe mencionar que la referencia únicamente puede ser almacenada en la pila de la máquina virtual y no se puede hacerlo en objetos asignados desde el heap[27]. Su objetivo es mantener el estado de la pila de llamadas y no es posible utilizarlo como puntero de información de contexto para el estado global.

En la figura 3.32 se observa el manejo de excepciones para Java y C#.

[27] Heap: es una estructura de datos del tipo árbol con información perteneciente a un conjunto ordenado de datos

```
package ex;
import com.ibm.saguaro.system.*;

 public class Ex {
    public int meth (int v) {
        if( v==0 )
            MoteException.throwIt(1);
        return v+1;
    }

    public int meth2 () {
        try {
            return meth(0);
        } catch (MoteException e) {
            if( e.reason==2 )
                throw e;
            return 2;
        }
    }
}
```

```
namespace ex {
    using com.ibm.saguaro.system;

    public class Ex {
        public int meth (int v) {
            if( v==0 )
                MoteException.throwIt(1);
            return v+1;
        }

        public int meth2 () {
            try {
                return meth(0);
            } catch (MoteException e) {
                if( e.reason==2 )
                    throw e;
                return 2;
            }
        }
    }
}
```


Figura 3.32 Excepciones en Java y C# para Mote Runner

3.3.6.5 Clase Initializers

En C# y Java se implementa el concepto de inicializar una clase antes de usarla por primera vez. El código de inicialización para una clase en C# se encuentra en un constructor estático y para Java se encuentra en uno o más bloques estáticos de código. El orden de inicialización depende de la ejecución del programa.

Todo este proceso toma bastante tiempo y por tanto recursos en la VM de los sistemas embebidos. Mote Runner, para hacer más eficiente el proceso convierte la inicialización dinámica en estática, ordenando la ejecución de los inicializadores de acuerdo a que clase es accedida primero. Por ejemplo si el iniciador estático de una clase A accede a la clase B, entonces B es inicializado antes que A.

Aunque esta es la regla general, existen circunstancias especiales donde Mote Runner debe introducir una anotación de peso para eliminar ambigüedades. Las clases con el peso más alto se inician antes que los que tienen menor peso.

Las reglas que sigue Mote Runner para decidir la prioridad de inicialización son las siguientes:

- Clases no relacionadas con el mismo peso se inicializan en cualquier orden.
- Si se detecta un ciclo en el compilador Mote Runner se recoge la clase con el peso más alto del ciclo para inicializar primero.
- Tener más de una clase con el mismo peso en un ciclo es un error.

En la figura 3.33 se observa el uso de una anotación de peso o ponderación para asegurar que la clase B se inicialice antes que las clases A y C.

```
using com.ibm.saguaro.system.attr.Weight;
class A {
  static A() { .. }
}
[Weight(3)]
class B {
  static B() { .. }
}
class C {
  static C() { .. }
}
```


Figura 3.33 La inicialización de la clase B tiene prioridad gracias a la ponderación

3.3.6.6 Inicializadores de datos y objetos inmutables

A la hora de inicializar los objetos Mote Runner coloca los datos de inicialización en la memoria persistente (flash o EEPROM), junto con los códigos y otras estructuras de datos del *assembly*. Luego de inicializar un conjunto, estos datos se copian en los objetos de matriz asignados desde el compilado de objetos en la memoria RAM. Para que se inicialice el assembly se debe haber realizado correctamente la instalación del mismo y el mote debe ser reseteado.

Cuando los objetos de matriz son solo de lectura hablamos de matrices inmutables. Java y C# no tienen una forma de declarar esta propiedad. Mote Runner en cambio define una forma de declarar una matriz inmutable y a través de esta opción se reduce el consumo de memoria RAM. Puesto que la matriz no se escribe en RAM, los datos se leen

directamente desde la memoria persistente.

En la figura 3.34 se observa los inicializadores de datos y objetos inmutables.

C# Implementation

```
[Immutable]
public static readonly int[] persistent = {1,1,1};
```

Java Implementation

```
@Immutable
public static final int[] persistent = {1,1,1};
```

Figura 3.34 Inicializadores de datos y objetos inmutables

3.3.6.7 Delegados

Los delegados contienen la dirección de implementación de los métodos y opcionalmente la referencia a un objeto si se está tratando de un método virtual. Los delegados también definen una firma de método en particular, es decir el número y tipo de parámetros y el tipo de valor de retorno. Los delegados son útiles en programación, están registrados en el sistema operativo y se invocan para eventos específicos, es decir son un mecanismo de devolución de llamada.

El lenguaje C# incorpora directamente el manejo de delegados, mientras que Java no. Aunque Java no los soporte, los delegados Mote Runner pueden ser exportados pero se debe utilizar un patrón de programación específico.

En la figura 3.35 se muestra el uso de los delegados en C# y Java.

C# code:

```
public delegate int Callback (byte[] buf, int len);
```

Java code:

```
public abstract class Callback
  extends com.ibm.saguaro.system.java.Delegate {
    public Callback(Object obj) { super(obj); }
    public abstract int invoke (byte[] buf, int len);
};
```


Figura 3.35 Delegados y llamado de métodos

En el código descrito en la figura 3.34 se puede observar que el delegado se llama con dos argumentos y devuelve un entero. La subclase “resumen”, debe tener un constructor que pasa en su único objeto como parámetro a la superclase. Además se debe tener un método abstracto llamado “*invoke*”, que defina la firma del delegado.

Luego de esto se crea una instancia de un señalador delegado a un método específico a fin de invocar este método por medio del delegado. En la figura 3.35 se observa el código que implementa una clase de aplicación *DelApp* que crea un delegado en el método *registerCallback*. Así también utiliza un delegado en invocación de devolución de llamada y define el método *onCallback*.

C# code:

```
public class DelApp {

    Callback currentCallback;

    public void registerCallback () {
        currentCallback = onCallback;
    }

    public int invokeCallback (byte[] data) {
        return currentCallback(data,data.Length);
    }

    public int onCallback (byte[] buf, int len) {
        // ... do something
        return 0;
    }
}
```

Java code:

```
public class JDelApp {
    Callback currentCallback;

    public void registerCallback () {
        currentCallback =
         new Callback (this) {
           public int invoke (byte[] buf, int len) {
             return ((JDelApp)obj).onCallback(buf,len);
           }
         };
    }

    public int invokeCallback (byte[] data) {
        return currentCallback.invoke(data,data.length);
    }

    public int onCallback (byte[] buf, int len) {
        // ... do something
        return 0;
    }
}
```


Figura 3.36 Métodos registerCallback y onCallback

3.4 Código implementado en los nodos

La aplicación desarrollada para los nodos consta de dos clases, Sense y SenseSocket. SenseSocket se encarga de crear el socket en el puerto “LOCAL_PORT”, el 1023 y que tiene como destino el 1024. Dentro del código se aprecia la creación del paquete de datos para ser enviado.

Bienvenido × Sense.cs ×

Sin selección

```
179         }
180     }
181
182     public class SenseSocket : UDPSocket {
183         /// <summary>Constructor </summary>
184         public SenseSocket() {
185             this.bind(Sense.LOCAL_PORT);
186         }
187
188         /// <summary>Receive packet </summary>
189         public override int onPacket(Packet packet) {
190             // Sends back incmoing data
191             uint len = packet.payloadLen;
192             byte[] buf = (byte[]) Util.alloca((byte) len, Util.BYTE_ARRAY);
193             Util.copyData(packet.payloadBuf, packet.payloadOff, buf, 0, len);
194             try {
195                 packet.swap(len);
196             } catch {
197                 return 0;
198             }
199             Util.copyData(buf, 0, packet.payloadBuf, packet.payloadOff, len);
200             this.send(packet);
201             return 0;
202         }
203     }
204 }
```

Figura 3.37 Socket que permite el envío de información recolectada

Dentro de la clase Sense se incluye la porción de código que se encarga del envío de los datos recolectados por los sensores.

En la figura 3.38 se observa el código que permite el envío de datos. Concretamente se analiza una de las entradas digitales de cada nodo, la misma que representa el estado de la luminaria LED, es decir si está encendida el dato será un 1 lógico y si está apagada el dato será un 0 lógico.

OnAdcData crea el paquete para ser enviado luego. Primero se crea un arreglo para la dirección 2000::11, si el valor leído actualmente desde la entrada digital es distinto al anterior, y luego adjunta ese valor al

paquete para luego enviarlo a través del socket que se encuentra enlazado al puerto 1023.


```
static int onAdcData (uint flags, byte[] data, uint len, uint info, long time) {
    if ((flags & Device.FLAG_FAILED) != 0){
    // Si al leer el sensor existe alguna falla el dato no sirve.
    return -1;
    }

        //guardo en result el valor leido de la entrada digital 1
        int result = gpio.doPin(GPIO.CTRL_READ, WASPMOTE.PIN_DIGITAL1);
        onLeds(1);
        onLeds(2);
        //si el valor leído es diferente al valor previo debo crear  un arreglo de 16 bytes pa
        //dirección 2000::11
        if (result != valorPrevio)
        {
            valorPrevio = result;
            Packet packet = Mac.getPacket();
            byte [] aux1 = new byte[16];
            aux1[0] = (byte) 0x20;
            aux1[1] = (byte) 0x11;
            //desde el elemento 2 del arreglo hasta el 15 lo lleno con ceros
            for(int i=2;i<15;i++)
            {
                aux1[i]=(byte) 0x00;
            }
            //el último byte es el 11 y así tenemos la dirección 2000::11
            aux1[15] = (byte) 0x11;
            //creo el paquete que se enviará con las direcciones origen de MAC, el modo full p
            //una dirección de 128 bits
            Address.setAddress(packet.dstaddr,0,Address.ADDRESS_MODE_FULL,aux1,0);
            //el puerto fuente es el asignado en LOCAL_PORT
            packet.srcport = LOCAL_PORT;
            //el puerto destino es el 1023
            packet.dstport = 1023;
            //creo el arreglo de 4 bytes para incluir el dato leido desde la entrada digital
            byte [] aux = new byte[4];
            aux[0]= (byte)result;
            //aux[1]= valorPrevio;
            //captura de excepciones
            try {
                //creo el paquete con el puerto destino y el origen y el tamaño del arreglo aux
                packet.create(1023, LOCAL_PORT, (uint)aux.Length);
            } catch {
                // en error se rompe el mensaje
                return -1;
            }
            //el id del paquete se obtiene del método getId
            packetId = packet.getId();
            //creo el payload con el valor de payload del paquete a enviar
            byte[] pybuf = packet.payloadBuf;
            //defino la variable entera pyoff con el payload off
            uint pyoff = packet.payloadOff;
            //copio los datos con el arreglo aux, desde el buffer 0 hasta el tamaño del payload
            Util.copyData(aux, 0, pybuf, pyoff, (uint)aux.Length);
            //envio el paquete creado
            socket.send(packet);
        }
        else
            timer.start();
    return 0;
}
    static void onLeds(uint dato)
    {
        byte tipo = (byte) dato;
        LED.setState(tipo, (byte)(LED.getState(tipo)^1));
    }
}
```

```
public class SenseSocket : UDPSocket {
    /// <summary>Constructor </summary>
    public SenseSocket() {
        this.bind(Sense.LOCAL_PORT);
    }

    /// <summary>Receive packet </summary>
    public override int onPacket(Packet packet) {
        // Sends back incmoing data
        uint len = packet.payloadLen;
        byte[] buf = (byte[]) Util.alloca((byte) len, Util.BYTE_ARRAY);
        Util.copyData(packet.payloadBuf, packet.payloadOff, buf, 0, len);
        try {
            packet.swap(len);
        } catch {
            return 0;
        }
        Util.copyData(buf, 0, packet.payloadBuf, packet.payloadOff, len);
        this.send(packet);
        return 0;
    }
}
}
```


Figura 3.38 Envío de datos desde el nodo

3.5 Aplicación para el servidor de procesamiento

3.5.1 Base de datos en SQL

Para el almacenamiento de los datos de estado de las luminarias LED recolectados por las entradas digitales de los nodos se utiliza una base de datos implementada en Microsoft SQL y que será alojada en el servidor SQL configurado para tal efecto en la computadora de procesamiento de la información. En la figura 3.39 se observa el código de creación de la base de datos GestorLuminarias y de la tabla Luminaria.

```
drop database GestorLuminarias

create database GestorLuminarias

use GestorLuminarias

Create table [Luminaria] (
    [IPv6] Varchar(50) NOT NULL UNIQUE,
    [NumeroMota] Integer NOT NULL,
    [Ubicacion] Varchar(900) NOT NULL,
    [Mantenimiento] Varchar(30),
    [TiempoDuracion] Integer,
    [NumeroLuminaria] Integer NOT NULL,
Primary Key  ([IPv6])
)
go

select * from Luminaria
```


Figura 3.39 Creación de base de datos y la tabla para almacenar datos de los nodos

La tabla Luminaria guarda la dirección IPv6 del nodo que envía el dato, un número con el que se identifica la mota para el software de interface, la ubicación que es un texto que será ingresado por el operador, un campo para saber si la luminaria está en mantenimiento o no y el tiempo que se encuentra encendida. La clave principal de la tabla es la dirección IPv6 aprovechando de que se trata de un número único.

3.5.2 Código de la solución

En la figura 3.40 se observa el código de la solución implementada en el servidor de procesamiento escrita en C#, la que permite guardar los datos enviados por los nodos en la base de datos de SQL y mostrar la información procesada. La aplicación también nos permite la gestión de

las luminarias puesto que muestra la información del tiempo que cada una permanece encendida y por tanto saber cuándo se debe realizar el mantenimiento preventivo de las mismas, así como también saber el tiempo real de vida útil de las luminarias.


```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Net.Sockets;
using System.Net;
using System.Threading;

namespace GestorLuminarias
{
    public partial class frmGestorLuminarias : Form
    {
        public frmGestorLuminarias()
        {
            InitializeComponent();
        }

        //Cadena de conexión a la BDD y su ruta
        string cadenaConexion = "Data Source=Cristian-PC;Initial Catalog=GestorLuminarias;Integrated Security=True";
        int puerto = 1023;
        string sentenciaSql;
        List<Mota> informacion;
        Thread hiloComunicaciones;
        UdpClient servidor;
        private delegate void CambiarColor(Mota mota);

        private void btnSalir_Click(object sender, EventArgs e)
        {
            this.Close();
            servidor.Close();
        }

        private void btnAñadir_Click(object sender, EventArgs e)
        {
            frmAñadirLuminaria nuevaLuminaria = new frmAñadirLuminaria();
            DialogResult res = nuevaLuminaria.ShowDialog();
            if (res == DialogResult.OK && nuevaLuminaria.sentenciaSql != null)
            {
                sentenciaSql = nuevaLuminaria.sentenciaSql;
                if (EjecutarSentencia(sentenciaSql) != 0)
                {
                    MessageBox.Show("Luminaria ha sido agregada exitosamente", "LUMINARIA AGREGADA EXITOSAMENTE",
                        MessageBoxButtons.OK, MessageBoxIcon.Information);
                    agregarDato(nuevaLuminaria.ipv6);
                }
                else
                {
                    MessageBox.Show("Luminaria no ha sido agregada exitosamente", "LUMINARIA NO AGREGADA",
                        MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
            }

        }

        private void btnAcercaDe_Click(object sender, EventArgs e)
        {
            frmAcercaDe nuevo = new frmAcercaDe();
            nuevo.ShowDialog();

        }
```

```
private int EjecutarSentencia(string sentenciaSql)
{
    // Realizo una nueva conexion SQL mediante la cadena de conexion
    SqlConnection conexion = new SqlConnection(cadenaConexion);
    // Creo un SQL comando y le ingreso la sentencia que deseo que realice y la conexión
    SqlCommand comando = new SqlCommand(sentenciaSql, conexion);
    // Creo un a variable entera resultado y la inicializo en 0
    int resultado = 0;
    // Captura de Excepciones
    try
    {
        // Abro la conexión
        conexion.Open();
        // Asigno a resultado el número de filas afectadas por la consulta SQL
        resultado = comando.ExecuteNonQuery();
    }
    // Captura de la excepción
    catch (SqlException)
    {
        // Cuando no haya ninguna fila afectada el resultado es 0
        resultado = 0;
    }
    // Finalmente esto pasará ocurra o no la excepción
    finally
    {
        // Cierro la conexión
        conexion.Close();
    }
    // Me retorna el resultado
    return resultado;
}
```

```
private void obtenerDatos()
{
    string sentenciaSql = "SELECT * from Luminaria";

    // Hago la conexión a la base de Datos la cual le paso la cadena de consulta
    // y la conexión
    SqlConnection conexion = new SqlConnection(cadenaConexion);
    SqlCommand comando = new SqlCommand(sentenciaSql, conexion);
    // Creo una nueva lista de motas llamada información
    informacion = new List<Mota>();
    // Captura de Excepciones
    try
    {
        // Abro la conexión
        conexion.Open();
        // Creo un SqlReader para leer las filas y lo inicio en null
        SqlDataReader leer = null;
        // Asigno a la variable el leer que lo que la ejecución de la consulta realizada
        leer = comando.ExecuteReader();
        // Si me leyó la conslta enviada
        while (leer.Read())
        {
            // Creo la mota llamada auxiliar que me permitirá ir agregando la información pert
            Mota auxiliar = new Mota();
            auxiliar.Ipv6 = leer["IPv6"].ToString().Trim();
            auxiliar.NumMota = Convert.ToInt16(leer["NumeroMota"].ToString().Trim());
            auxiliar.TiempoDuracion = Convert.ToDecimal(leer["TiempoDuracion"].ToString().Trim
            auxiliar.Ubicacion = leer["Ubicacion"].ToString().Trim();
            auxiliar.NumLuminaria = Convert.ToInt16(leer["NumeroLuminaria"].ToString().Trim())
            if (leer["Mantenimiento"].ToString().Trim() == "Si")
                auxiliar.Mantenimiento = true;
            else
              auxiliar.Mantenimiento = false;
           lstMotas.Items.Add("Mota: " + auxiliar.NumMota);
           cambiarColor(auxiliar);
           informacion.Add(auxiliar);
        }
    }
    // Captura de la excepción del tipo SQL
    catch (SqlException)
    {
        // Defino a cliente como null
        informacion = null;
    }
    // Esto pasará ocurra o no la excepción
    finally
    {
        // Cierro la conexión
        conexion.Close();
    }
```

```
private void agregarDato(string dato)
{
    string sentenciaSql = "SELECT * from Luminaria";
    sentenciaSql += " WHERE IPv6=" + "'" + dato + "'";
    // Hago la conexión a la base de Datos la cual le paso la cadena de consulta
    // y la conexión
    SqlConnection conexion = new SqlConnection(cadenaConexion);
    SqlCommand comando = new SqlCommand(sentenciaSql, conexion);
    // Creo un Cliente con los datos, y le inicializamos como nulo
    informacion = new List<Mota>();
    // Captura de Excepciones
    try
    {
        // Abro la conexión
        conexion.Open();
        // Creo un SqlReader para leer las filas y lo inicio en null
        SqlDataReader leer = null;
        // Asigno a la variable el leer que lo que la ejecución de la consulta realizada
        leer = comando.ExecuteReader();
        // Si me leyó la conslta enviada
        while (leer.Read())
        {
            // Creo un nuevo objeto Mota
            Mota auxiliar = new Mota();
            auxiliar.Ipv6 = leer["IPv6"].ToString().Trim();
            auxiliar.NumMota = Convert.ToInt16(leer["NumeroMota"].ToString().Trim());
            auxiliar.TiempoDuracion = Convert.ToDecimal(leer["TiempoDuracion"].ToString().Trim
            auxiliar.Ubicacion = leer["Ubicacion"].ToString().Trim();
            auxiliar.NumLuminaria = Convert.ToInt16(leer["NumeroLuminaria"].ToString().Trim())
            if (leer["Mantenimiento"].ToString().Trim() == "Si")
                auxiliar.Mantenimiento = true;
            else
              auxiliar.Mantenimiento = false;
           lstMotas.Items.Add("Mota: " + auxiliar.NumMota);
           informacion.Add(auxiliar);
           cambiarColor(auxiliar);
        }
    }
    // Captura de la excepción del tipo SQL
    catch (SqlException)
    {
        // Defino a cliente como null
        informacion = null;
    }
    // Esto pasará ocurra o no la excepción
    finally
    {
        // Cierro la conexión
        conexion.Close();
    }
}

private void frmGestorLuminarias_Load(object sender, EventArgs e)
{
    hiloComunicaciones = new Thread(new ThreadStart(Comunicaciones));
    hiloComunicaciones.Start();
    obtenerDatos();
}

private void btnEliminar_Click(object sender, EventArgs e)
{
    string[] cadena = lstMotas.FocusedItem.Text.Split(':');
    string sentenciaSql = "DELETE from Luminaria";
    sentenciaSql += " WHERE NumeroMota=" + "'" + cadena[1].Trim() + "'";
    if (EjecutarSentencia(sentenciaSql) != 0)
    {
        MessageBox.Show("Luminaria eliminada", "Informacion", MessageBoxButtons.OK, MessageBoxIcon.Inf
        lstMotas.Items.Remove(lstMotas.FocusedItem);
        foreach (Mota valor in informacion)
        {
            if (valor.NumMota == Convert.ToInt16(cadena[1].Trim()))
            {
                informacion.Remove(valor);
                break;
            }
        }
    }
}
```

```
private void lstMotas_ItemActivate(object sender, EventArgs e)
{
    string[] cadena = lstMotas.FocusedItem.Text.Split(':');
    foreach (Mota valor in informacion)
    {
        if (valor.NumMota == Convert.ToInt16(cadena[1].Trim()))
        {
            frmDatosLuminaria mostrarInformacion = new frmDatosLuminaria(valor);
            DialogResult res = mostrarInformacion.ShowDialog();
            if (res == DialogResult.OK)
            {
                if (actualizarMota(valor) != 0)
                {
                    MessageBox.Show("Mota actualizada correctamente", "Informacion", MessageBoxButtons.
                    cambiarColor(valor);
                }
                else
                    MessageBox.Show("Error al momento de actualizar", "Error", MessageBoxButtons.OK, Me

            }

            break;
        }
    }

    lstMotas.FocusedItem.Selected = false;

}


 private int actualizarMota(Mota valor)
 {
     string sentenciaSql = "UPDATE Luminaria SET ";
     // Modifico la dirección IPv6 de la mota
     sentenciaSql += "IPv6='" + valor.Ipv6 + "', ";
     // Modifico el número de luminaria de la mota
     sentenciaSql += "NumeroLuminaria='" + valor.NumLuminaria + "', ";
     if(valor.Mantenimiento == true)
         sentenciaSql += "Mantenimiento='" + "Si" + "'";
     else
         sentenciaSql += "Mantenimiento='" + "No" + "'";
     // Para llevar el proceso de actualización lo ubico a la mota mediante el número de mota
     sentenciaSql += " WHERE NumeroMota='" + valor.NumMota + "'";
     return EjecutarSentencia(sentenciaSql);
 }

 private void cambiarColor(Mota valor)
 {
     if (valor.Mantenimiento == true)
         lstMotas.FindItemWithText("Mota: " + valor.NumMota).BackColor = Color.Orange;
     else
     {
         if(valor.Estado == 1)
             lstMotas.FindItemWithText("Mota: " + valor.NumMota).BackColor = Color.GreenYellow;
         else
             lstMotas.FindItemWithText("Mota: " + valor.NumMota).BackColor = Color.Red;
     }
 }

 // Servidor de comunicaciones
 public void Comunicaciones()
 {
     servidor = new UdpClient(puerto, AddressFamily.InterNetworkV6);
     // Lazo infinito para esuchar por peticiones
     while (true)
     {
         //MessageBox.Show(IPAddress.IPv6Any.ToString());
         // Creación del socket mediante IPEndPoint para UDP
         IPEndPoint equipoRemoto = new IPEndPoint(IPAddress.IPv6Any, puerto);
         //MessageBox.Show(equipoRemoto.Address.ToString());
         try
         {
             Byte[] datosRx = servidor.Receive(ref equipoRemoto);
             // MessageBox.Show(datosRx.Length.ToString());
             foreach (Mota auxiliar in informacion)
             {
                 if (equipoRemoto.Address.Equals(IPAddress.Parse(auxiliar.Ipv6)))
                 {
                     if (auxiliar.Mantenimiento == false)
                     {
                         int datoRx = Convert.ToInt16(datosRx[0]);
                         if (datoRx == 1)
```

```
                        {
                            auxiliar.Estado = datoRx;
                            auxiliar.TiempoFinal = DateTime.Now.Hour + (DateTime.Now.Minute / 60);
                            auxiliar.TiempoDuracion += auxiliar.calcular();
                        }
                        lstMotas.Invoke(new CambiarColor(cambiarColor), new object[] { auxiliar });
                        actualizar(auxiliar);
                    }
                    break;
                }
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }

    }
}

private int actualizar(Mota valor)
{
    string sentenciaSql = "UPDATE Luminaria SET ";
    sentenciaSql += "TiempoDuracion='" + valor.TiempoDuracion + "'";
    // Para llevar el proceso de actualización lo ubico al cliente mediante el clienteId
    sentenciaSql += " WHERE NumeroMota='" + valor.NumMota + "'";
    return EjecutarSentencia(sentenciaSql);
}
}
}
```

Figura3.40 Código de aplicación de procesamiento de información

Al ejecutar el programa, por defecto corre el hilo principal que se encarga de correr la aplicación y los distintos formularios que sirven de interface con el usuario, paralelamente se inicia el hiloComunicaciones y por tanto el método Comunicaciones que implementa el servidor de comunicaciones y se encarga de recibir los datos enviados por los nodos y de procesar la información que recibe. Si el dato enviado implica el cambio de estado de la luminaria, determinado por la lectura de la entrada digital de la mota, se llama al método cambiarColor que muestra el color de acuerdo al estado de la entrada, es decir roja si está apagada, verde si está encendida y naranja si se encuentra en mantenimiento. El estado de mantenimiento sirve para cuando se apagó la luminaria para realizar un mantenimiento preventivo o para cambiarla por una nueva.

Mientras se ejecuta el hilo de comunicaciones en segundo plano, el programa llama al método obtenerDatos que establece la comunicación con la base de datos y lee los registros de la tabla. Mediante este hilo, cada vez que un nodo envía un dato, este se refleja en la ventana de estado de las luminarias.

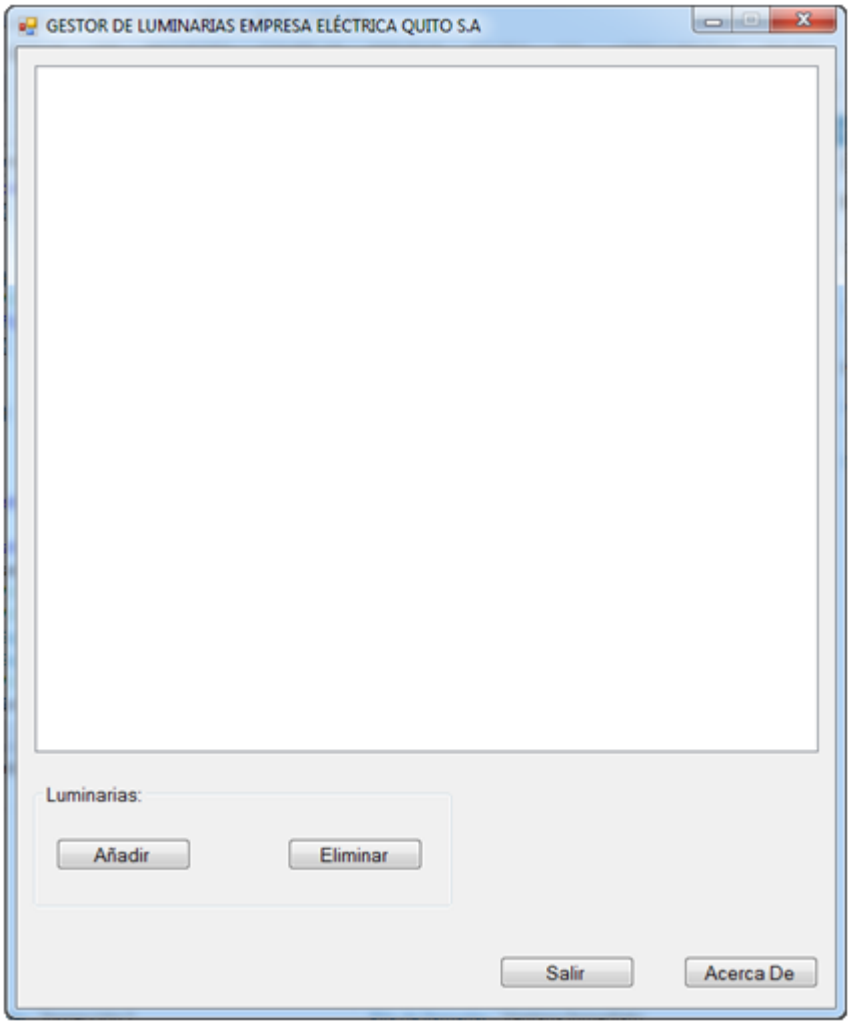


Figura 3.41 Formulario de inicio del Software de Gestión de Luminarias

En la figura 3.41 se observa el formulario de inicio de la aplicación y la ventana principal donde se mostrarán las motas con el estado de la

luminaria conectada a la entrada digital de la misma.

El primer paso consiste en abrir el formulario para añadir una Luminaria, por medio del botón Añadir. La figura 3.42 muestra el formulario.

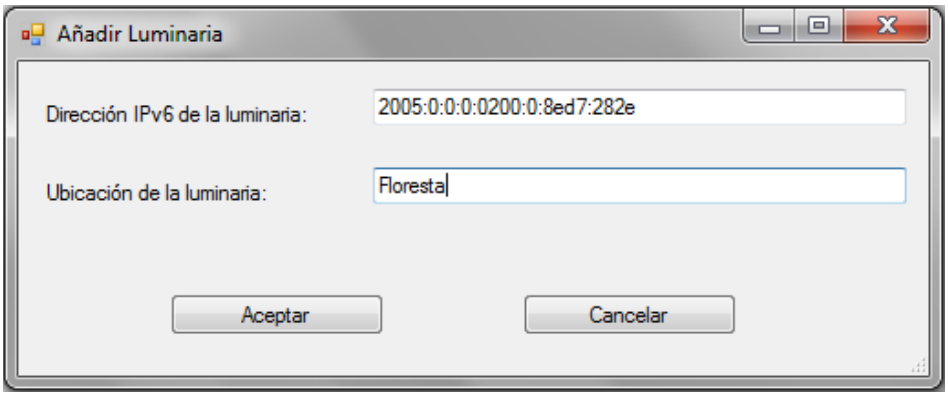


Figura 3.42 Formulario para Añadir Luminaria

En este formulario se ingresa el dato de la dirección IPv6 de la mota a la que está conectada la luminaria y la ubicación de la misma. La dirección IPv6 se obtiene de la ventana de configuración del túnel y debe ser escrita tal como aparece.

También existe la posibilidad de eliminar la mota seleccionándola en la ventana principal y luego presionando el botón Eliminar. Este proceso elimina el registro de la mota especificada en la tabla de la base de datos y por tanto no se muestra en la ventana de la aplicación, pero el nodo sigue siendo parte de la red WSN mientras no se apague.

Una vez ingresados los datos de las motas tenemos una pantalla como la que se muestra en la figura 3.43. El color indica si la luminaria está

encendida, apagada o en mantenimiento.

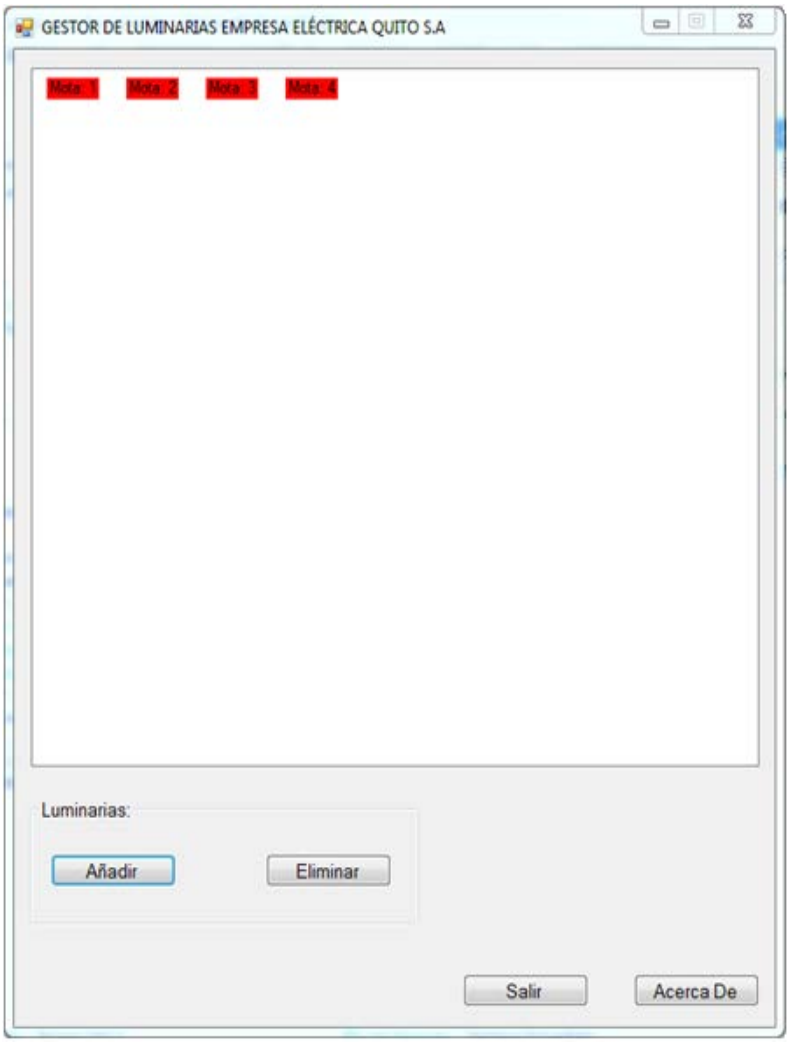


Figura 3.43 Formulario de la aplicación con los datos de las motas

Haciendo doble clic sobre una mota, se presenta una ventana con la información de la misma, tal como se muestra en la figura 3.44.

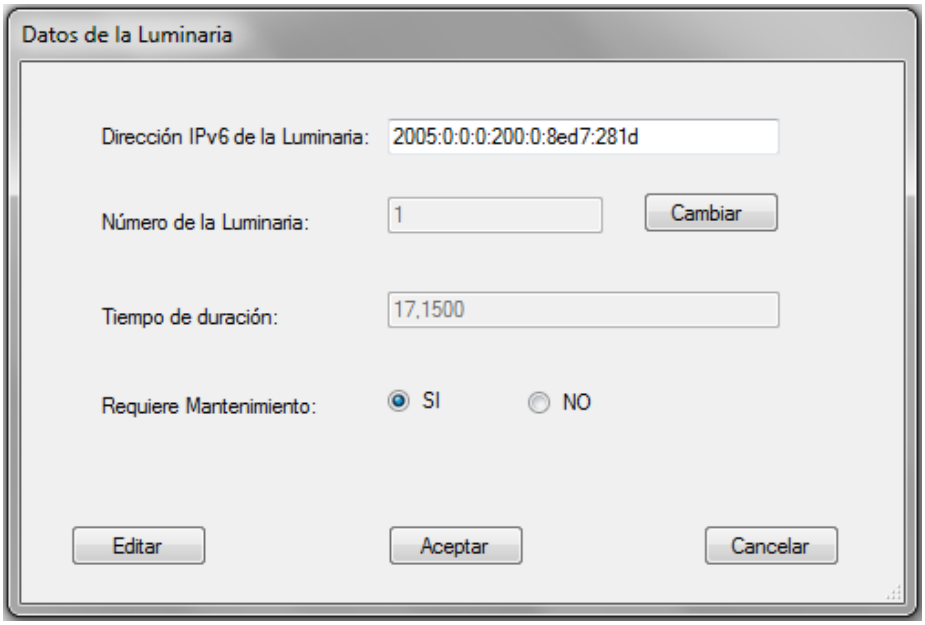


Figura 3.44 Información de una luminaria

Presionando el botón Editar, se tiene la posibilidad de cambiar la dirección IPv6, el número de luminaria y de indicar que requiere mantenimiento. Cualquiera de los parámetros mencionados se pueden modificar o todos a la vez. Al presionar el botón Aceptar, los datos son actualizados en la tabla de la base de datos.

Cuando se indica que la luminaria requiere mantenimiento, el color de la misma en la ventana de monitoreo cambia a color naranja como se observa en la figura 3.45.

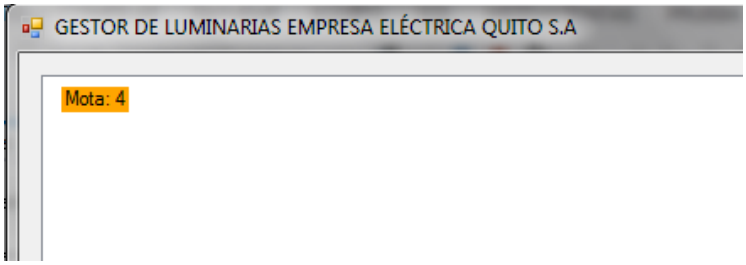


Figura 3.45 Luminaria en mantenimiento

3.6 Pruebas y resultados

Las pruebas que se realizaron con el prototipo comprenden el comprobar que:

- Los nodos se enlazan inalámbricamente al Gateway,
- Los nodos envían la información pertinente cuando se enciende o apaga una luminaria. Para este propósito se coloca un 1 lógico o un 0 lógico en la entrada digital 1 de cada mota.
- La información enviada es recibida por el servidor y por tanto la muestra en la pantalla específica de la aplicación desarrollada.

Para comprobar que los nodos se enlazan inalámbricamente y por tanto forman la red 6LoWPAN se procede como sigue:

Se enciende primero el nodo Gateway, es decir el que tiene conectado el módulo Ethernet. Luego se encienden uno por uno los nodos restantes en los que cargamos la aplicación que enviará el valor de la entrada digital hacia el Gateway.

Una vez que hemos encendido todos los nodos, en la ventana del terminal donde se ejecutó el túnel se observa la dirección de cada uno. En la figura 3.46 se observan los nodos reportándose e indicando su dirección.

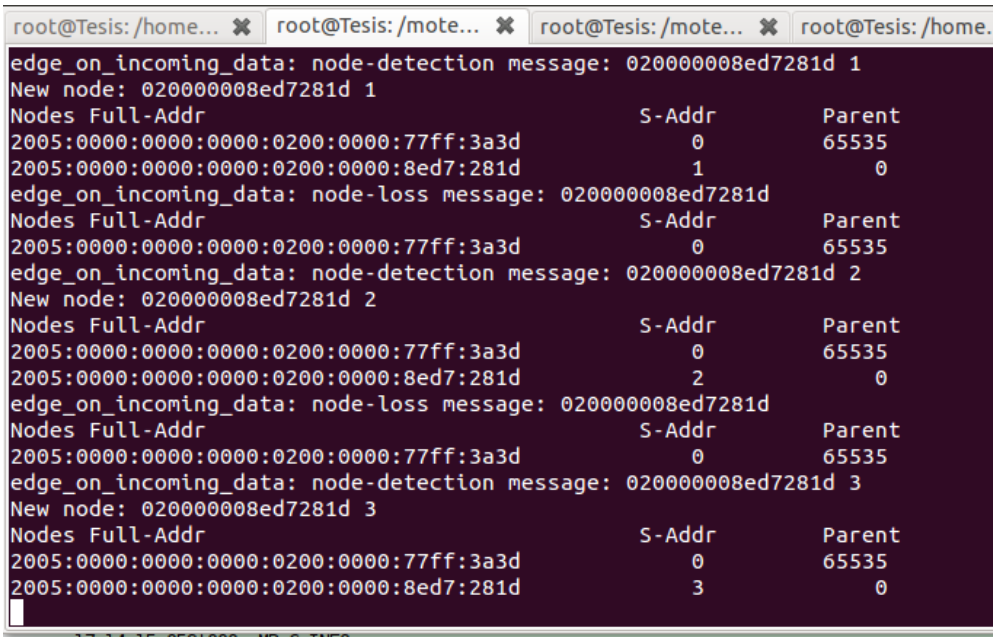


Figura 3.46 Los nodos de la red 6LoWPAN interconectados

En cuanto al envío del estado de la entrada digital, comenzamos con el estado de apagado en las luminarias que se suponen conectadas a cada nodo y por tanto debemos colocar un 1 lógico en la entrada lo que correspondería a la luminaria que ha sido encendida. En la figura 3.47 observamos la alerta que se presenta en el Shell debido al envío del dato de encendido correspondiente a la mota 1 y luego lo propio al enviar el dato correspondiente al apagado.

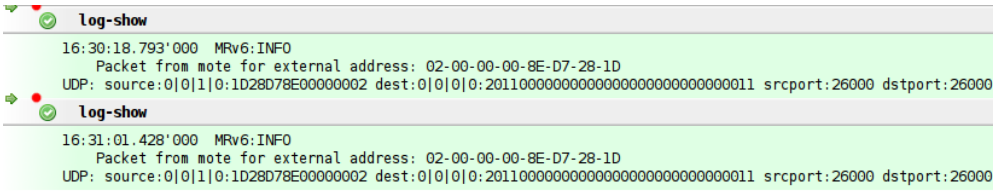


Figura 3.47 Dato de encendido enviado por la mota 1

El payload se modifica a 01000000 cuando se envía 1 y vuelve a ser 00000000 cuando se envía un 0.

Finalmente en la figura 3.48 se observa que el servidor recibe el dato enviado por la mota y actualiza la pantalla de monitoreo con el color correspondiente.

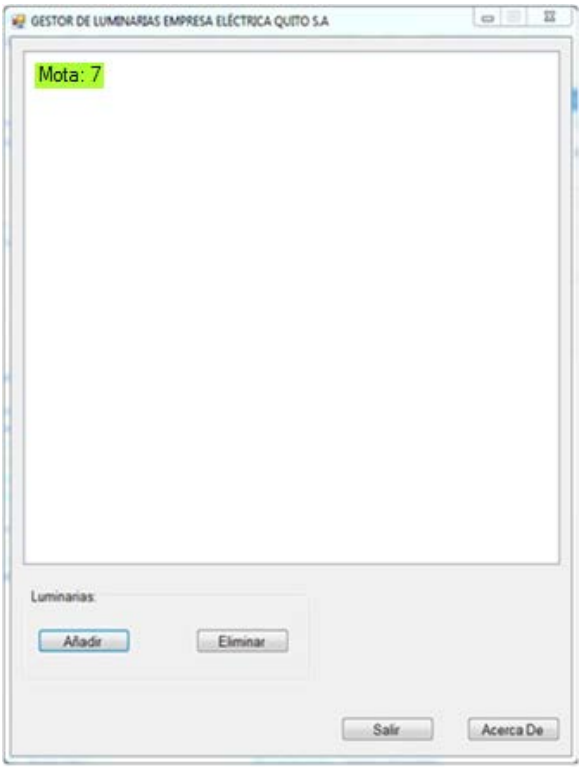


Figura 3.48 Ventana de monitoreo con la luminaria encendida

Con estas pruebas se comprobó que el prototipo cumple con los requisitos que se definieron antes de realizar la implementación. Para poder utilizar el prototipo en ambiente real se necesita tomar en cuenta ciertos requerimientos que se detallan a continuación:

- Para utilizar la entrada digital del mote se debe implementar un circuito que transforme el voltaje de alimentación de la luminaria a los valores que maneja ese tipo de entrada (0 – 5Vdc).
- La fuente de alimentación debe garantizarse por un tiempo prolongado, por lo que se sugiere utilizar el dispositivo de carga solar o tomar energía directamente desde la red pública de alimentación eléctrica.
- Los motes deberán protegerse del medioambiente, es decir deberán estar protegidos del sol, la lluvia, el viento y el polvo. Se sugiere utilizar un tipo de caja de protección para los motes.
- El Gateway debe tener una fuente de alimentación bastante segura debido a que trabaja casi todo el tiempo puesto que no tiene un modo de trabajo "*sleep*".
- Se deberán utilizar las ventanas de estado de la aplicación de procesamiento que se necesiten de acuerdo a la cantidad de luminarias que se vayan a incluir en el sistema.

3.7 Análisis de implementación en la EEQ

En cuanto al análisis del escenario de implementación del sistema en la

Empresa Eléctrica Quito se puede comentar lo siguiente:

Existe un inconveniente con Libelium puesto que la empresa desarrolla sus propias aplicaciones de implementación y el motivo de estudio del presente trabajo estaría contemplado como una aplicación dentro de lo que ellos conocen como Smart Cities. Libelium no vende sus equipos para el desarrollo de aplicaciones comerciales sino que lo hace únicamente para evaluación de los mismos.

En el Anexo I Proforma de Libelium, se observa el costo de un kit de evaluación. De acuerdo a la proforma el kit está en 3526 euros, es decir aproximadamente 3950 USD. El kit incluye 6 motas de las cuales una se utiliza como Gateway. Según esto podríamos decir que, si se pudiese utilizar estos kit para la instalación comercial, por cada 5 luminarias que incluyamos tendríamos ese costo. En cada luminaria se debería incluir un circuito que transforme el voltaje de alimentación de la luminaria LED a los rangos aceptados por la entrada digital del nodo 0 ~ 3.3 V DC.

Aparte de esto se debe incluir la computadora que implementa el túnel para cada Gateway y el servidor de procesamiento que puede ser único para todo el sistema descrito.

En el apartado de comunicación se sugiere usar GPRS en los gateway para enviar los datos hacia el computador túnel el mismo que se conecta a la red mediante cable UTP y por tanto con el servidor de

procesamiento.

Un plan de 300 Mb para transmisión de GPRS está alrededor de $100 mensuales, valor que se tiene que considerar a la hora de implementar un proyecto como el que se presenta en este trabajo.

En resumen, para instalar el sistema en 5 luminarias LED se requiere de una inversión en hardware que bordea los $5700 y en cuanto a la comunicación se necesitarían $100 mensuales.

CAPÍTULO 4

4. CONCLUSIONES Y RECOMENDACIONES

4.1 Conclusiones

- Al finalizar el presente trabajo se puede concluir que el objetivo principal que se planteó en el mismo se ha cumplido, es decir, se implementó un prototipo de sistema de Gestión de Luminarias LED de acuerdo a lo previsto y que cumple con las especificaciones iniciales.

- Al implementar como parte del prototipo, una red 6LoWPAN se demuestra que este tipo de red es perfectamente viable a la hora de solucionar la necesidad planteada en el presente proyecto.

- La pila de protocolos TCP/IP es demasiado compleja para las capacidades de procesamiento limitadas de los nodos de una red WSN, esto implica que la energía que se consuma sea alta y por tanto no es óptimo su utilización en este tipo de redes.

- Si se pretende utilizar IPv6 en las redes WSN sin ninguna modificación, nos enfrentamos a un caso de sobrecarga, puesto que la cabecera sería muy grande para el uso específico que se requiere, es decir, la transmisión de unos pocos bits. Es por esto que se implementa la compresión de cabeceras, lo que hace más eficiente la transmisión en la red.

- Las motas en general tienen limitaciones de procesamiento y en el prototipo implementado lo hemos evidenciado a la hora de escribir la aplicación para los mismos. Tal vez la más importante es la imposibilidad de trabajar con hilos.

- Para enviar el paquete con el dato de la entrada digital 1 que representa a la luminaria, se utiliza un socket lo que nos permite comunicarlo a través de la red y que llegue al Gateway, al túnel y en última instancia al servidor de procesamiento.

- Puesto que la aplicación en las motas está construida en base a eventos, el ciclo de actividad de las motas será la mayor parte de tiempo en estado *sleep*, lo que nos asegura que la duración de la batería sea bastante prolongada. De todas formas al colocarse cada mota junto a una luminaria es factible que se

puedan alimentar de la energía eléctrica pública y evitar el inconveniente de la alimentación por batería.

- Para la comunicación entre las motas y el Gateway nos servimos de MRv6 que es una librería incluida en el kit de desarrollo para las redes WSN con 6LoWPAN de Libelium. MRv6 está desarrollado en C# y se lo debe instalar en los nodos y el Gateway para que se pueda implementar la red 6LoWPAN entre ellos.

- Como parte de la implementación de la red se utiliza el túnel a través de un computador que lo ejecuta. Esto es necesario puesto que el módulo LAN que se conecta en el Gateway únicamente funciona con IPv4. Este túnel además de servir como interface entre los dos protocolos nos permite ver los mensajes de la red 6LoWPAN en el PC y por lo tanto monitorear la misma.

- Es importante modificar el archivo de configuración del túnel puesto que por defecto el mismo viene únicamente con una dirección de Link Local que no asigna direcciones IPv6 a los nodos. Sin esta configuración los nodos no logran tener comunicación fuera de la red 6LoWPAN.

- La aplicación para el servidor de procesamiento incluye la creación de un hilo que implementa el servidor de comunicaciones, es decir, el que se encarga de escuchar el puerto por el cual llegará el paquete con el dato de encendido o apagado de la luminaria. Por otro lado el hilo principal se encarga de los eventos que se presentan en las ventanas de la aplicación.

- Los datos enviados por las motas se almacenan en una tabla de la base de datos Microsoft SQL de forma que se pueden realizar los análisis que se crean convenientes a futuro.

4.2 Recomendaciones

- Se recomienda desarrollar nuevos proyectos en base a lo planteado en el presente trabajo, como por ejemplo incluir la etapa de control de las luminarias, lo que permitiría encender y apagarlas remotamente y de acuerdo a las necesidades.

- Otro campo de desarrollo que se recomienda tomar es el de aplicar el prototipo a otro tipo de aplicaciones con sensores, tales como la medición de energía eléctrica consumida por lo abonados del sistema. En otros ámbitos que no son el de la energía eléctrica se pueden realizar proyectos como el de

administración de un parqueadero, soluciones de vivienda inteligentes y otros más.

- Para desarrollar aplicaciones para las motas se deben tomar muy en cuenta las limitaciones de procesamiento de las mismas y en especial las limitaciones de programación. Esto debido a que, a pesar que se pueden usar lenguajes de programación como C# o Java, no todas las instrucciones y tipos de datos están disponibles para las motas.

- Se recomienda desarrollar una documentación más amigable para el proceso de implementación de soluciones a la hora de utilizar la plataforma de Moterunner pues la que existe es escasa y requiere de bastante análisis para entenderla y utilizarla posteriormente.

REFERENCIAS BIBLIOGRÁFICAS

[1] "I Congreso EI: Iluminación LED en edificios inteligentes - CASADOMO." [Online]. Available: https://www.casadomo.com/comunicaciones/i-congreso-ei-iluminacion-led-en-edificios-inteligentes. [Accessed: 18-Oct-2015].

[2] D. Básicos and D. E. L. a Asignatura, "' Redes De Sensores ,'" pp. 1–16, 2010.

[3] F. Zhao and L. Guibas, "Wireless Sensor Networks," 2004.

[4] E. O. Sosa, "Contribuciones al establecimiento de una red global de sensores inalámbricos interconectados," 2011.

[5] "Arduino - ArduinoBoardUno." [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardUno. [Accessed: 11-Oct-2015].

[6] "Software - Sensinode Ltd." [Online]. Available: http://sensinode.sivuviidakko.fi/EN/products/software.html. [Accessed: 18-Oct-2015].

[7] "Libelium - Connecting Sensors to the Cloud." [Online]. Available: http://www.libelium.com/. [Accessed: 11-Oct-2015].

[8] "Waspmote Mote Runner - 6LoWPAN Development Platform - IPv6 for the Internet of Things and Sensors | Libelium." [Online]. Available: http://www.libelium.com/products/waspmote-mote-runner-6lowpan/. [Accessed: 11-Oct-2015].

[9] S. R. MAROTO CANTILLO, "Desarrollo de aplicaciones basadas en WSN," 2010.

[10] W. Staff, "The Ultimate on-the-fly Network," Dec-2003. [Online]. Available: http://www.wired.com/2003/12/network-2/. [Accessed: 07-Nov-2015].

[11] D. F. Chicaiza García, “Estudio de las redes de sensores bajo el agua y sus principales aplicaciones,” 2009.

[12] J. Dignani, “Análisis del protocolo ZigBee,” 2012.

[13] S. R. Caprile, *Equisbí Desarrollo de aplicaciones con comunicación remota basadas en módulos ZigBee y 802.15.4.pdf*. Sergio R. Caprile, 2009.

[14] “Introducción a Zigbee y las redes de sensores inalámbricas.” [Online]. Available: about:reader?url=http%3A%2F%2Fwww.javierlongares.com%2Farte-en-8-bits%2Fintroduccion-a-zigbee-y-las-redes-de-sensores-inalambricas%2F. [Accessed: 11-Oct-2015].

[15] Ó. Vadillo Gutiérrez and others, “Provisión de servicios de la Internet de las Cosas sobre redes de sensores basadas en 6LoWPAN,” 2014.

[16] Libelium Comunicaciones Distribuidas S.L., “Waspmote Mote Runner Technical Guide wasp mote,” *Tech. Guid.*, p. 85, 2015.

ANEXOS

ANEXO I

PROFORMA EVALUATOR KIT

Libelium Comunicaciones Distribuidas S.L
Escatrón 16. C.P: 50014
Zaragoza (SPAIN)
Telf.: +34 976547492
Fax: +34 976473186
CIF: B99135832

Universidad de Las Américas

Ing. Patricio Arellano V.

Sede Colón: Av. Colón y 6 de Diciembre

- Quito

Ecuador

Cliente Universidad de Las

Representante David Bordonada

Proforma Invoice Número de documento 1006583 - 17/04/2015

Referencia	Descripción	Precio	Cantidad	Importe
WEVA	Evaluator Kit	3,350.00	1	3,350.00
Shipping	Shipping	176.00	1	176.00
	Exportaciones (%N=>0%)	3,526.00		0.00
Suma				***3526.00***

EUR 3526.00

Transferencia (Inmediato)

Bank: Santander EUR

IBAN: ES1700491824452210388257

Swift: BSCHESMM

GASTOS BANCARIOS COMPARTIDOS

Todos los precios están indicados en EUR

Incoterm: EXW

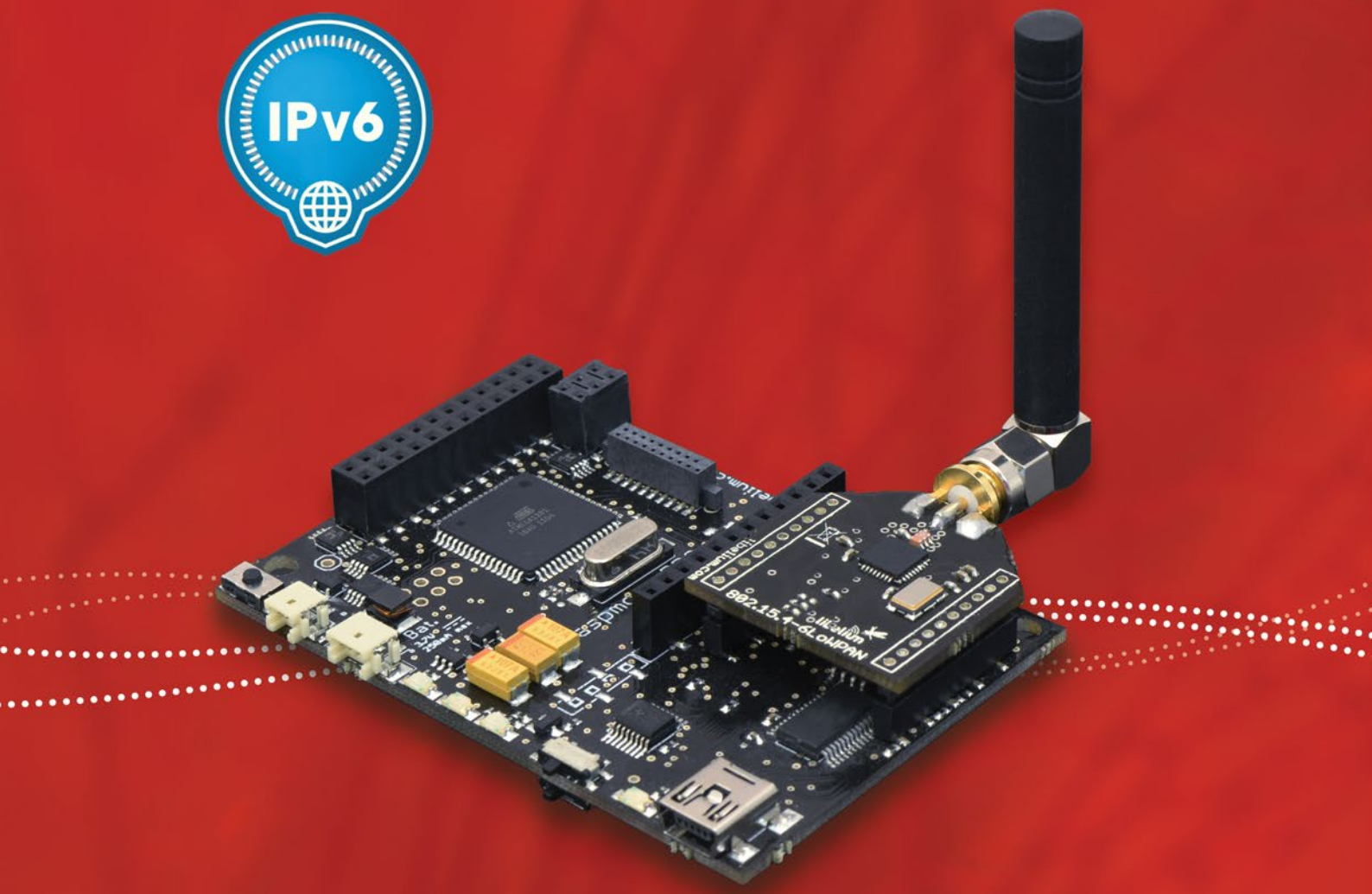
Términos y condiciones: http://www.libelium.com/legal

ANEXO II

MOTERUNNER TECHNICAL GUIDE

Waspmote

Mote Runner Technical Guide



libelium

waspmote

Document version: v4.1 - 01/2014

INDEX

1. Introduction

1.1. Concepts

IBM and Libelium have joined efforts to offer a unique IPv6 development platform for sensor networks and the Internet of Things (IoT). By integrating the IBM Mote Runner SDK on top of Libelium Waspmote sensor platform we get a unique and powerful tool for developers and researchers interested in 6LoWPAN / IPv6 connectivity for the Internet of Things.

6LoWPAN is the acronym of IPv6 over Low power Wireless Personal Area Network. This protocol offers encapsulation and header compression mechanisms that allow IPv6 packets to be sent to and received from over IEEE 802.15.4 based networks.

The Waspmote Mote Runner Development Platform allows Developers and Researchers to study, analyse and modify the 6LoWPAN protocol in order to improve it and test new routing algorithms on top.

1.2. Licences and Conditions of Use

- The Waspmote Mote Runner Kits consist of LIBELIUM Waspmote sensor nodes modified and ready to work ONLY with the IBM Mote Runner environment. If you want to use the Waspmote IDE or other modules specific to the original Waspmote platform, you should buy one of these kits. **http://www.libelium.com/products/waspmote-mote-runner-6lowpan/#buy**
- Although quite similar, Waspmote nodes and Waspmote Mote Runner nodes are not the same Hardware. For this reason Developers cannot interchange code, libraries or radio modules between both platforms".
- The Mote Runner SDK must be downloaded from the IBM website; it does not come with the Waspmote Mote Runner Kit. Mote Runner is distributed by IBM under an evaluation license which allows to be used at no cost for non-comercial purposes. It is highly recommended to read the license at the IBM Mote Runner Website before purchasing the kit. **http://www.zurich.ibm.com/moterunner/license.html**
- Waspmote nodes included in the Kit come without any firmware installed on them. Users have to compile it using the Mote Runner SDK.
- The platform is intended to be used at this first stage by Researchers and Developers who want to get in touch with 6LoWPAN connectivity. This is perfect for laboratory study, test beds and small deployments.
- The platform is not recommended for large scale deployments, commercial usages, or final product commercialization.
- Libelium has made available a thread in the FORUM to treat any issue related to the Waspmote Mote Runner Development Platform. However, Libelium will be responsible of answering ONLY the Hardware issues. Any Software, Networking or Data related issue will be answered by the IBM developers in this thread. **http://www.libelium.com/forum/viewforum.php?f=34**

1.3. Waspmote VS Waspmote Mote Runner

	Waspmote	Waspmote Mote Runner
Wireless Modules	XBee (802.15.4 2.4GHz, DigiMesh 2.4GHz, 868MHz, 900MHz, ZigBee), GPRS, 3G, RFID, WiFi Bluetooth	6LoWPAN radio modules (2.4 GHz and 868 MHz)
Sensor Boards	Yes	Yes, but Videocamera Board
OTA Programming	Yes, by Libelium (available on all XBees, GPRS, 3G and WiFi)	Yes, by IBM (available on the 6LoWPAN module)
SD Card	Yes (2GB)	No
IDE/SDK	Waspmote IDE (Open Source License)	Mote Runner (Evaluation License)
Programming language	Similar to C/C++	Java or C#
Sleep modes	Yes (three modes: sleep, deep sleep, hibernate)	Yes, but automated, controlled by the OS
Float variables	Yes. Native, easy handling	Only simulated handling
String variables	Yes. Native, easy handling	No, only byte arrays
Software Simulation	No	Yes
Code debugging	No	Yes
Programming type	Sequential	Reactive
Gateway	Two Options: Waspmote GW and Meshlium	Waspmote Mote Runner Gateway + PC for IPv4/IPv6 tunneling
Meshlium	Yes, Meshlium has modules to interoperate with Waspmote (XBee, 3G, WiFi, Bluetooth). Remote access, mesh networks, Sensor Parser, MySQL Database, ..	No, Meshlium cannot interoperate with Mote Runner
Enclosure	Yes (Plug & Sense! line)	No
Bootloader	Yes, preinstalled and ready to work	No. User must install it using the Mote Runner SDK. An AVR Programmer hardware is needed. (included in the kit)
Support, Technical Assistance	HW + SW + Networking by Libelium	HW by Libelium. SW + Networking by IBM.
Recommended for experimentation with 6LoWPAN	No	Yes
Recommended for industrial / commercial projects	Yes	No
Recommended for new users	Yes	No
Suitable for users with poor programming skills	Yes	No
Recommended for experts	Yes	Yes
Time to market	Small	Medium, experimental platform, not market-focused “as is”

More info about the Waspmote platform here:
http://www.libelium.com/products/waspmote

More info about the Waspmote Mote Runner Platform here:
http://www.libelium.com/products/waspmote-mote-runner-6lowpan

2. Waspmote Mote Runner Dev Kit

2.1. Box content

2.1.1. Waspmote Mote Runner 6LoWPAN Networking Kit

- 6 x Waspmote
- 6 x 2300 mAh Battery
- 6 x 6LoWPAN Module (2.4GHz or 868MHz)
- 6 x Antenna (2dB for 2.4GHz version and 0dB for 868MHz version)
- 1 x Expansion Board
- 1 x Ethernet Module
- 1 x Ethernet Cable
- 1 x Atmel AVR Programmer (with USB cable)
- 6 x Mini-USB Cable
- 6 x USB-220V Adapter

Figure 1: Waspmote Mote Runner 6LoWPAN Networking kit

2.1.2. Waspmote Mote Runner 6LoWPAN Lab Kit

- 6 x Waspmote
- 6 x 2300mAh Battery
- 6 x 6LoWPAN Module (2.4GHz or 868MHz)
- 6 x Antenna (5dB for 2.4GHz version and 4.5dB for 868MHz version)
- 6 x Mini-USB Cable
- 6 x USB-220V Adapter
- 1 x Expansion Board
- 1 x Ethernet Module
- 1 x Ethernet Cable
- 1 x Atmel AVR Programmer (with USB cable)
- 1 x Gases Board
- 1 x O_2 Sensor
- 1 x Humidity Sensor
- 1 x CO Sensor
- 1 x CO_2 Sensor
- 1 x Events Board
- 1 x Temperature Sensor
- 1 x Presence Sensor (PIR)
- 1 x Luminosity LDR Sensor
- 1 x Smart Cities Board
- 1 x Dust Sensor – PM – 10
- 1 x Smart Metering Board
- 1 x Ultrasound Sensor 0-100A
- 1 x Agriculture PRO Board
- 1 x Soil Temperature Sensor
- 1 x Weather Station WS-3000 (Anemometer + Wind + Pluviometer)
- 1 x Prototyping Board
- 1 x GPS Module
- 1 x Rigid Solar Panel 7V - 500mA
- 1 x Flexible Solar Panel 7.2V – 100mA

Figure 2: Waspmote Mote Runner 6LoWPAN Lab Kit

2.2. General and safety information

Important:

In this section, the term "Waspmote" encompasses both the Waspmote device itself and its modules and sensor boards.

- Please read carefully through the document "General Conditions of Libelium Sale and Use".
- Do not let the electronic parts come into contact with any steel elements, to avoid injuries and burns.
- NEVER submerge the device in any liquid.
- Keep the device in a dry place and away from any liquids that might spill.
- Waspmote contains electronic components that are highly sensitive and can be accessed from outside; handle the device with great care and avoid hitting or scratching any of the surfaces.
- Check the product specifications section for the maximum allowed power voltage and amperage range and always use current transformers and batteries that work within that range. Libelium will not be responsible for any malfunctions caused by using the device with any batteries, power supplies or chargers other than those supplied by Libelium.
- Keep the device within the range of temperatures stated in the specifications section.
- Do not connect or power the device with damaged cables or batteries.
- Place the device in a location that can only be accessed by maintenance operatives (restricted area).
- In any case, keep children away from the device at all times.
- If there is an electrical failure, disconnect the main switch immediately and disconnect the battery or any other power supply that is being used.
- If using a car lighter as a power supply, be sure to respect the voltage and current levels specified in the "Power Supplies" section.
- When using a battery as the power supply, whether in combination with a solar panel or not, be sure to use the voltage and current levels specified in the "Power supplies" section.
- If a software or hardware failure occurs, consult the Libelium Web Development section
- Check that the frequencies and power levels of the radio communication modules and the integrated antennas are appropriate for the location in which you intend to use the device.
- The Waspmote device should be mounted in a protective enclosure, to protect it from environmental conditions such as light, dust, humidity or sudden changes in temperature. The board should not be definitively installed "as is", because the electronic components would be left exposed to the open-air and could become damaged. For a ready-to-install product, we advise our Plug & Sense! line.

General:

- Read the "General and Safety Information" section carefully and keep the manual for future reference.
- Read carefully the "General Conditions of Sale and Use of Libelium". This document can be found at:
  http://www.libelium.com/development/waspmote/technical_service. As specified in the Warranty document, the client has 7 days from the day the order is received to detect any failure and report that to Libelium. Any other failure reported after these 7 days may not be considered under warranty.
- Use Waspmote in accordance with the electrical specifications and in the environments described in the "Electrical Data" section of this manual.
- Waspmote and its components and modules are supplied as electronic boards to be integrated within a final product. This product must have an enclosure to protect it from dust, humidity and other environmental interactions. If the product is to be used outside, the enclosure must have an IP-65 rating, at the minimum.
- Do not place Waspmote in contact with metallic surfaces; they could cause short-circuits which will permanently damage it.

Specific:

- Reset and ON/OFF button: Handle with care, do not force activation or use tools (pliers, screwdrivers, etc) to handle it.
- Battery: Only use the original lithium battery provided with Waspmote.
- Mini USB connection: Only use mini USB, mod. B, compatible cables.
- Solar panel connection: Only use the connector specified in the Power supplies section and always respect polarity.
- Lithium battery connection: Only use the connector specified in the Battery section and always respect polarity.
- GPS board connection: Only use the original Waspmote GPS board.
- XBee module connection: Waspmote allows the connection of any module from the XBee family, respect polarity when connecting (see print).
- Antenna connections: Each of the antennas that can be connected to Waspmote (or to its GPS board) must be connected using the correct type of antenna and connector in each case, or using the correct adaptors.
- USB voltage adaptors: To power and charge the Waspmote battery, use only the original accessories: 220V AC – USB adaptor and 12V DC (car cigarette lighter) – USB adaptor

Usage and storage recommendations for the batteries:

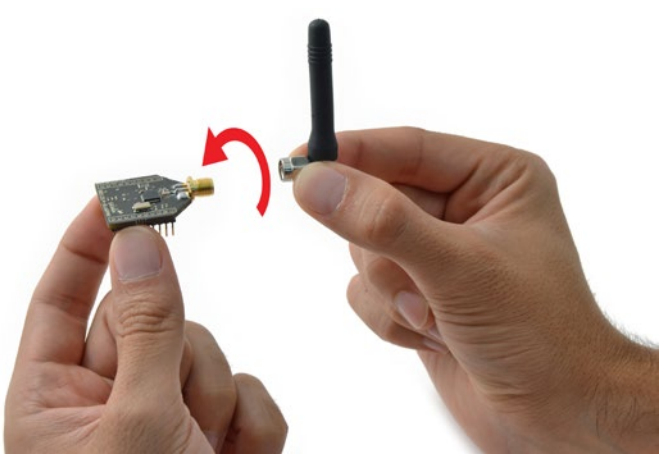
The rechargeable, ion-lithium batteries, like the ones provided by Libelium (capacities of 2300 and 6600 mAh), have certain characteristics which must be taken into account:

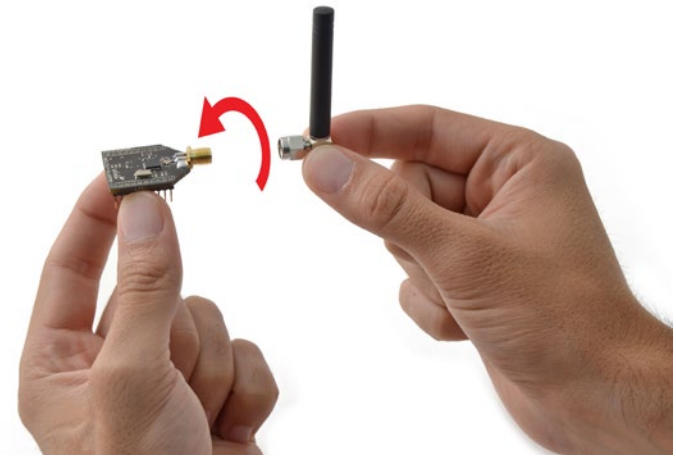
- Charge the batteries for 24 hours before a deployment. The aim is to have the charge of the batteries at 100% of their capacity before a long period in which they must supply current, but it is not necessary to improve the performance.
- It is not advised to let the charge of the batteries go below 20% of capacity, since they suffer stress. Thus, it is not advised to wait for the battery to be at 0% to charge it.
- Any battery self-discharges: connected to Waspmote or not, the battery loses charges by itself.
- Maximum capacity loss: as the charge and discharge cycles happen, the maximum charge capacity is reduced.
- Batteries work better in cool environments: their performance is better at 10 ºC than at 30 ºC.
- At temperatures below 0 ºC, batteries can supply current (discharge), but the charge process cannot be done. In particular:
  - discharge range = [-10, 60] ºC
  - charge range = [0, 45] ºC

It is not reccommended to have the non-rechargeable batteries (13000, 26000, 52000 mA·h) connected to Waspmote when the USB cable is conneted too. The reason is, Waspmote will try to inject current in them if the USB is connected. This is dangerous for the good working of a non-rechargeable battery. It could be damaged or even damage Waspmote. That is to say, when you need to upload code to Waspmote via USB, disconnect the battery if it is non-rechargeable. That applies to Waspmote OEM, but not to the Plug & Sense! line, since its hardware is modified to avoid this.

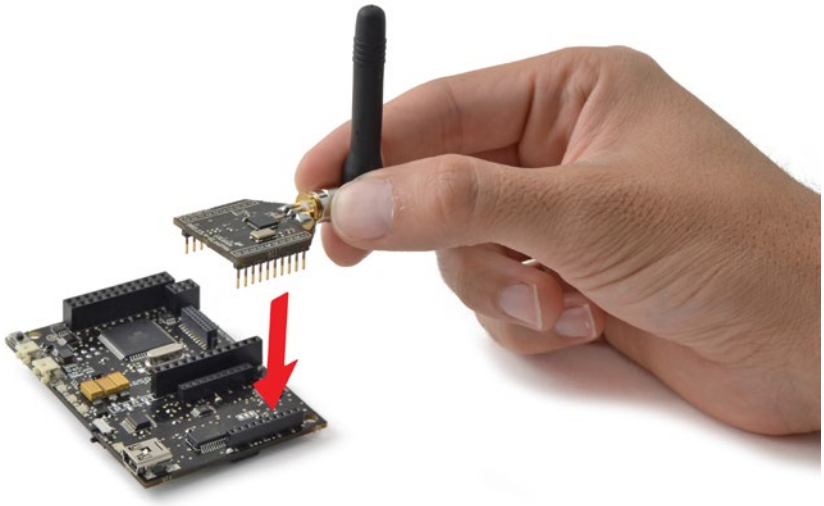
2.3. Assembly

- **Connect the antenna to the 6LoWPAN module**

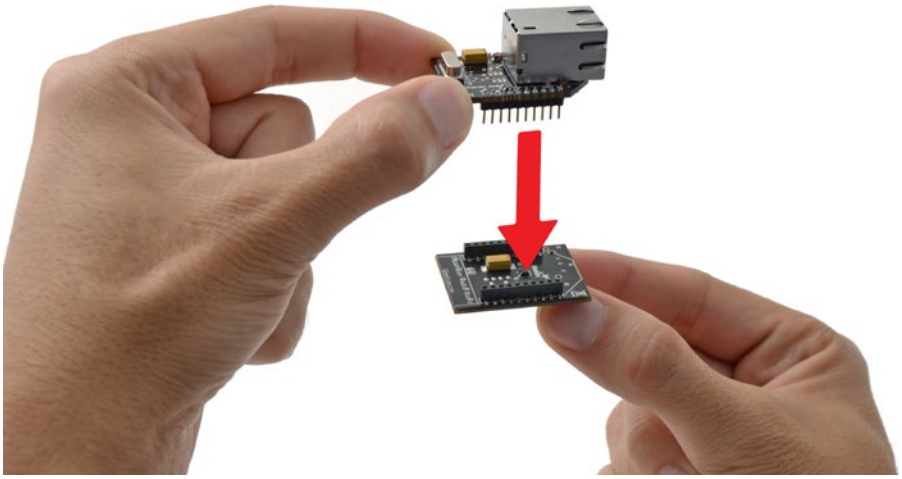




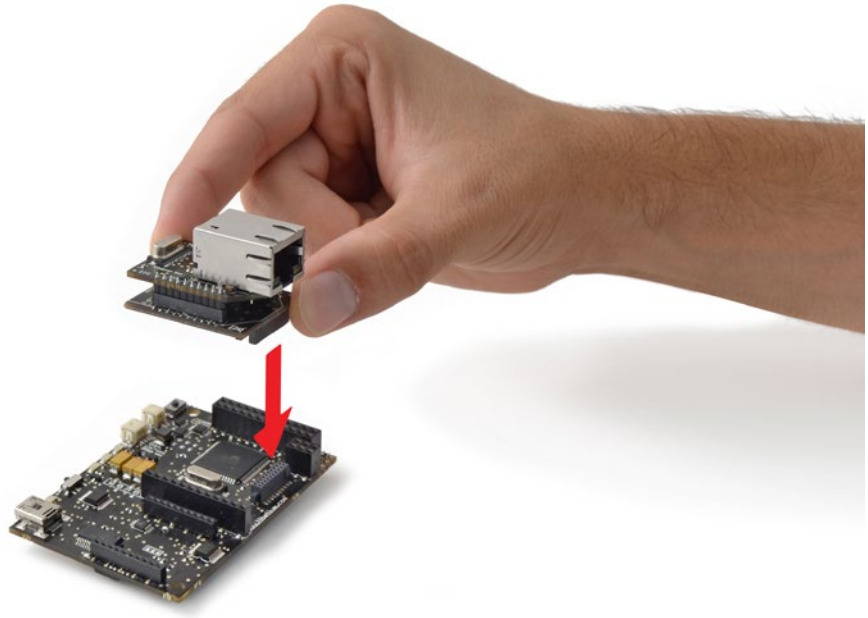
- **Place 6LoWPAN module in Waspmote**



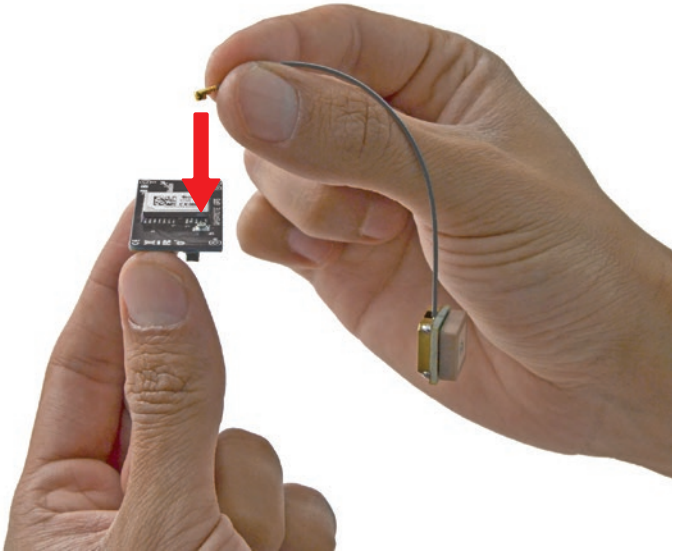
- **Place the Ethernet Module in the Expansion board**



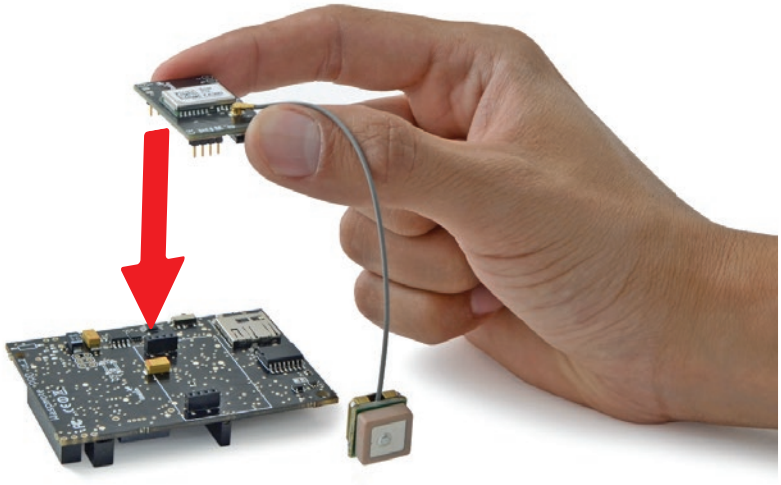
- **Place the Expansion Board + Ethernet Module in Waspmote**



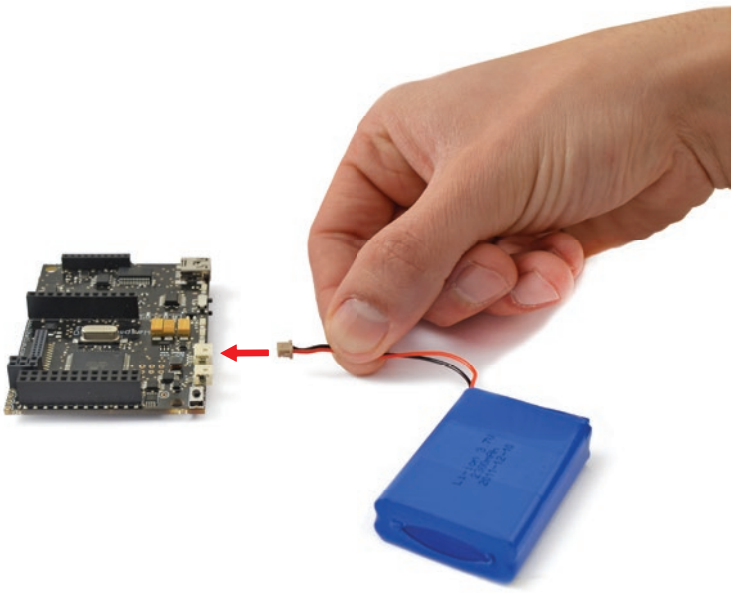
- **Connect the antenna in the GPS module**



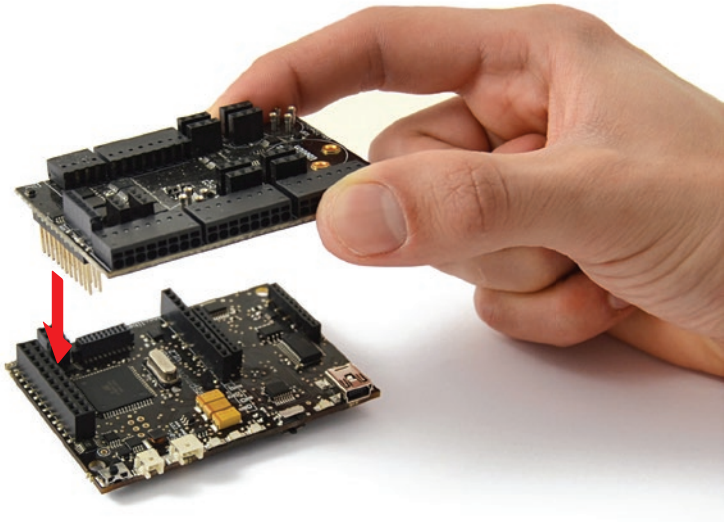
- **Place the GPS module in Waspmote**



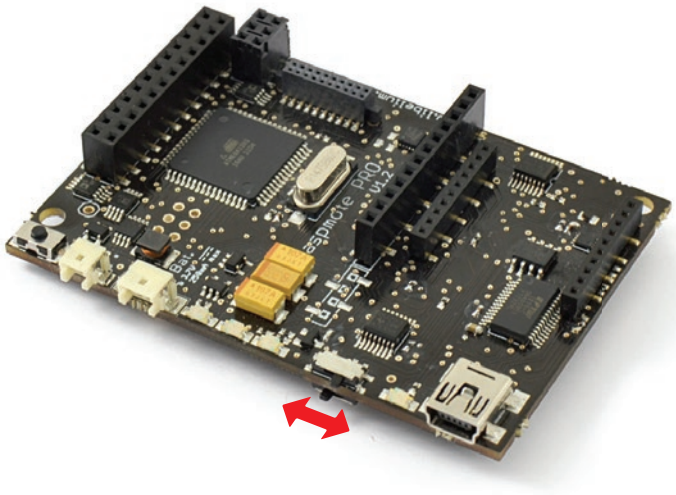
- **Connect the battery in Waspmote**



- **Connect the sensor board**

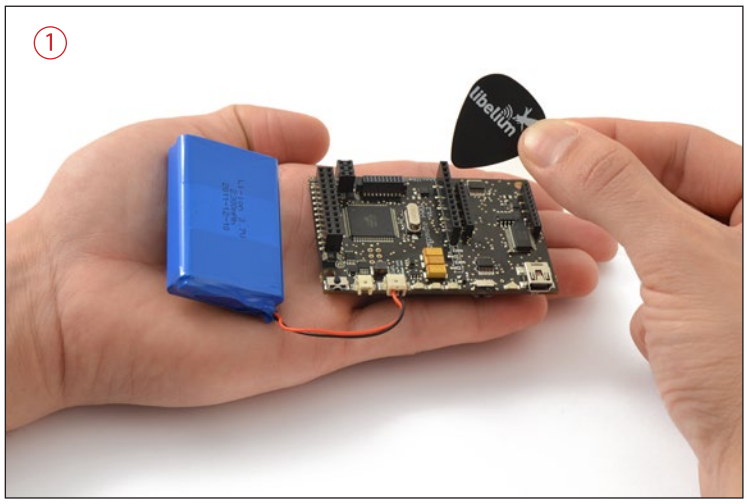


- **Switch it on**

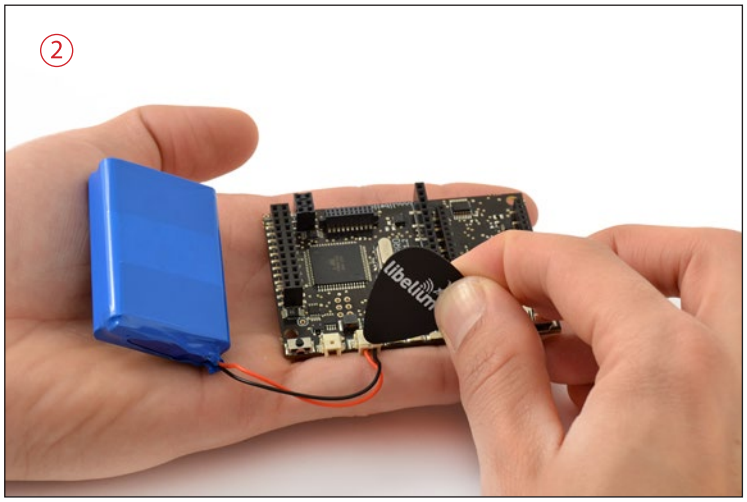


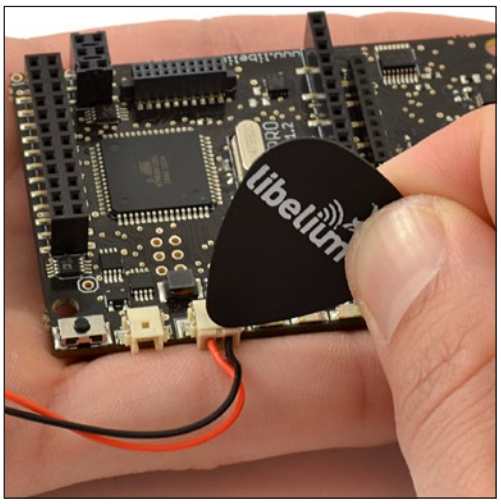
- **Waspmote battery disconnection**

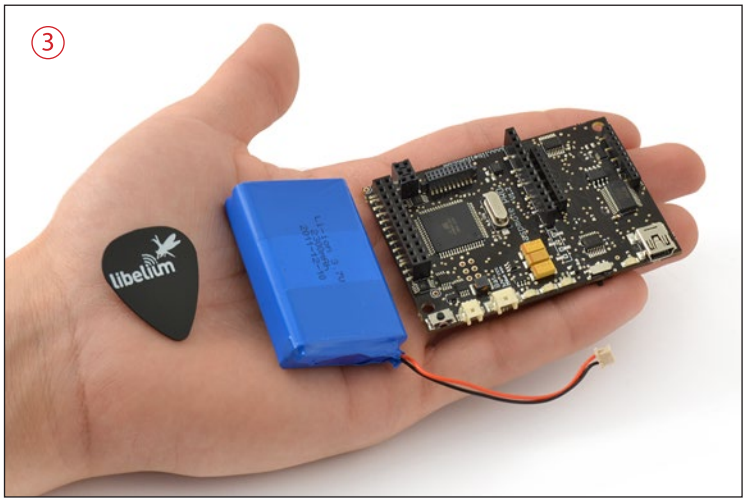
  Use the pick supplied by Libelium in order to disconnect Waspmote battery.



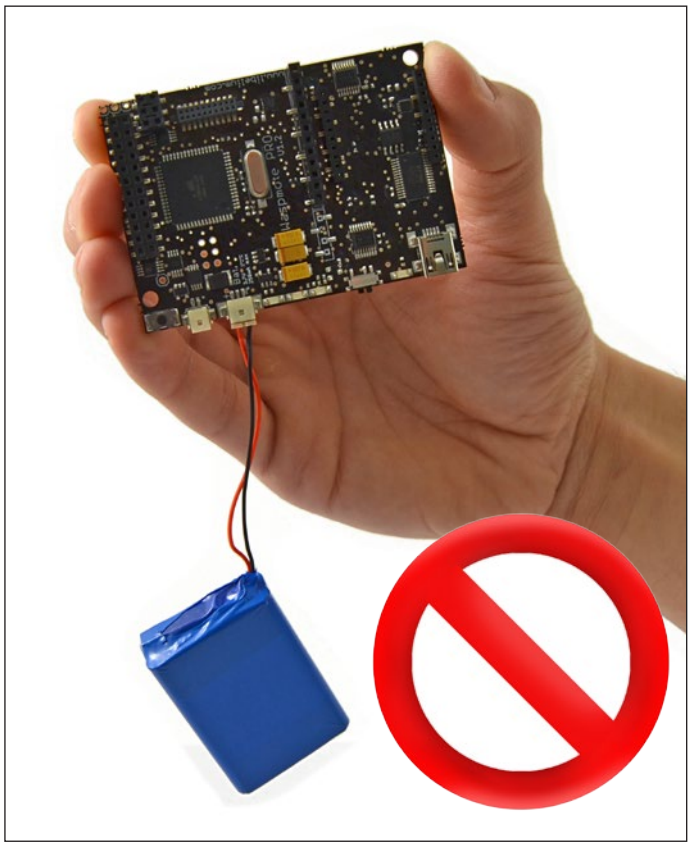
Insert the pick on the slot of the battery connector and pull straight out.
Do not pull the battery cables.



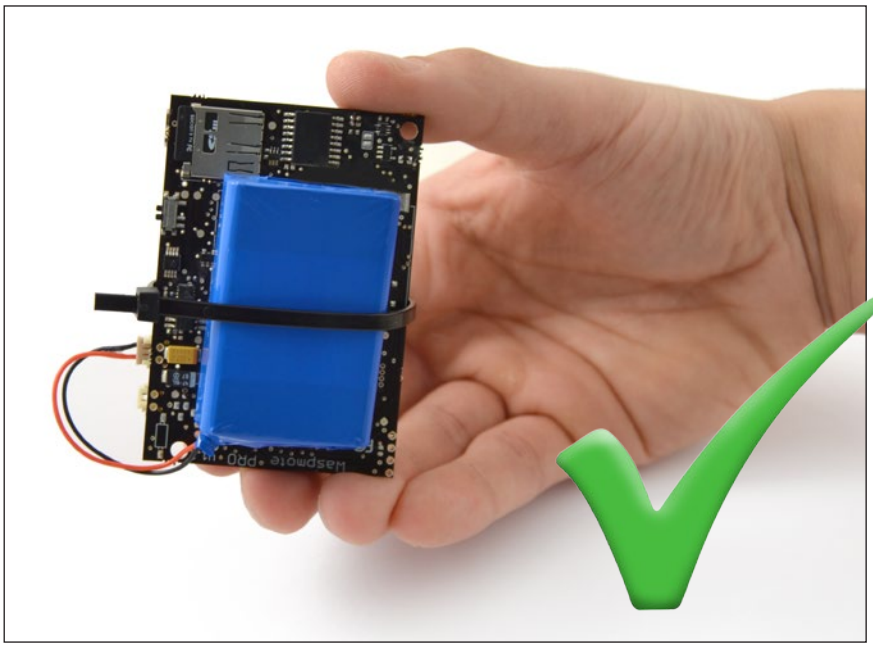




- **Battery handling instructions**
  In order to prevent from cable breaking, avoid leaving battery freely suspended.



Use a nylon clamp in order to attach battery to Waspmote.



2.4. Powering the Nodes

2.4.1. Battery

The battery included with the Waspmote Mote Runner Kits is a Lithium-ion battery (Li-Ion) with 3.7V nominal voltage and 2300mAh of capacity.

Waspmote has a control and safety circuit which makes sure the battery charge current is always adequate.

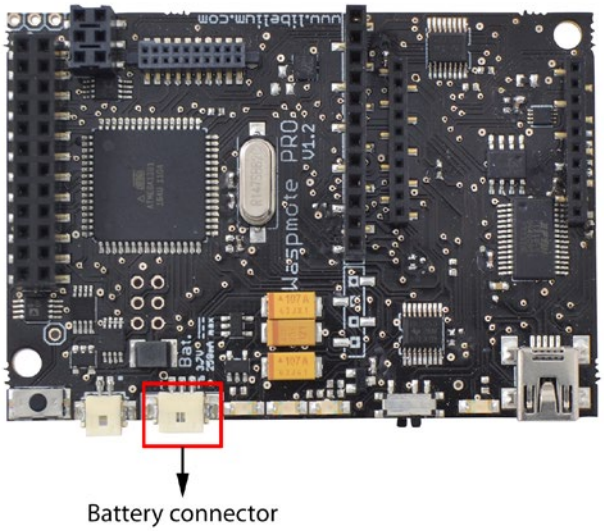


Figure 3: Battery connector

Battery connection

The figure below shows the connector in which the battery is to be connected. The position of the battery connector is unique, therefore it will always be connected correctly (unless the connector is forced).

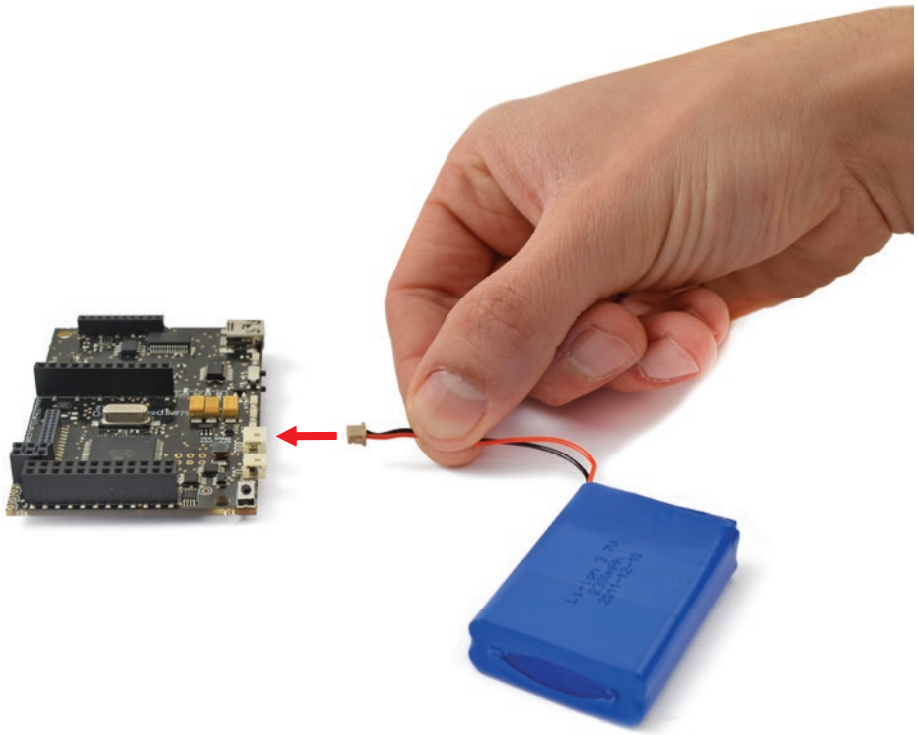


Figure 4: Battery connection

2.4.2. USB

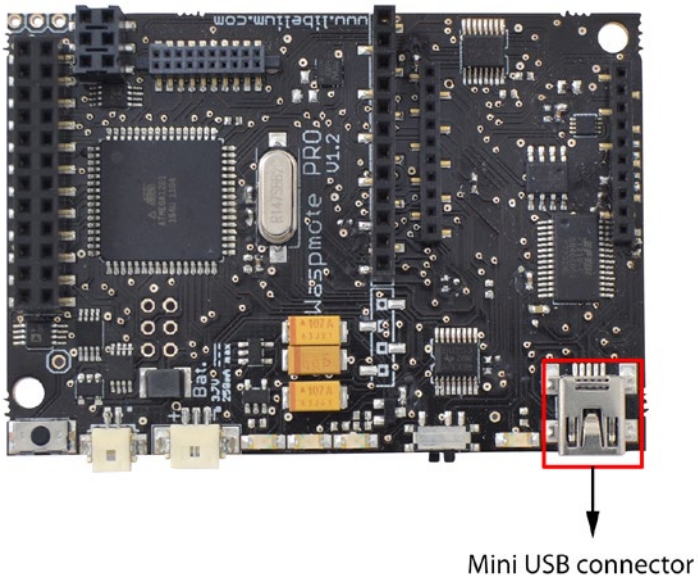


Figure 5: Mini USB connector

Waspmote's USB power sources are:

- USB to PC connection
- 110/220V to 5V USB Connection

> ***Note:*** *The nodes can not work with the radio and the USB plugged to a PC. The right way of powering them up once the firmware has been uploaded is by using the 110/220V to 5V USB Power Adaptor supplied in the kit.*
>
> *Obviously they can work too by using just the batteries provided without the need of connecting any cable (always they are with load).*

The models supplied by Libelium are shown below:

Figure 6: 110/220V to 5V USB Power Adaptor

2.4.3. Solar Panel

The solar panel must be connected using the cable supplied.

Both the mini USB connector and the solar panel connector allow only one connection position which must be respected without being forced into the incorrect position. In this way connection polarity is respected.

Solar panels up to **12V** are allowed. The maximum charging current through the solar panel is **280mA**.

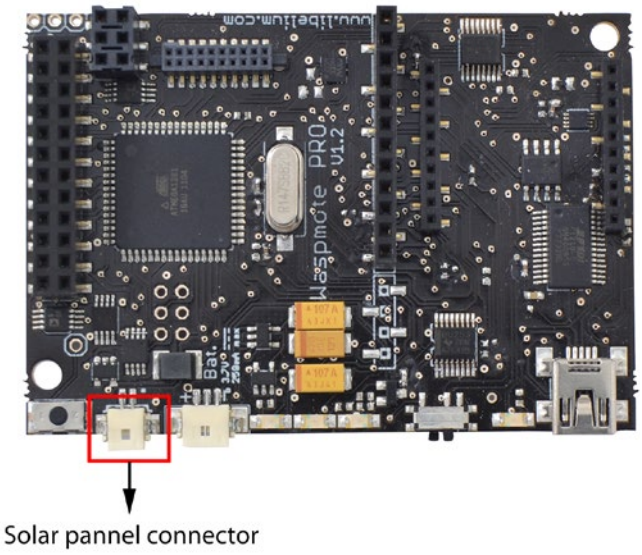


Figure 7: Solar panel connector

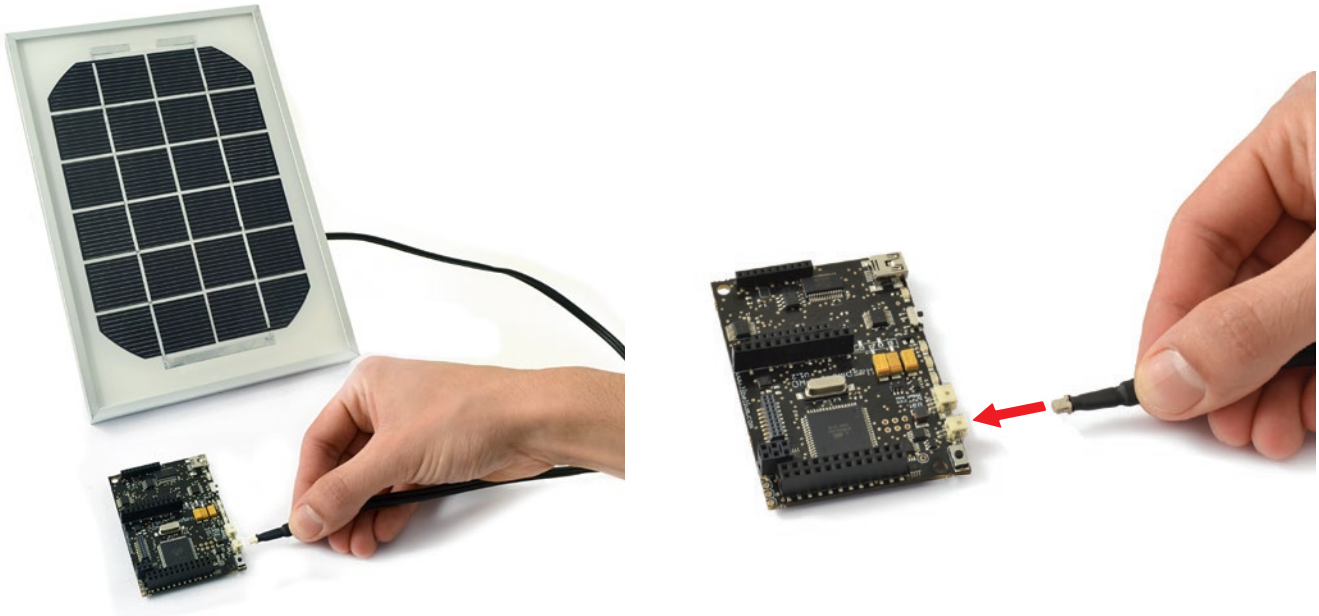


Figure 8: Solar panel connection

The models supplied by Libelium are shown below:

- **Rigid Solar Panel**

  7V - 500mA

Figure 9: Rigid Solar Panel

- **Flexible Solar Panel**

  7.2V - 100mA

Figure 10: Flexible Solar Panel

3. Waspmote's Hardware

3.1. Modular Architecture

Waspmote is based on a modular architecture. The idea is to integrate only the modules needed in each device. These modules can be changed and expanded according to needs.

The modules available for integration in Waspmote are categorized in:

- ZigBee/802.15.4 modules (2.4GHz, 868MHz, 900MHz). Low and high power.
- GSM/GPRS Module (Quadband: 850MHz/900MHz/1800MHz/1900MHz)
- 3G/GPRS Module (Tri-Band UMTS 2100/1900/900MHz and Quad-Band GSM/EDGE, 850/900/1800/1900 MHz)
- GPS Module
- Sensor Modules (Sensor boards)
- Storage Module: SD Memory Card

3.2. Specifications

- **Microcontroller:** ATmega1281
- **Frequency:** 14.7456 MHz
- **SRAM:** 8KB
- **EEPROM:** 4KB
- **FLASH:** 128KB
- **SD Card:** 2GB
- **Weight:** 20gr
- **Dimensions:** 73.5 x 51 x 13 mm
- **Temperature Range:** [-10ºC, +65ºC]

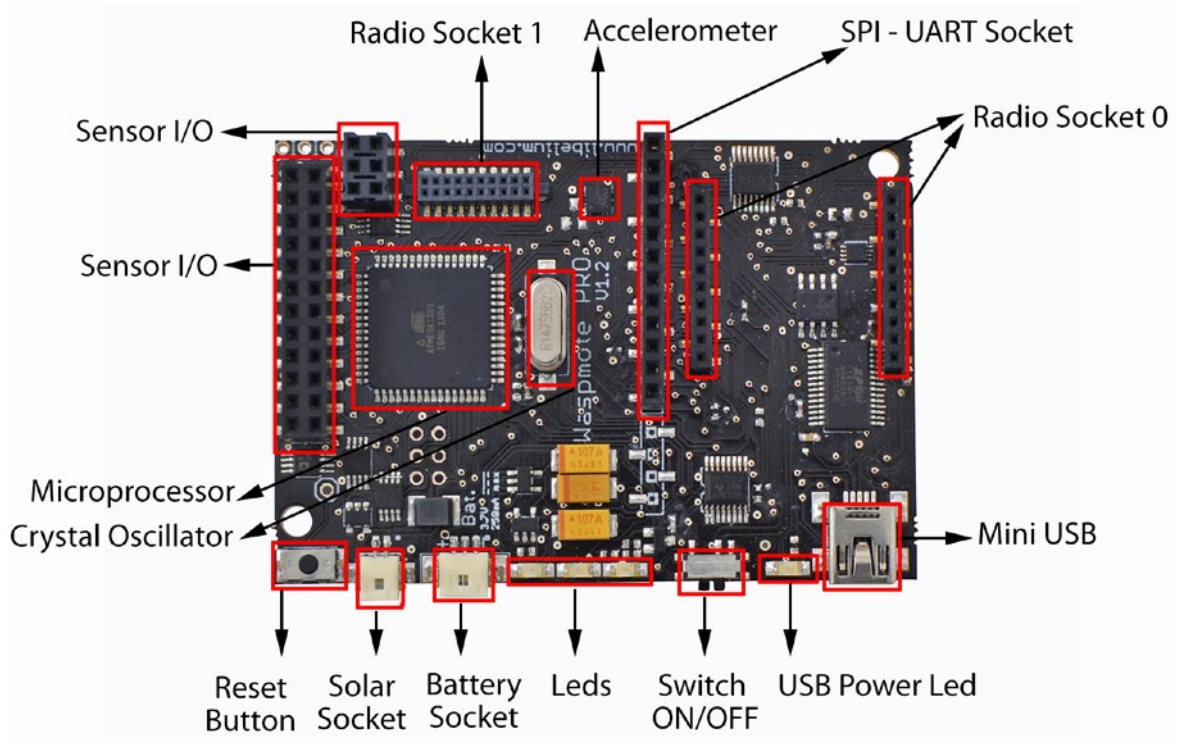


Figure 11: Main Waspmote components – Top side

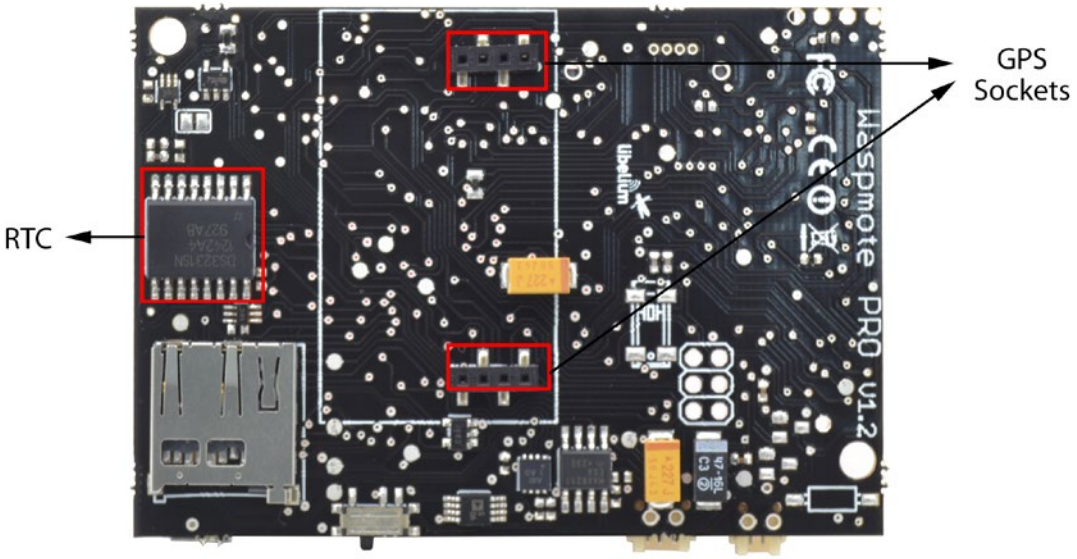


Figure 12: Main Waspmote components – Bottom side

3.3. Block Diagram

Data signals:

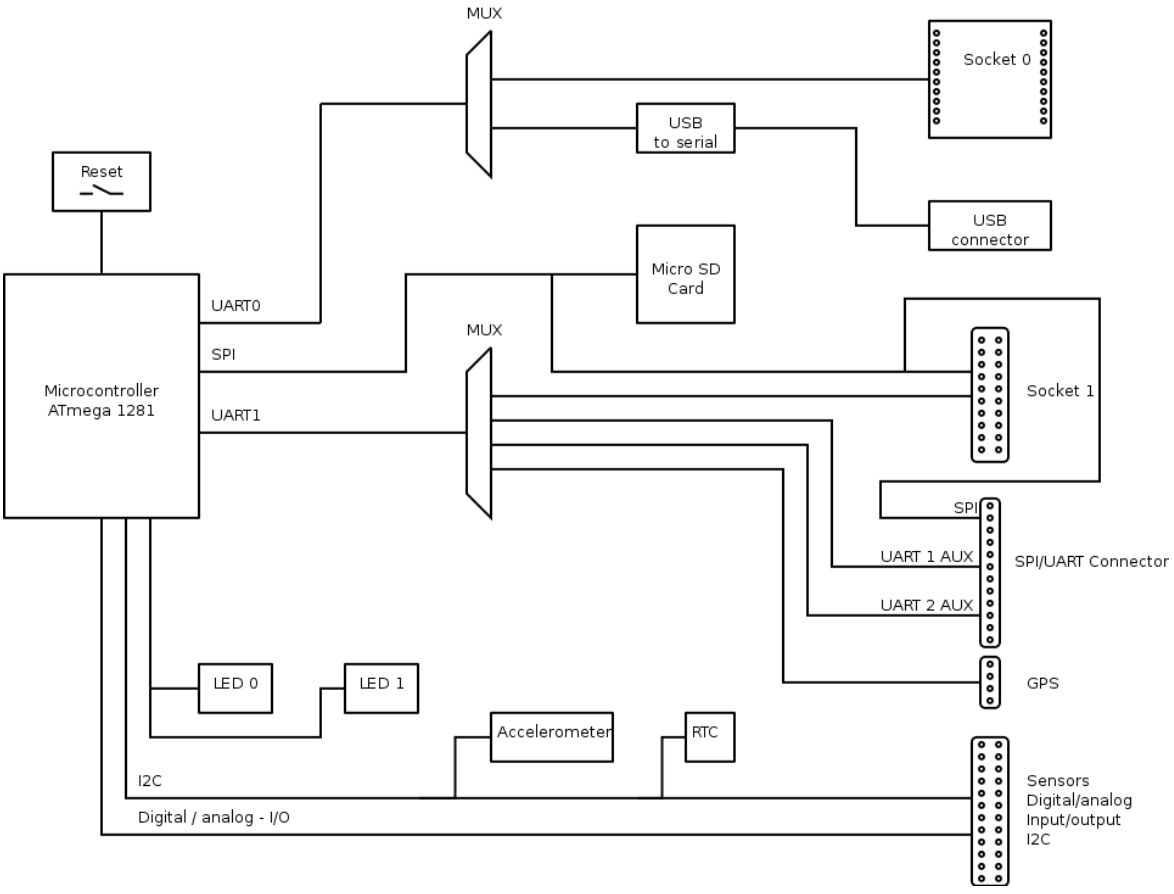


Figure 13: Waspmote block diagrams – Data signals

Power signals:

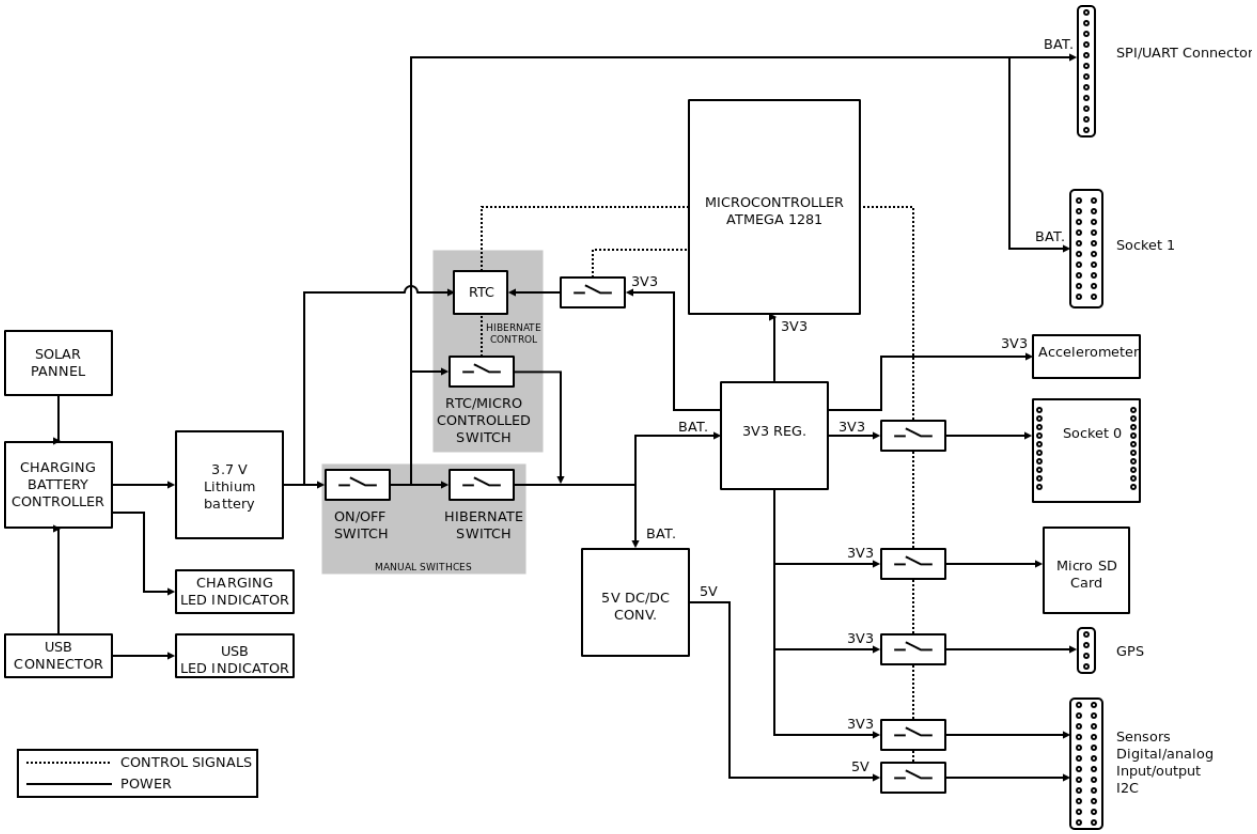


Figure 14: Waspmote block diagrams – Power signals

3.4. Electrical Data

Operational values:

- Minimum operational battery voltage 3.3 V
- Maximum operational battery voltage 4.2V
- USB charging voltage 5 V
- Solar panel charging voltage 6 - 12 V
- Battery charging current from USB 100 mA (max)
- Battery charging current from solar panel 280 mA (max)

Absolute maximum values:

- Voltage in any pin [-0.5 V, +3.8 V]
- Maximum current from any digital I/O pin 40 mA
- USB power voltage 7V
- Solar panel power voltage 18V
- Charged battery voltage 4.2 V

3.5. I/O

Waspmote can communicate with other external devices through the using different **input/output** ports.

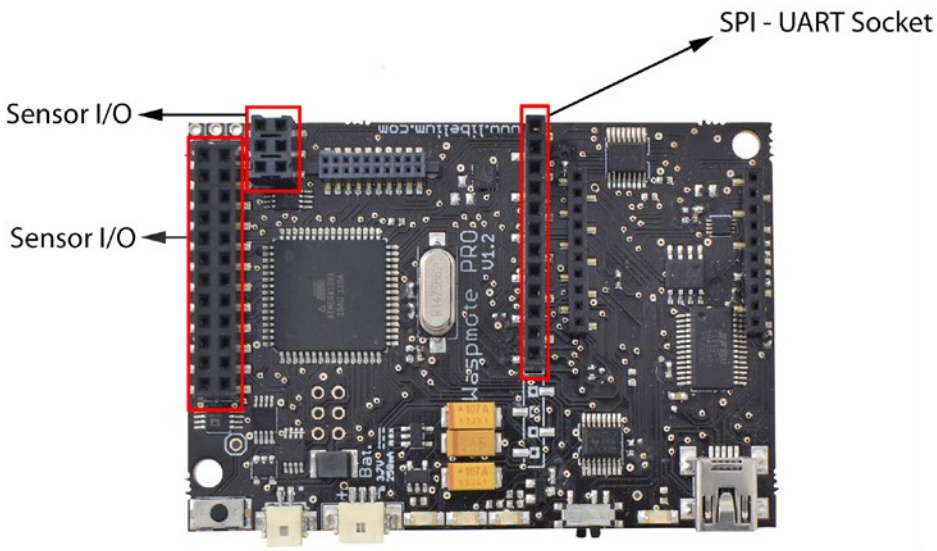


Figure 15: I/O connectors in Waspmote

Sensor connector:

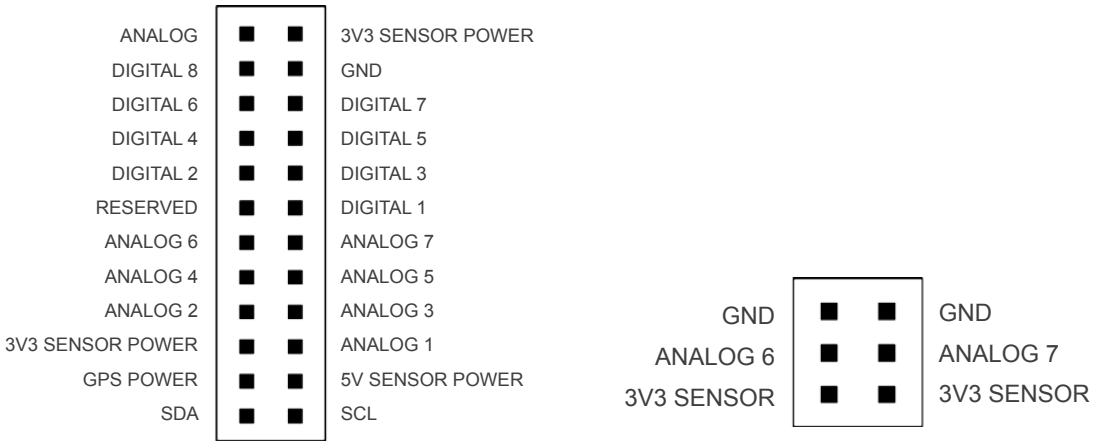


Figure 16: Description of sensor connector pins

Auxiliary SPI-UART connector:

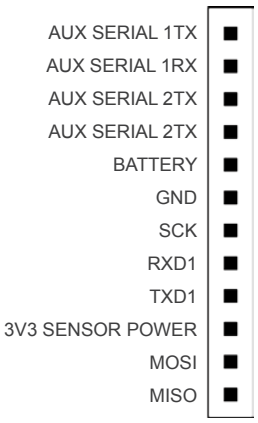


Figure 17: Description of auxiliary SPI-UART connector pins

3.5.1. Analog

Waspmote has 7 accessible analog inputs in the sensor connector. Each input is directly connected to the microcontroller. The microcontroller uses a 10 bit successive approximation analog to digital converter (ADC). The reference voltage value for the inputs is 0V (GND). The maximum value of input voltage is 3.3V which corresponds with the microcontroller's general power voltage.

There are two ways to obtain the input values, ayncrhonously and synchronously.

In both ways, the cannels to be read must be openned with the method `open()` of the ADC class, indicating them inside the channel map used as parameter.

```
static byte ADC_1 = 1;
static ADC adc = new ADC();
chmap = chmap | (int) (1 << ADC_1);
adc.open(chmap, GPIO.NO_PIN, 0, 0);
```

To obtain input values asynchronously, the method `read()` of the ADC class and the callback system might be used:

```
adc.setReadHandler(new DevCallback(null) {
  public int invoke(int flags, byte[] data, int len, int info, long time) {
      return <Class>.callbackMethod(flags, data, len, info, time);
  }
});
adc.read(Device.ASAP, 1, 0);
```

To obtain input values synchronously, the method `readChannel(byte channel)` of the ADC class might be used.

```
int value = adc.readChannel(ADC_4);
```

> Note that the code in this section is Java code.

3.5.2. Digital

Waspmote has digital pins which can be configured as input or output depending on the needs of the application. The voltage values corresponding to the different digital values would be:

- 0V for logic 0
- 3.3V for logic 1

There are two ways to control digital pins, ayncrhonously and synchronously.

In both ways, a GPIO instance must be created and opened as follows:

```
GPIO gpio = Gpio.getInstance(); //Unique instance of GPIO object
gpio.open();
```

For writing in the digital pins the first time, the method `configureOutput(byte pin, byte mode)` of the class GPIO must be used, and the following times, the method `doPin(byte ctrl, byte pin)` is the correct:

```
gpio.configureOutput(WASPMOTE.PIN_DIGITAL1, GPIO.OUT_SET);
gpio.doPin(GPIO.CTRL_CLR, WASPMOTE.PIN_DIGITAL1);
```

For reading from the digital pins asynchronously, the method `read()` from the GPIO class and the callback system must be used:

```
gpio.configureInput(WASPMOTE.PIN_DIGITAL1, GPIO.IRQ_DISABLED, (byte) 0);
gpio.setEventHandler(new DevCallback(null) {
  public int invoke(int flags, byte[] data, int len, int info, long time) {
       return <Class>.callbackMethod(flags, data, len, info, time);
  }
});
```

For reading from the digital pins syncrhonously, the method `doPin()` with the `CTRL_READ` control constant of the class GPIO must be used:

```
gpio.configureInput(WASPMOTE.PIN_DIGITAL1, GPIO.IRQ_DISABLED, (byte) 0);
int result = gpio.doPin(GPIO.CTRL_READ, WASPMOTE.PIN_DIGITAL1);
```

Note that the code on this section is Java code and assumes that com.libelium.common is imported.

3.5.3. UART

Tranmision to the UART has to be done using Cdev class as follows:

```
static{
  CDev xbee = new CDev();

  byte[] settings = csr.allocaByteArray((byte)4);
  settings[0] = WASPMOTE.UART0;   // UART0, plain (no CRC)
  settings[1] =WASPMOTE.UART_FRAME_DATA_BIT_8
       |WASPMOTE.UART_FRAME_STOP_BIT_1
       |WASPMOTE.UART_FRAME_PARITY_NONE;// frame (8-bits), no parity, 1-bit stop
  settings[2] = WASPMOTE.UART_BAUD_4800;// baud high (4800)

  xbee.open(WASPMOTE.DID_UART,settings,0,4);

       xbee.setTxHandler(new DevCallback(null) {
      public int invoke(int flags, byte[] data, int len, int info, long time){
          return <Class>.callbackMethod(flags, data, len, info, time);
      }
       });

  xbee.transmit(Device.ASAP,csr.s2b("+++"),(byte)0,1, 0);
}
```

3.5.4. I2C

The I2C communication bus is also used in Waspmote where two devices are connected in parallel: the accelerometer and the RTC . In all cases, the microcontroller acts as master while the other devices connected to the bus are slaves.

See complete I2C documentation at:
http://localhost:5000/gac/saguaro-system-11.4.htm?fqn=r:com.ibm.saguaro.system.I2C

Note that mrsh should be running in order to access the mentioned link.

3.5.5. SPI

The SPI port on the microcontroller is used for communication with the micro SD card. All operations using the bus are performed clearly by the specific library. The SPI port is also available in the SPI/UART connector.

See complete SPI documentation at:
http://localhost:5000/gac/saguaro-system-11.4.htm?fqn=r:com.ibm.saguaro.system.SPI

Note that mrsh should be running in order to access the mentioned link

3.5.6. USB

USB is used in Waspmote for communication with a computer or compatible USB devices. This communication allows the microcontroller's program to be loaded. For USB communication, microcontroller's UART0 is used. The FT232RL chip carries out the conversion to USB standard.

3.6. Real Time Clock-RTC

Waspmote has a built in Real Time Clock – RTC, which keeps it informed of the time. This allows Waspmote to be programmed to perform time-related actions such as:

"Sleep for 1h 20 min and 15sec, then wake up and perform the following action"

Or even programs to perform actions at absolute intervals, e.g.:

"Wake on the 5th of each month at 00:20 and perform the following action"

All RTC programming and control is done through the I2C bus.

Alarms:

Alarms can be programmed in the RTC specifying day/hour/minute/second. That allows total control about when the mote wakes up to capture sensor values and perform actions programmed on it. This allows Waspmote to be in the saving energy modes (**Deep Sleep** and **Hibernate**) and makes it wake up just at the required moment.

As well as relative alarms, periodic alarms can be programmed by giving a time measurement, so that Waspmote reprograms its alarm automatically each time one is triggered.

The RTC chosen is the Maxim **DS3231SN**, which operates at a frequency of **32.768Hz** (a second divisor value which allows it to quantify and calculate time variations with high precision).

The DS3231SN is one of the most accurate clocks on the market because of its internal compensation mechanism for the oscillation variations produced in the quartz crystal by changes in temperature (**Temperature Compensated Crystal Oscillator – TCXO**).

Most RTCs on the market have a variation of ± 20ppm which is equivalent to a 1.7s loss of accuracy per day (10.34min/year), however, the model chosen for Waspmote has a loss of just **± 2ppm**, which equates to variation of 0.16s per day (1min/year).

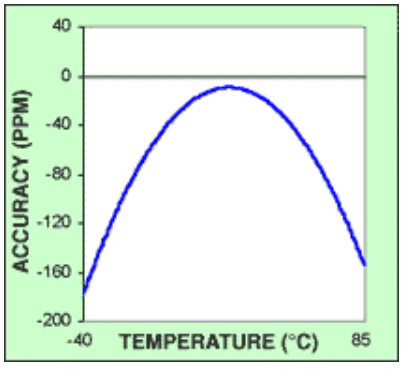


Figure 18: Uncompensated variation curve

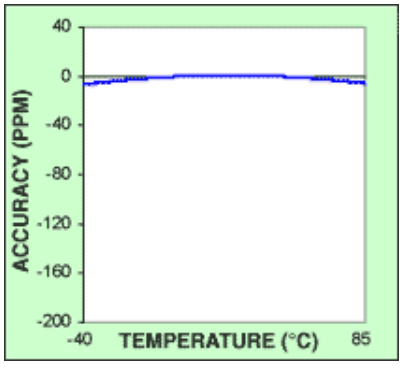


Figure 19: Compensated variation curve

Source: Maxim-ic.com

Figure on the left shows the temperature variation curve in a typical commercial clock, and in figure on the right that for the DS3231SN model built into Waspmote. As can be seen, variations in accuracy are practically zero at room temperature and minimal when moved to the ends of the temperature scale.

(For more information about clock calibrating methods in real time, consult web page:
http://www.maxim-ic.com/appnotes.cfm/an_pk/3566)

The recalibration process of the oscillation crystal is carried out thanks to the data received by the RTC's **internal temperature sensor**. The value of this digital sensor can be accessed by Waspmote through the I2C bus, which lets it know the **temperature of the board** at anytime in the range of **-40ºC to +85ºC** with an accuracy of 0.25°C. For more information about the acquisition of this value by the microprocessor, see the section "Sensors in Waspmote → Temperature".

Note: *the RTC's internal temperature sensor is only meant for the time derive compensation, but not for common air temperature sensing (we advise our Sensor Boards for that).*

The RTC is powered by the battery. When the mote is connected, the RTC is powered through the battery, but take into account that if the battery is removed or out of load, then time data will be not maintained. That is why we suggest to use RTC time like 'relative' and not 'absolute' (see Programming Guide for more info).

A coin or button battery is no longer needed. They had a limited life and therefore Waspmote can now have a much longer power life expectancy. This is so because the RTC is powered from the "main" battery which has a much bigger charge.

See complete documentation at section "RTC".

3.7. LEDs

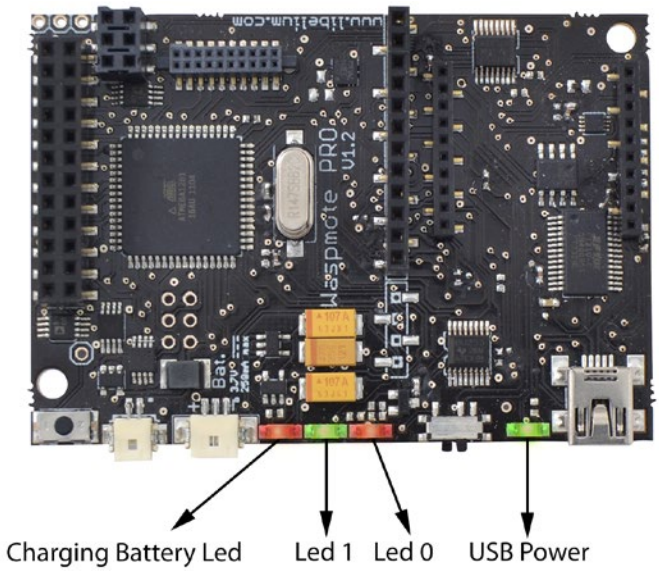


Figure 20: Visual indicator LEDs

- **Charging battery LED indicator**
  A red LED indicating that there is a battery connected in Waspmote which is being charged, the charging can be done through a mini USB cable or through a solar panel connected to Waspmote. Once the battery is completely charged, the LED switches off automatically.

- **LED 0 – programmable LED**
  A green indicator LED is connected to the microcontroller. It is totally programmable by the user from the program code. In addition, the LED 0 indicates when Waspmote resets, blinking each time a reset on the board is carried out.

- **LED 1 – programmable LED**
  A red indicator LED is connected to the microcontroller. It is totally programmable by the user from the program code.

- **USB Power LED indicator**
  A green LED which indicates when Waspmote is connected to a compatible USB port either for battery charging or programming. When the LED is on it indicates that the USB cable is connected correctly, when the USB cable is removed the LED will switch off automatically.

3.8. Communication Modules

3.8.1. 6LoWPAN Radios

6LoWPAN Radio (2.4GHz)

- Chipset: AT86RF231
- Frequency: 2.4GHz
- Link Protocol: IEEE 802.15.4
- Usage: Worldwide
- Sensitivity: -101dBm
- Output Power: 3dBm
- Encryption: AES 128b

Figure 21: 6LoWPAN Radio 2.4GHz

6LoWPAN Radio (868MHz)

- Chipset: AT86RF212
- Frequency: 868MHz
- Link Protocol: IEEE 802.15.4
- Usage: Europe
- Sensitivity: -110dBm
- Output Power: 10dBm
- Encryption: AES 128b

Figure 22: 6LoWPAN Radio 868MHz

3.8.2. Ethernet Module

- Chipset: W5100
- Protocol: Ethernet IPv4
- Physical: 100BASE-TX
- Services: TCP/IP, UDP/IP / ICMP
- Internal Buffer: 16KB

Figure 23: Ethernet Module

4. Architecture and System

IBM Mote Runner is a run-time platform and development environment for wireless sensor networks (WSN) currently under development at the IBM Zurich Research Laboratory, Switzerland. It consists of an on-mote environment, the so-called "firmware," with some standard libraries, an off-mote environment (including an edge server facilitating communication between the WSN and the outside world, a command shell permitting high-level control of the WSN, and source-level debugger for refinement of the application code running inside the WSN), and a set of development tools such as a command-line compiler and graphical integrated development environment (IDE).

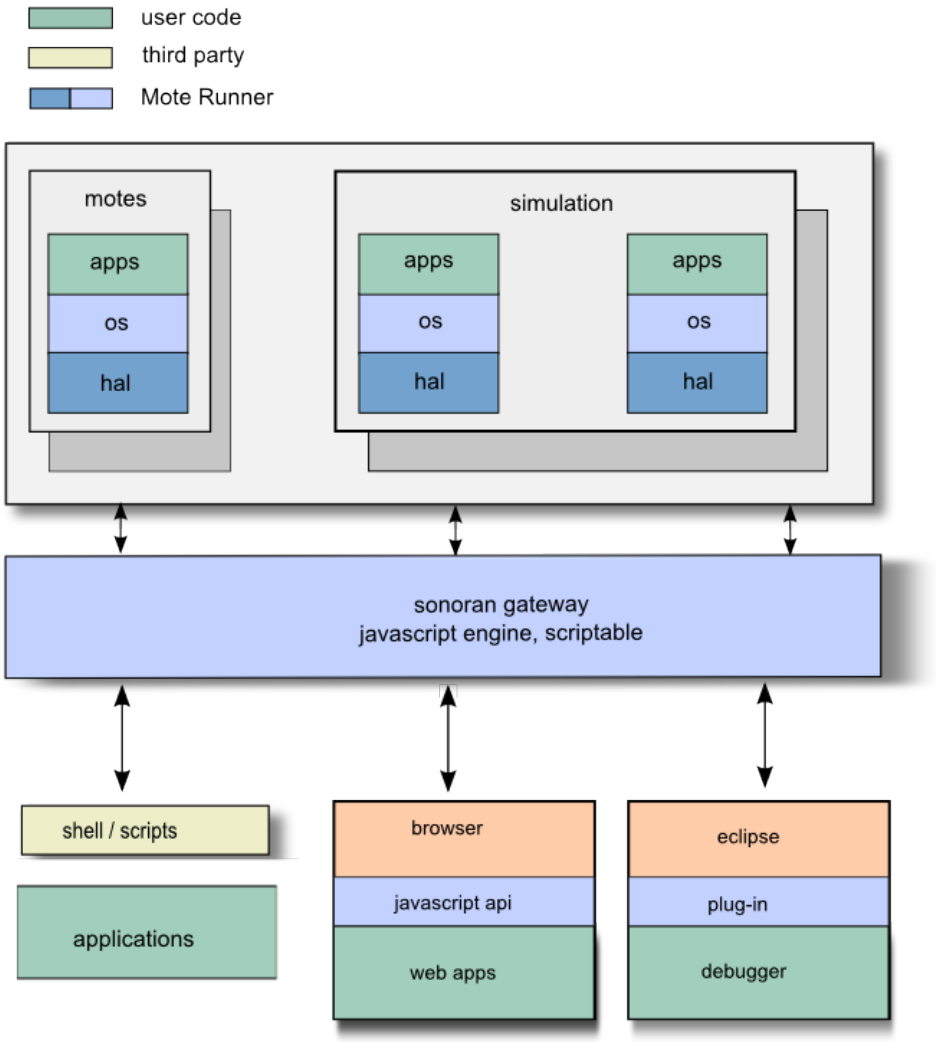


Figure 24: Overview of Mote Runner Ecosystem

4.1. Concepts

On the mote platform the Mote Runner firmware provides a run-time environment, which incorporates a virtual machine (VM) for executing byte codes and operating system (OS) to organize access to different devices and to schedule the various activities of applications.

Those byte codes are generated compiling an application with the Mote Runner Compiler. Mote Runner applications can be written in C# or Java.

In general applications are shielded from the underlying hardware and management functionality is provided by the VM/OS The Mote Runner VM provides only a single thread of execution. However, multiple applications can be running on top of the VM/OS at the same time. As a general rule, synchronous calls or busy looping will block the possible execution of other VM applications, and may not be energy efficient.

Managing assemblies on the motes is done through the sonoran gateway, a process running on a computer. This process can establish communication also with some other processes as for example scripts, browser or Eclipse IDE acting as a bridge between motes and the outside world.

Sonoran is a Javascript based programming and management environment for Mote Runner. It uses the V8 Javascript engine which is extended by a number of native functions (e.g. file and network IO) to service the Sonoran Javascript framework. On top of the core and Sonoran javascript framework, a number of interfaces exist allowing the interaction with Sonoran and motes. Sonoran features a command-line-shell (either on stdin/stdout or in a telnet session) and a HTTP interface (used by the Comote API) to setup and communicate with a network of motes. Additionally, Sonoran can execute user-defined scripts and applications building upon the base Sonoran functionality.

> See complete Sonoran documentation at:
> **http://localhost:5000/doc/system/index.html#system_scriptenv_0**
>
> Note that mrsh should be running in order to access the mentioned link

4.2. Programming modes

4.2.1. Reactive Programming (Asynchronous)

Applications implemented on top of the VM follow a reactive programming model. An application registers callbacks with operating services which will be invoked on certain events. An event starts with calling a specific method in the VM which probably will in turn call other methods. The application code may call the operating system to request certain actions but none of these calls will block the application. Long lasting operations create completion events leading again to callbacks. All VM callbacks execute to completion and will not interrupt other callbacks.

Such a reactive programming model is very resource conservative. The image below shows the basic interactions between the VM and the operating system. The central abstraction of the OS are devices. Devices represent sensors, actuators, communication devices like radio or serial and USB connections.

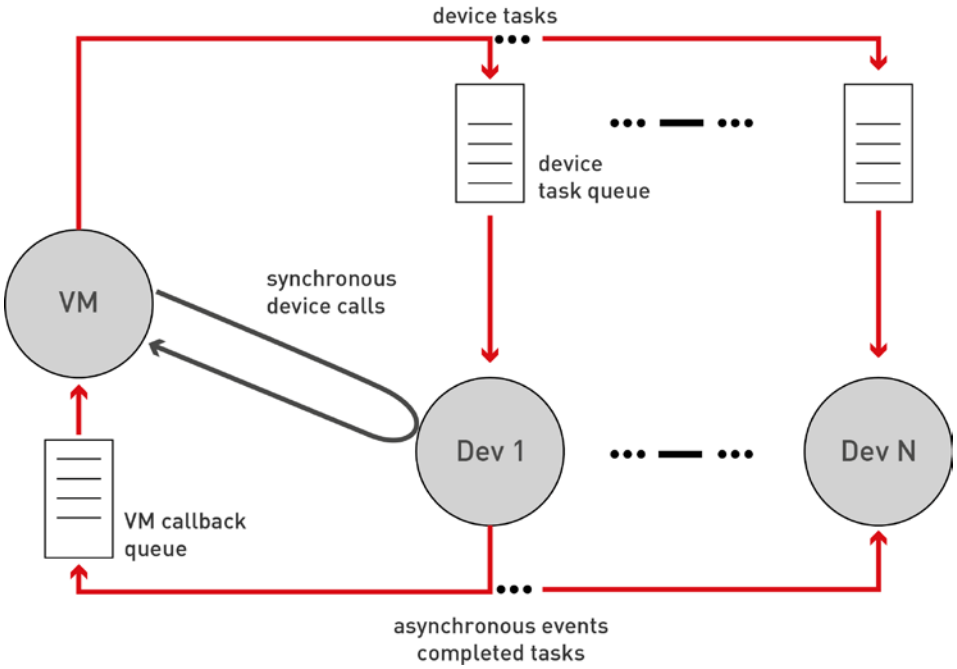


Figure 25: Reactive programming flow

4.2.2. Imperative programming (Synchronous)

Despite all applications can be programmed following a reactive programming model, some of the Mote Runner APIs offer synchronous functions for accessing some devices like:

- ADC
- GPIO
- I2C
- SPI

Is important to notice that synchronous calls will block the possible execution of other VM applications, and may not be energy efficient.

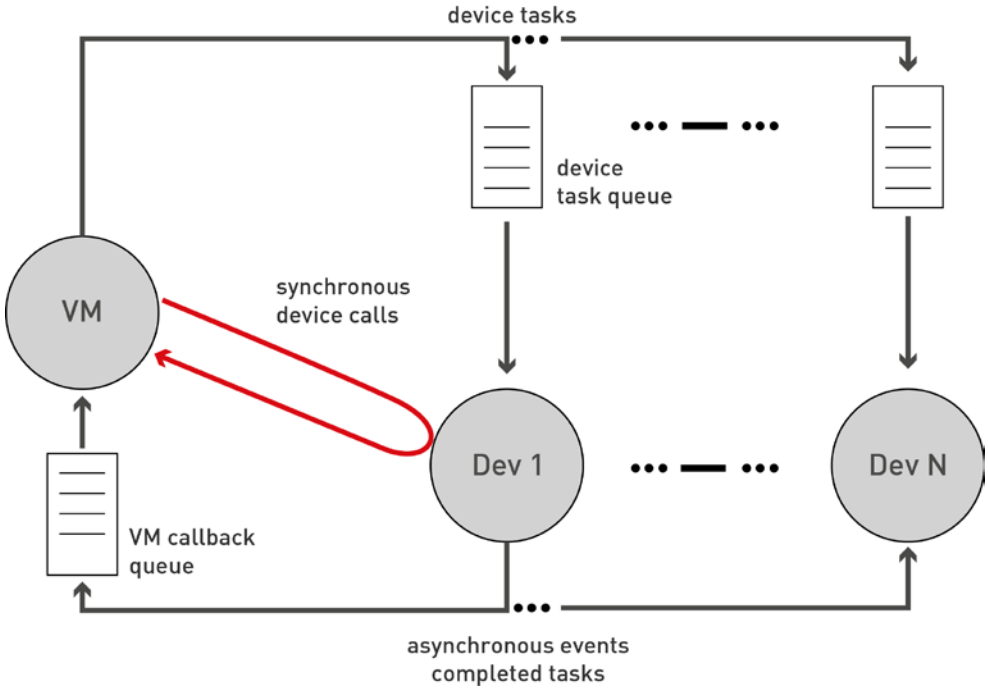


Figure 26: Imperative programming flow

4.3. Boot time connections

In order to connect with a Sonoran process running on a computer different kind of connections can be established.

As a general rule, at boot time, the Mote Runner OS scans for available communication connections. In case of the Waspmote the Mote Runner firmware will try the following connections in the given order.

1. **Ethernet:** found when the Ethernet extension board is attached to the Waspmote (Note the EEPROM of the Waspmote requires an IP configuration).

2. **Serial (LIP) via the USB port:** found when the Waspmote is attached to the PC via a USB cable, the mrsh started and, a connection to the mote exists in mrsh, e.g. `mote-create -p /dev/ttyUSB0`.

3. **WLIP via the Radio module (both 2.4 GHz and 868 MHz):** found when the RF2xx radio extension board is attached.

If one connection is found it will be selected as the default for communication via the LIP API and *further possible connections are not scanned:*

Note: *It is not possible to operate both a Mote Runner Radio device (with the RF2xx radio boards) and a LIP connection via USB.*

Note: *If an application opens the Radio device in the static initializer, then WLIP (which itself needs to open the Radio) will not be started up by the OS.*

At boot time the VM/OS will execute the static class initializers for all installed applications, one by one. If the class initializer for one application fails, e.g., because of an uncaught exception, the system will restart. Thus, *it is always good practice to catch all possible exceptions in the class initializers*.

An application can query the state of the LIP connection by using `LIP.isAvailable` call provide by the LIP API. Querying the state of LIP can be safely performed on the `onSystemInfo` callback event, which informs applications that the OS finished the boot sequence.

4.3.1. USB connection and Radio Modules

The USB connection for data transmission uses the same UART as the radio modules socket. For this reason data transmission through USB can't be done at the same time as data transmission through radio modules.

There is no problem with using the radio modules and the USB at the same time if the USB is only used to power the mote or charge the battery.

4.4. Networking

4.4.1. IPv4-UDP

4.4.1.1. IP configuration through MOMA

Using the moma-ipv4 command the IPv4 configuration can be queried and set. Without any parameters the command would print the current settings. All moma commands should be executed on Mote Runner shell (mrsh).

```
> moma-ipv4
IPv4 address:  192.168.1.100
IPv4 gateway:  192.168.1.1
UDP port:             4369 (0x1111)
```

4.4.1.2. IP configuration through source code

The IPv4 configuration, similar to the EUI configuration, can also be changed and queried programatically using the `getParam/setParam` calls in the Mote class provided by the saguaro-system API.

```
// set up my own IP address
byte[] myIPaddr = {192,168,1,100};
Mote.setParam(Mote.IPv4_ADDRESS,myIPaddr,4);
// set up my gateway address (used for routing)
byte[] myIPaddr = {192,168,1,1};
Mote.setParam(Mote.IPv4_GATEWAY,myIPgw,4);
// set the UDP server port, which would be used by the udp-create command
byte[] myUDPport = {0x11,0x11}; // little endian = corresponds to 4369
Mote.setParam(Mote.IPv4_UDPPORT,myUDPport,2);
```

See complete Mote class documentation at:

http://localhost:5000/gac/saguaro-system-11.4.htm?fqn=r:com.ibm.saguaro.system.Mote

Note that mrsh should be running in order to access the mentioned link

4.4.2. IPv6-UDP

> See complete IPv6 documentation at:
> "6LoWPAN Network" section

4.4.3. Networking between sonoran and motes

4.4.3.1. Mote Runner Shell (mrsh) Sockets

For simulated mote as well as a hardware motes, which are connected via LIP to mrsh, LIP/UDP messages are routed through the Javascript environment and forwarded accordingly. The socket commands can be used on the 'mrsh' command line to send UDP messages to motes and to print received UDP messages. The following command creates a new socket in the shell named "SOCK".

```
> sock-bind SOCK
```

The following creates a new simulation process and new mote.

```
> sag-start; mote-create
```

The following sends a message to mote 'a0' and 'mrsh' prints the immediately following message on the console.

```
> a0 sock-send SOCK 0 03
  16:27:629: DSTPORT 206    SRCPORT 0     SRC 02-00-00-00-00-DF-60-00
  00: 83 00 0B 03 49 2C 0E 73 61 67 75 61 72 6F 2D 73 ....I,.saguaro-s
  …
```

Destination port 0 of the command is the MOMA port and the bytes sent form the `moma-list` message as described in the MOMA section.

4.4.3.2. Mote Runner Shell (mrsh) as a UDP/LIP bridge

The `udp-bridge` command in mrsh can be used to exchange LIP messages between an external host and a connected mote both simulated and hardware.

The following command opens a UDP port which is used to forward UDP messages to a mote, in this case the l0 mote.

```
> l0 udp-bridge 61234
```

LIP messages received from mrsh on port 61234 are forwarded to the mote, host and source port in the LIP message modified. Messages from the mote to the UDB bridge are forwarded to the external host (i.e. the host the UDP bridge received the last message from).

4.4.4. Networking on mote applications

For efficiency reasons, as a general rule, Mote Runner uses UDP-based communication to and from motes. Moreover, applications have a uniform interface for communication using LIP which abstracts from the actual communication channel (be it serial, Ethernet, or WLIP).

The format of LIP messages is as follows

[4] IPv4 address: source, when a message is received, destination when a message is sent

[2] UDP port: source, when a message is received, destination when a message is sent

[1] Application port (used when receiving a message)

… **payload**

To receive LIP messages, an application needs to either open a port using the `LIP.open` call, or simply register a callback using the `Assembly.setOnData` call.

```
…
   Assembly.setDataHandler(onLipData);
   LIP.open(Defs.LIP_ASM_MR_PORT);
 …
 private static int onLipData (uint proto, byte[] data, uint len) {
        // Handle LIP message received in data
        …
        // Send a response by returning number of bytes in data to send background
        return n;
 }
```

A mote application can send a LIP message using `LIP.send`. In the header, the IPv4 address and UDP port can be specified:

```
// set the IP address
byte [] udpDestination = {192, 168, 1, 2, 0, 0};
// set the UDP port
Util.set16le(udpDestination,4,61234);
// send the message
LIP.send(udpDestination,6,message,0,(uint)message.length);
```

Note that the 7-byte LIP header can be prefixed by an additional header depending on the communication channel. The offset of the LIP application port can be queried using the `LIP.getPortOff` call.

See complete LIP class documentation at:
http://localhost:5000/gac/saguaro-system-11.4.htm?fqn=r:com.ibm.saguaro.system.LIP

Note that mrsh should be running in order to access the mentioned link

4.5. Timers

The hardware timers available on the underlying hardware platform are abstracted using the Mote Runner.

The granularity of the timers are the underlying MCU ticks. In case of the Waspmote, a tick is roughly equivalent with 1 us. However, as underlying hardware may change, applications should use the corresponding conversion functions from SI units to hardware ticks when dealing with timers or scheduling of tasks.

```
Timer timer = new Timer();

//Set the callback for this timer
timer.setCallback(new TimerEvent(null){
    public void invoke(byte param, long time){
        <name_of_the_class>.name_of_callback_function(param,time);
    }
});

// convert 2 seconds to the platform ticks
INTERVAL = Time.toTickSpan(Time.SECONDS, 2);

// set a new alarm in 2 seconds from now
timer.setAlarmBySpan(INTERVAL);
```

See complete Timer API at:
http://localhost:5000/gac/saguaro-system-11.4.htm?fqn=r:com.ibm.saguaro.system.Time

Note that mrsh should be running in order to access the mentioned link

4.6. Watchdog

In general hardware platforms provide a watchdog mechanism. In Mote Runner applications can enable this mechanism using the `Mote.watchdog` API call.

By default, the watchdog is not started. Once, started the watchdog must be restarted periodically.

> See complete Watchdog documentation at:
> **http://localhost:5000/gac/saguaro-system-11.4.htm?fqn=com.ibm.saguaro.system.Mote.watchdog(u)**
>
> Note that mrsh should be running in order to access the mentioned link

4.7. RTC

The `com.libelium.rtc` is the responsible of managing the use of the real time clock attached in the motes.

For proper use of this library, the common assembly is needed to be loaded into the Waspmote. See "Common Library" section.

To build the library, this set of commands must be executed in the Mote Runner shell (mrsh):

```
>  cd /API path/com/libelium/rtc
> mrc --assembly=rtc-1.0 --system=waspmote --ref=../common/common-1.0 RTC.java
```

To upload the compiled assembly to the Waspmote, run the following commands:

```
> cd/API path/com/libelium/rtc
> moma-load rtc
```

These are the methods implemented in the library:

```
public void setDate(byte year,byte month, byte date, byte day, byte hours, byte minutes,
byte seconds);

public byte[] getDate();

public void setAlarm1(Callback cb, byte mode, boolean offset, byte date, byte day, byte
hours, byte minutes, byte seconds);

public byte[] getAlarm1();

public void setAlarm2(Callback cb, byte mode, boolean offset, byte date, byte day, byte
hours, byte minutes);

public byte[] getAlarm2();

public static void disableAlarm(byte alarm);

public long getTemperature();
```

And a simple example of the use of them:

```
static{

   static RTC rtc = RTC.getInstance(); //Unique instance of the RTC Object

   //Set the actual date of the RTC to Saturday, 2010-06-04 at 02:03:50
   rtc.setDate((byte)10,(byte) 6,(byte) 4, (byte)7,(byte) 2, (byte)3,    (byte)50);

   //Set the alarm 1 to be fired every second (see the alarm modes on
   //the documentation of the methods)
   rtc.setAlarm1(new Callback(null){
    public int invoke(byte[] result, int len) {
         return PruebasRTC.alarmCallback(result, len);
    }
   }, (byte)Constants.RTC_ALM1_MODE1, false,(byte) 0,(byte) 0, (byte)0,(byte) 0, (byte)0);

   //Set the alarm 2 to be fired on the minute 1 of every hour
   rtc.setAlarm2(new Callback(null){
    public int invoke(byte[] result, int len) {
         return PruebasRTC.alarmCallback2(result, len);
    }
   }, (byte)Constants.RTC_ALM2_MODE2,false,(byte) 0,(byte) 0, (byte)0,(byte) 1);

   //Get the actual info of the alarm 2 and print it on the console
   byte[] alarma2 = rtc.getAlarm2();
   Logger.appendString(csr.s2b("minute: "));
   Logger.appendInt(alarma2[0]);
   Logger.appendString(csr.s2b(" hour: "));
   Logger.appendInt(alarma2[1]);
   Logger.appendString(csr.s2b(" date: "));
   Logger.appendInt(alarma2[2]);
   Logger.flush(Mote.INFO);
}

public static int alarmCallback(byte[] result, int len){
      //Get the actual Date stored on the RTC and print it on the console
      byte[] date = rtc.getDate();
  Logger.appendString(csr.s2b("second: "));
  Logger.appendInt(date[0]);
  Logger.appendString(csr.s2b(" minute: "));
  Logger.appendInt(date[1]);
  Logger.appendString(csr.s2b(" hour: "));
  Logger.appendInt(date[2]);
  Logger.appendString(csr.s2b(" date: "));
  Logger.appendInt(date[3]);
  Logger.appendString(csr.s2b(" month: "));
  Logger.appendInt(date[4]);
  Logger.appendString(csr.s2b(" year: "));
  Logger.appendInt(date[5]);
      Logger.flush(Mote.INFO);
  LED.setState((byte)0,(byte)(LED.getState((byte)0)^1));
  return 0;
}
public static int alarmCallback2(byte[] result, int len){
  LED.setState((byte)1,(byte)(LED.getState((byte)1)^1));
  return 0;
}
```

Further documentation of the methods can be found embedded within the code and in a generated javadoc delivered with the libraries.

See more examples at:

http://www.libelium.com/development/waspmote/examples/?cat=mr-general&subcat=mr-rtc

4.8. Interruptions

The low-level hardware interruptions are entirely handled by the VM/OS. Applications will only received corresponding callback events for which they have registered. In the asynchronous callback, the applications deal with the processing of the corresponding data, generated by the interrupts. As a general rule, synchronous calls or busy looping will block the possible execution of other VM applications, and may not be energy efficient.

4.9. Sleep mode

To save energy, based on the tasks ahead, the Mote Runner VM/OS decides on the best sleep mode for the underlying hardware system, including sensor and communication devices. While, applications do not have a direct API access to the hardware sleep modes of the MCU or the Radio device, the applications do influence the sleep mode by their queued tasks. The following rules are used by the VM/OS to decide on the sleep mode.

- **SLEEP:** the next task is scheduled far enough ahead (no other task which can generate a callback); all hardware modules (MCU, sensors, radios, etc.) and will be powered off.
- **IDLE:** no task which can potentially generate a callback, e.g., a receive task from a communication device such as the radio, Ethernet, or USB.
- **ACTIVE:** whenever VM callbacks and OS code execute.

For example, if there is only one task to be performed a few seconds ahead and no other task is scheduled in the meantime, the OS will try to put the system into the most efficient sleep mode. As another example, if a communication device such as the radio has queued a reception task.

5. Sensors

5.1. Internal Sensors

5.1.1. Temperature

The Waspmote RTC (DS3231SN from Maxim) has a built in internal temperature sensor which it uses to recalibrate itself. Waspmote can access the value of this sensor through the I2C bus.

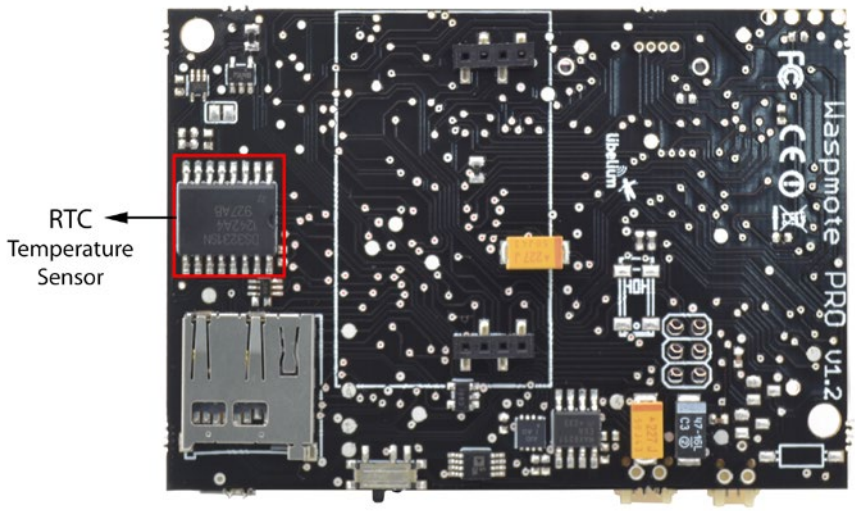


Figure 27: Temperature sensor in the RTC

The sensor is shown in a 10-bit two's complement format. It has a resolution of 0.25º C. The measurable temperature range is between -40ºC and +85ºC.
As previously specified, the sensor is prepared to measure the temperature of the board itself and can thereby compensate for oscillations in the quartz crystal it uses as a clock. As it is a sensor built in to the RTC, for any application that requires a probe temperature sensor, this must be integrated from the micro's analog and digital inputs, as has been done in the case of the sensor boards designed by Libelium.

See complete temperature documentation at section "RTC"

5.1.2. Accelerometer

Waspmote has a built in acceleration sensor LIS3331LDH STMicroelectronics which informs the mote of acceleration variations experienced on each one of the 3 axes (X,Y,Z).

The integration of this sensor allows the measurement of acceleration on the 3 axes (X,Y,Z), establishing 4 kind of events: Free Fall, inertial wake up, 6D movement and 6D position which are explained in the Interruptions Programming Guide.

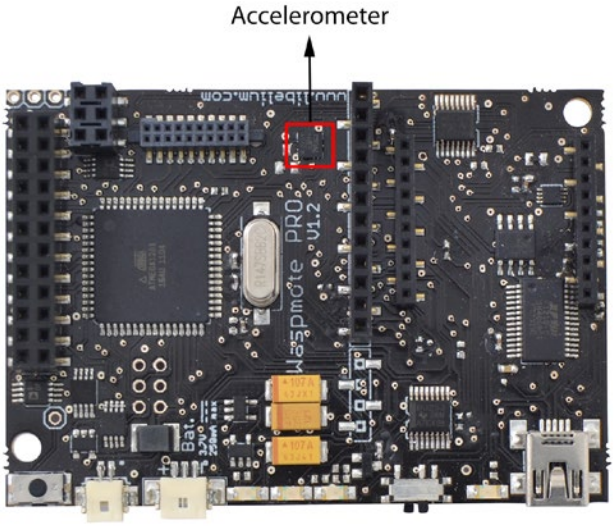


Figure 28: Accelerometer

The LIS331DLH has dynamically user selectable full scales of **±2g/±4g/±8g** and it is capable of measuring accelerations with output data rates from **0.5 Hz to 1 kHz**.

The device features ultra low-power operational modes that allow advanced power saving and smart sleep to wake-up functions.

The accelerometer has 7 power modes, the output data rate (ODR) will depend on the power mode selected. The power modes and output data rates are shown in this table:

Power mode	Output data rate (Hz)
Power down	--
Normal mode	1000
Low-power 1	0,5
Low-power 2	1
Low-power 3	2
Low-power 4	5
Low-power 5	10

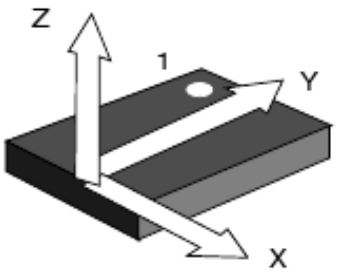


Figure 29: Axes in the LIS3LV02DL accelerometer

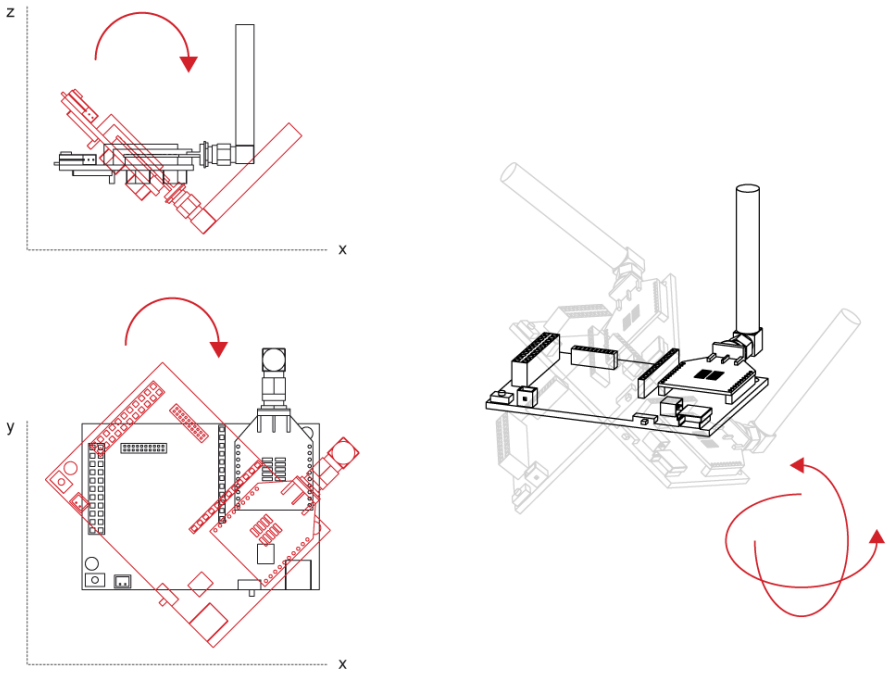
This accelerometer has an auto-test capability that allows the user to check the functioning of the sensor in the final application. Its operational temperature range is between -40ºC and +85ºC.

The accelerometer communicates with the microcontroller through the I2C interface. The pins that are used for this task are the SCL pin and the SDA pin, as well as another INT pin to generate the interruptions.

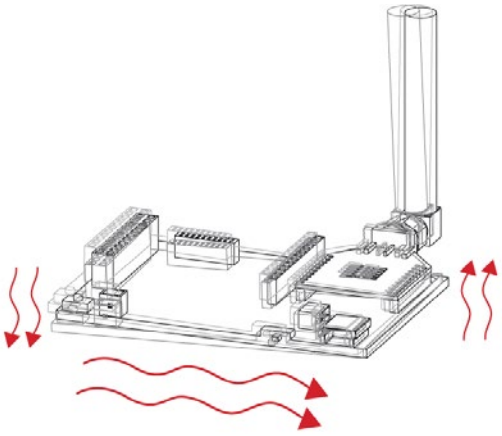
The accelerometer has 4 types of event which can generate an interrupt: free fall, inertial wake up, 6D movement and 6D position.

Some figures with possible uses of the accelerometer are shown below:

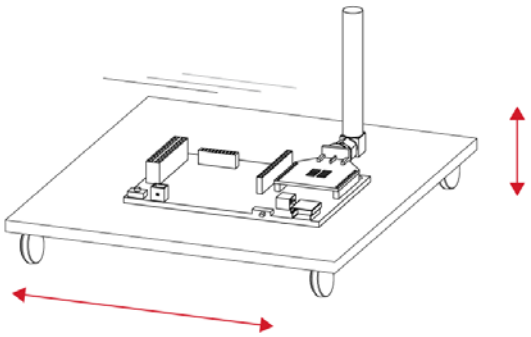
Rotation and Twist:



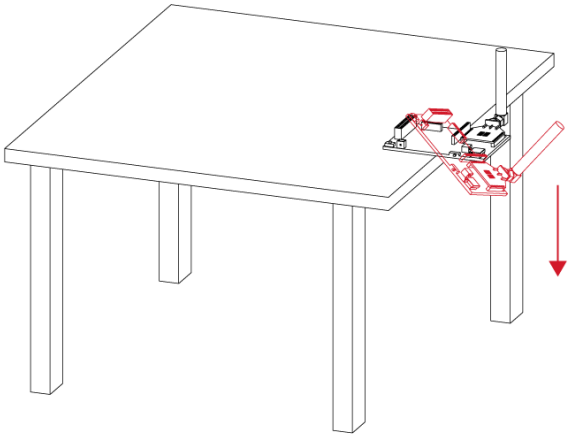
Vibration:



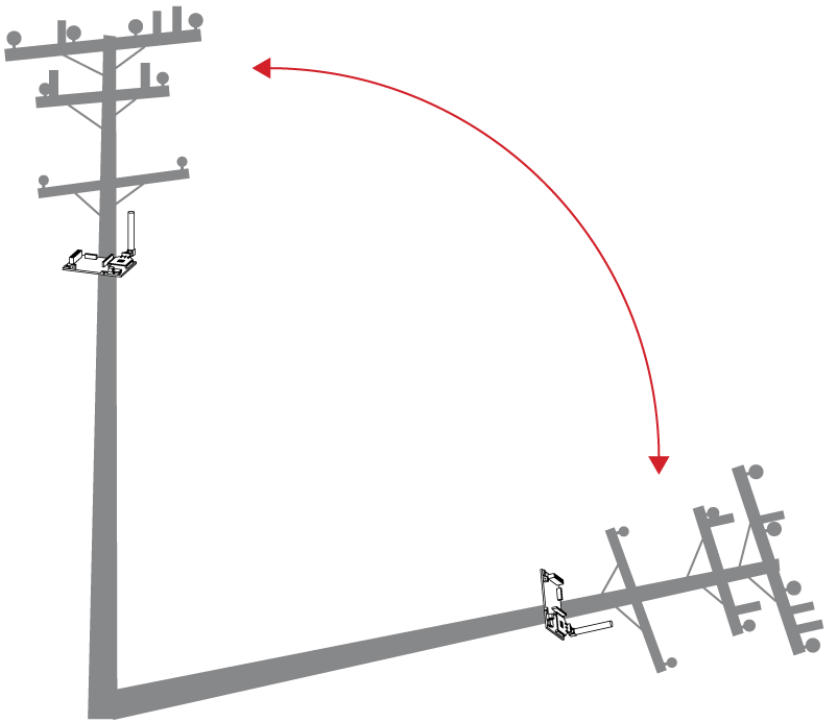
Acceleration:



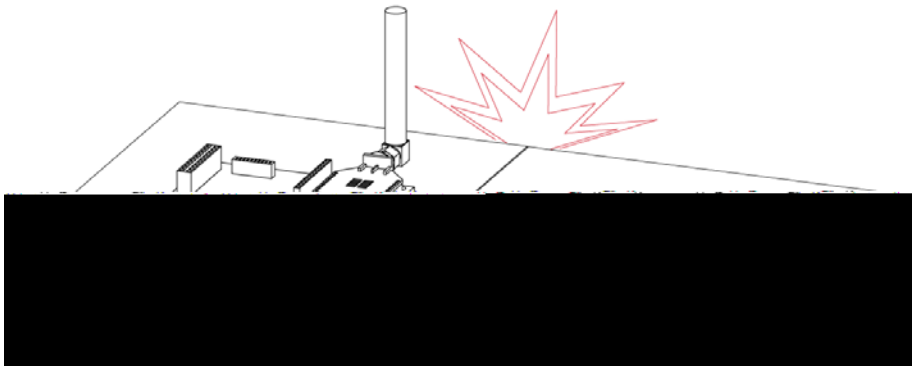
Free fall:



Free fall of objects in which it is installed:



Crash:



See complete I2C documentation at:
http://localhost:5000/gac/saguaro-system-11.4.htm?fqn=r:com.ibm.saguaro.system.I2C

Note that mrsh should be running in order to access the mentioned link

5.2. Sensor boards

5.2.1. Common library

The `com.libelium.common` library, which includes Callback.java, Common.java, Constants.java, Gpio.java and Interrupts.java is a library of methods and utilities used by the rest of the board libraries.

The Constants.java class includes the global constants of the API.

The Gpio.java class manages general purpose inputs and outputs in the API.

The Interrupts.java class handles interrupts, enabling, capturing and treating them.

The Common.java class contains methods that all the libraries of the API use.

The Callback.java is an instantiable object that will be responsible for containing the callback method that will be called when an asynchronous action has been finished.

To build the library, this set of commands must be executed in the Mote Runner shell (mrsh):

```
> cd /API path/com/libelium/common
> mrc --assembly=common-1.0 --system=waspmote Constants.java Common.java Gpio.java Inte-
rrupts.java Callback.java
```

To upload the compiled assembly to the Waspmote, runs as follows:

```
> cd/API path/com/libelium/common
> moma-load common
```

For further information about building and loading programs in mote runner, see the section "Application development".

This library must be referenced when building any of the other libraries and must be loaded in the waspmote for a proper working of the system.

5.2.2. Smart Cities Library

The `com.libelium.smartcities` is the responsible of managing the use of the sensor board called Smart Cities.

> ***Note:*** The necessary documentation to make the correct assembly of the sensors into the board and to know where must they be connected can be found in the technical guide of the board:
> **http://www.libelium.com/development/waspmote/documentation/smart-cities-board-technical-guide/**
>
> Please note that this guide is made for the original Waspmote system and the code that there appears will not work on Mote Runner.

For proper use of this library, the common assembly is needed to be loaded into the Waspmote. See "Common Library" section.

To build the library, this set of commands must be executed in the Mote Runner shell (mrsh):

```
>  cd /API path/com/libelium/smartcities
> mrc --assembly=smartcities-1.0 --system=waspmote --ref=../common/common-1.0 SmartCities.java
```

To upload the compiled assembly to the Waspmote, run the following commands:

```
> cd/API path/com/libelium/smartcities
> moma-load smartcities
```

Here are the necessary steps to read a value from a sensor:

1. Create the object of the SmartCities library class and switch on the same:

```
static smartCities sc = new SmartCities();
sc.ON();
```

2. Set the state of the sensor that is gonna be read:

```
sc.setSensorMode(byte mode, int sensor);
```

Where mode constants are:

```
- Constants.SENS_ON : turn on the sensor
- Constants.SENS_OFF : turn off the sensor
```

And sensor unique id constants are:

```
- Constants.SENS_CITIES_DUST : Dust Sensor
- Constants.SENS_CITIES_LD : Linear Displacement Sensor
- Constants.SENS_CITIES_AUDIO : Noise Sensor
- Constants.SENS_CITIES_HUMIDITY : Humidity Sensor
- Constants.SENS_CITIES_TEMPERATURE : Temperature Sensor
- Constants.SENS_CITIES_ULTRASOUND : Ultrasound Sensor
- Constants.SENS_CITIES_LDR : Luminosity Sensor
- Constants.SENS_CITIES_CD : Crack detection Sensor
- Constants.SENS_CITIES_CP : Crack propagation Sensor
```

3. Call the read method and store the value returned:

```
long result = sc.readValue(int sensor);
```

A simple complete example of the use of this library for reading the current sensor:

```
static SmartMetering sm = new SmartMetering();
sc.ON();
sc.setSensorMode(Constants.SENS_ON, Constants.SENS_CITIES_AUDIO);
long noise = sc.readValue(Constants.SENS_CITIES_AUDIO);

Logger.appendString(csr.s2b("Noise measurement: "));
Logger.appendHexLong(Float.toLong(noise, (byte)2));
Logger.flush(Mote.INFO);
```

This library also provides a system of interrupts that may be caused when a sensor detects that the value read is higher than an indicated threshold.

The steps to configure that threshold are the following:

1. Set the threshold value of the sensor:

```
sc.setThreshold(int sensor, long threshold);
```

2. Attach the interrupts and set the callback function that will be called when the interrupt is fired:

```
sc.attachInt(new Callback(null) {
  public int invoke(byte[] buf, int len) {
         return callbackInterrupt(buf, len);
  }
});
```

3. Then, in the callbackInterrupt method, discriminate the sensor that caused the interruption:

```
byte flags = sc.loadInt();
if((flags & <sensor constant>) > 0))
```

A simple complete example of the use of this library for setting a threshold in the noise sensor and capture the interruption caused:

```
static {
  sc.ON();
       sc.setSensorMode(Constants.SENS_ON, Constants.SENS_CITIES_AUDIO);
       sc.setThreshold(Constants.SENS_CITIES_AUDIO, Float.make(10, (byte)-1));

       sc.attachInt(new Callback(null) {
      public int invoke(byte[] buf, int len) {
           return callbackInterrupt(buf, len);
      }
       });
}

private static int callbackInterrupts(byte[] buf, int len){
       byte flags = sc.loadInt();

       if((flags & Constants.SENS_CITIES_AUDIO)>0){
         Logger.appendString(csr.s2b("Noise interruption"));
         Logger.flush(Mote.INFO);
  }
}
```

Further documentation of the methods can be found embedded within the code and in a generated javadoc delivered with the libraries.

See more examples at:

http://www.libelium.com/development/waspmote/examples/?cat=mr-sensor-boards&subcat=mr-cities

5.2.3. Smart Metering Library

The `com.libelium.smartmetering` is the responsible of managing the use of the sensor board called Smart Metering.

> ***Note:*** *The necessary documentation to make the correct assembly of the sensors into the board and to know where must they be connected can be found in the technical guide of the board:*
>
> **http://www.libelium.com/development/waspmote/documentation/smart-metering-board-technical-guide/**
>
> Please note that this guide is made for the original Waspmote system and the code that there appears will not work on Mote Runner.

For proper use of this library, the common assembly is needed to be loaded into the Waspmote. See "Common Library" section.

To build the library, this set of commands must be executed in the Mote Runner shell (mrsh):

```
>  cd /API path/com/libelium/smartmetering
> mrc --assembly=smartmetering-1.0 --system=waspmote --ref=../common/common-1.0 SmartMetering.java
```

To upload the compiled assembly to the Waspmote, run the following commands:

```
> cd/API path/com/libelium/smartmetering
> moma-load smartmetering
```

Here are the necessary steps to read a value from a sensor:

1. Create the object of the SmartMetering library class and switch on the same:

```
static SmartMetering sm = new SmartMetering();
sm.ON();
```

2. Set the state of the sensor that is gonna be read:

```
sm.setSensorMode(byte mode, int sensor);
```

Where mode constants are:

```
- Constants.SENS_ON : turn on the sensor
- Constants.SENS_OFF : turn off the sensor
```

And sensor unique id constants are:

```
- Constants.SENS_SMART_LDR : luminosity sensor
- Constants.SENS_SMART_FLOW_5V : flow sensor at 5V socket
- Constants.SENS_SMART_LCELLS_5V : load cell
- Constants.SENS_SMART_LCELLS_10V : load cell
- Constants.SENS_SMART_CURRENT : current clamp
- Constants.SENS_SMART_TEMPERATURE : temperature sensor
- Constants.SENS_SMART_HUMIDITY : humidity sensor
- Constants.SENS_SMART_US_3V3 : ultrasound sensor at 3.3V socket
- Constants.SENS_SMART_FLOW_3V3 : flow sensor at 3.3V socket
- Constants.SENS_SMART_US_5V : ultrasound sensor at 5V socket
- Constants.SENS_SMART_DFS_3V3 : foil sensor at 3.3V socket
- Constants.SENS_SMART_DFS_5V : foil sensor at 5V socket
```

3. Call the read method and store the value returned:

```
long result = sm.readValue(int sensor);
```

A simple complete example of the use of this library for reading the current sensor:

```
static SmartMetering sm = new SmartMetering();
sm.ON();
sm.setSensorMode(Constants.SENS_ON, Constants.SENS_SMART_CURRENT);
long current = sm.readValue(Constants.SENS_SMART_CURRENT);

Logger.appendString(csr.s2b("Current measurement: "));
Logger.appendHexLong(Float.toLong(current, (byte)2));
Logger.flush(Mote.INFO);
```

Further documentation of the methods can be found embedded within the code and in a generated javadoc delivered with the libraries.

See more examples at:

http://www.libelium.com/development/waspmote/examples/?cat=mr-sensor-boards&subcat=mr-smart

5.2.4. Agriculture Library

The `com.libelium.agriculture` is the responsible of managing the use of the sensor board called Agriculture.

Note: *The necessary documentation to make the correct assembly of the sensors into the board and to know where must they be connected can be found in the technical guide of the board:*

http://www.libelium.com/development/waspmote/documentation/agriculture-board-technical-guide/

Please note that this guide is made for the original Waspmote system and the code that there appears will not work on Mote Runner.

For proper use of this library, the common assembly is needed to be loaded into the Waspmote. See "Common Library" section. To build the library, this set of commands must be executed in the Mote Runner shell (mrsh):

To build the library, this set of commands must be executed in the Mote Runner shell (mrsh):

```
>  cd /API path/com/libelium/agriculture
> mrc --assembly=agriculture-1.0 --system=waspmote --ref=../common/common-1.0 Agriculture.java
```

To upload the compiled assembly to the Waspmote, run the following commands:

```
> cd/API path/com/libelium/agriculture
> moma-load agriculture
```

Here are the necessary steps to read a value from a sensor:

1. Create the object of the Agriculture library class and switch on the same:

```
static Agriculture agr = new Agriculture();
agr.ON();
```

2. Set the state of the sensor that is gonna be read:

```
agr.setSensorMode(byte mode, int sensor);
```

Where mode constants are:

```
- Constants.SENS_ON : turn on the sensor
- Constants.SENS_OFF : turn off the sensor
```

And sensor unique id constants are:

```
- Constants.SENS_AGR_PRESSURE : Atmospheric Pressure Sensor
- Constants.SENS_AGR_WATERMARK1 : Soil Moisture 1 Sensor
- Constants.SENS_AGR_WATERMARK2 : Soil Moisture 2 Sensor
- Constants.SENS_AGR_WATERMARK3: Soil Moisture 3 Sensor
- Constants.SENS_AGR_ANEMOMETER : Weather Station Sensor
- Constants.SENS_AGR_VANE : Weather Station Sensor
- Constants.SENS_AGR_DENDROMETER : Trunk diameter Sensor
- Constants.SENS_AGR_PT100 : Soil temperature Sensor
- Constants.SENS_AGR_LEAF_WETNESS : Leaf wetness Sensor
- Constants.SENS_AGR_TEMPERATURE : Temperature Sensor
- Constants.SENS_AGR_HUMIDITY : Humidity Sensor
- Constants.SENS_AGR_RADIATION : Solar radiation Sensor
- Constants.SENS_AGR_SENSIRION : Humidity and temperature Sensor
- Constants.SENS_AGR_LDR : Luminosity Sensor
```

3. Call the read method and store the value returned:

```
long result = agr.readValue(int sensor);
```

The sensors that can be read this way are:

```
- Constants.SENS_AGR_PRESSURE : Atmospheric Pressure Sensor
- Constants.SENS_AGR_WATERMARK1 : Soil Moisture 1 Sensor
- Constants.SENS_AGR_WATERMARK2 : Soil Moisture 2 Sensor
- Constants.SENS_AGR_WATERMARK3: Soil Moisture 3 Sensor
- Constants.SENS_AGR_ANEMOMETER : Weather Station Sensor
- Constants.SENS_AGR_VANE : Weather Station Sensor
- Constants.SENS_AGR_LEAF_WETNESS : Leaf wetness Sensor
- Constants.SENS_AGR_TEMPERATURE : Temperature Sensor
- Constants.SENS_AGR_HUMIDITY : Humidity Sensor
- Constants.SENS_AGR_LDR : Luminosity Sensor
```

A simple complete example of the use of this library for reading the temperature sensor:

```
static Agriculture agr = new Agriculture();
agr.ON();
agr.setSensorMode(Constants.SENS_ON, Constants.SENS_AGR_TEMPERATURE);
long temperature = agr.readValue(Constants.SENS_AGR_TEMPERATURE);

Logger.appendString(csr.s2b("Temperature measurement: "));
Logger.appendHexLong(Float.toLong(temperature, (byte)2));
Logger.flush(Mote.INFO);
```

To read the rest sensors there are unique methods that needs an special parameter that will be the invoked callback when the reading is done. This is because they are doing an asynchronous reading.

The callback method will receive a byte array containing the long result.

```
public void readPT1000(Callback cb);
public void readDendrometer(Callback cb);
public void readRadiation(Callback cb);

/*
 * Sensirion type:
 *     Constants.SENSIRION_TEMP
 *     Constants.SENSIRION_HUM
 */
public void readSensirion(byte type, Callback cb);
```

And an example of the use of this library for reading one of the special sensor:

```
static Agriculture agr = new Agriculture();
agr.ON();
agr.setSensorMode(Constants.SENS_ON, Constants.SENS_AGR_DENDROMETER);

agr.readDendrometer(new Callback(null) {
  public int invoke(byte[] buf, int len) {
         return PruebasAgricultura.callbackDendrometer(buf,len);
  }
});

private static int callbackDendrometer(byte[] buf, int len){

  Logger.appendString(csr.s2b("Medicion dendro: "));
  long result = Util.get32(buf, 0);
  Logger.appendHexLong(Float.toLong(result, (byte)2));
  Logger.flush(Mote.INFO);
  return 0;
}
```

This Library can also attach Pluviometer interrupts:

```
agr.attachPluvioInt();
```

And then, the value of the actual count of pluviometer interruptions can be read as:

```
int count = readPluviometerValue();
```

Further documentation of the methods can be found embedded within the code and in a generated javadoc delivered with the libraries.

See more examples at:

http://www.libelium.com/development/waspmote/examples/?cat=mr-sensor-boards&subcat=mr-agriculture

5.2.5. Events Library

The `com.libelium.events` is the responsible of managing the use of the sensor board called Smart Cities.

Note: *The necessary documentation to make the correct assembly of the sensors into the board and to know where must they be connected can be found in the technical guide of the board:*

http://www.libelium.com/development/waspmote/documentation/events-board-technical-guide/

Please note that this guide is made for the original Waspmote system and the code that there appears will not work on Mote Runner.

For proper use of this library, the common assembly is needed to be loaded into the Waspmote. See "Common Library" section.

To build the library, this set of commands must be executed in the Mote Runner shell (mrsh):

```
> cd /API path/com/libelium/events
> mrc --assembly=events-1.0 --system=waspmote --ref=../common/common-1.0 Events.java
```

To upload the compiled assembly to the Waspmote, run the following commands:

```
> cd/API path/com/libelium/events
> moma-load events
```

Here are the necessary steps to read a value from a sensor:

1. Create the object of the Events library class and switch on the same:

```
static Events events = new Events();
events.ON();
```

2. Call the read method and store the value returned:

```
long result = events.readValue(int sensor, byte type);
```

or (if it's none of the types described below)

```
long result = events.readValue(int sensor);
```

Where sensor unique id constants are:

```
- Constants.SENS_SOCKET1
- Constants.SENS_SOCKET2
- Constants.SENS_SOCKET3
- Constants.SENS_SOCKET4
- Constants.SENS_SOCKET5
- Constants.SENS_SOCKET6
- Constants.SENS_SOCKET7
- Constants.SENS_SOCKET8
```

And the types are:

```
- Constants.SENS_RESISTIVE
- Constants.SENS_FS100
- Constants.SENS_FS200A
- Constants.SENS_FS400
- Constants.SENS_TEMPERATURE
- Constants.SENS_HUMIDITY
```

A simple complete example of the use of this library for reading the temperature sensor:

```
static Events events = new Events();
sc.ON();
long temperature = events.readValue(Constants.SENS_SOCKET5,Constants.SENS_TEMPERATURE);

Logger.appendString(csr.s2b("Temperature measurement: "));
Logger.appendHexLong(Float.toLong(temperature, (byte)2));
Logger.flush(Mote.INFO);
```

This library also provides a system of interrupts that may be caused when a sensor detects that the value read is higher than an indicated threshold.

The steps to configure that threshold are the following:

1. Set the threshold value of the sensor:

```
events.setThreshold(Constants.SENS_SOCKET6, Float.make(20, (byte)-1));
```

2. Attach the interrupts and set the callback function that will be called when the interrupt is fired:

```
events.attachInt(new Callback(null) {
  public int invoke(byte[] buf, int len) {
         return callbackInt(buf, len);
  }
});
```

3. Then, in the `callbackInt` method, discriminate the sensor that caused the interruption:

```
byte flags = events.loadInt();
if((flags & <sensor constant>) > 0))
```

A simple complete example of the use of this library for setting a threshold in the socket1 sensor and capture the interruption caused:

```
static {
  events.ON();
       events.setThreshold(Constants.SENS_SOCKET1, Float.make(20, (byte)-1));

       sc.attachInt(new Callback(null) {
      public int invoke(byte[] buf, int len) {
           return callbackInterrupt(buf, len);
      }
       });
}

private static int callbackInterrupts(byte[] buf, int len){
       byte flags = sc.loadInt();

       if((flags & Constants.SENS_SOCKET1)>0){
          Logger.appendString(csr.s2b("Socket 1 INTERRUPT"));
          Logger.flush(Mote.INFO);
  }
}
```

Further documentation of the methods can be found embedded within the code and in a generated javadoc delivered with the libraries.

See more examples at:

http://www.libelium.com/development/waspmote/examples/?cat=mr-sensor-boards&subcat=mr-events

5.2.6. Gases Library

The `com.libelium.gases` is the responsible of managing the use of the sensor board called Smart Metering.

Note: *The necessary documentation to make the correct assembly of the sensors into the board and to know where must they be connected can be found in the technical guide of the board:*

http://www.libelium.com/development/waspmote/documentation/gases-board-technical-guide/

Please note that this guide is made for the original Waspmote system and the code that there appears will not work on Mote Runner.

For proper use of this library, the common assembly is needed to be loaded into the Waspmote. See "Common Library" section.

To build the library, this set of commands must be executed in the Mote Runner shell (mrsh):

```
>  cd /API path/com/libelium/gases
> mrc --assembly=gases-1.0 --system=waspmote --ref=../common/common-1.0 Gases.java
```

To upload the compiled assembly to the Waspmote, run the following commands:

```
> cd/API path/com/libelium/gases
> moma-load gases
```

Here are the necessary steps to read a value from a sensor:

1. Create the object of the Gases library class and switch on the same:

```
static Gases gas = new Gases();
sm.ON();
```

2. Set the state of the sensor that is gonna be read:

```
sm.setSensorMode(byte mode, int sensor);
```

Where mode constants are:

```
- Constants.SENS_ON : turn on the sensor
- Constants.SENS_OFF : turn off the sensor
```

And sensor unique id constants are:

```
- SENS_PRESSURE : Atmospheric Pressure Sensor
- SENS_CO2 : Carbon Dioxide Sensor
- SENS_O2 : Oxygen Sensor
- SENS_SOCKET2A : Sensor placed on SOCKET2A
- SENS_SOCKET2B : Sensor placed on SOCKET2B
- SENS_SOCKET3A : Sensor placed on SOCKET3A
- SENS_SOCKET3B : Sensor placed on SOCKET3B
- SENS_SOCKET4A : Sensor placed on SOCKET4
```

3. Configure the sensor:

```
public byte configureSensor(int sensor, byte gain, long resistor)
```

4. Call the read method and store the value returned:

```
long result = gas.readValue(int sensor);
```

A simple complete example of the use of this library for reading the current sensor:

```
static Gases gas = new Gases();
gas.ON();
gas.setSensorMode(Constants.SENS_ON, Constants.SENS_SOCKET4CO);
gas.configureSensor(Constants.SENS_SOCKET4CO, (byte)2, Float.make(100, (byte)0));

long socket4CO = gas.readValue(Constants.SENS_SOCKET4CO);

Logger.appendString(csr.s2b("4CO measurement: "));
Logger.appendHexLong(Float.toLong(socket4CO, (byte)2));
Logger.flush(Mote.INFO);
```

Further documentation of the methods can be found embedded within the code and in a generated javadoc delivered with the libraries.

See more examples at:

http://www.libelium.com/development/waspmote/examples/?cat=mr-sensor-boards&subcat=mr-gases

5.2.7. Parking Library

The `com.libelium.parking` is the responsible of managing the use of the sensor board called Smart Metering.

> ***Note:*** *The necessary documentation to make the correct assembly of the sensors into the board and to know where must they be connected can be found in the technical guide of the board:*
>
> **http://www.libelium.com/development/waspmote/documentation/smart-parking-board-technical-guide/**
>
> *Please note that this guide is made for the original Waspmote system and the code that there appears will not work on Mote Runner.*

For proper use of this library, the common assembly is needed to be loaded into the Waspmote. See "Common Library" section.

To build the library, this set of commands must be executed in the Mote Runner shell (mrsh):

```
>  cd /API path/com/libelium/parking
> mrc --assembly=parking-1.0 --system=waspmote --ref=../common/common-1.0 Parking.java
```

To upload the compiled assembly to the Waspmote, run the following commands:

```
> cd/API path/com/libelium/parking
> moma-load parking
```

A complete example of reading the state of a parking place using this library:

```
static{
      long coefX2 = Float.make(-1250745, (byte) -11);
  long coefX = Float.make(-1955917, (byte) -9);
  long constX = Float.make(1043527, (byte) -6);

  long coefY2 = Float.make(1434863, (byte) -12);
  long coefY = Float.make(-3030432, (byte) -10);
  long constY = Float.make(1006304, (byte) -6);

  long coefZ2 = Float.make(-7289752, (byte) -12);
  long coefZ = Float.make(6671079, (byte) -11);
  long constZ = Float.make(1001546, (byte) -6);

            park.loadReference(coefX2, coefX, constX, coefY2, coefY, constY
      coefZ2, coefZ, constZ);

            park.readParkingSetReset();
  int temperature = park.readTemperature();

  park.calculateReference(temperature);

  boolean status = park.estimateState();

     if(status==Constants.PARKING_OCCUPIED){
        Logger.appendString(csr.s2b(" OCCUPIED"));
        Logger.flush(Mote.INFO);
  }else{
        Logger.appendString(csr.s2b(" EMPTY"));
        Logger.flush(Mote.INFO);
  }
}
```

Further documentation of the methods can be found embedded within the code and in a generated javadoc delivered with the libraries.

See more examples at:

http://www.libelium.com/development/waspmote/examples/?cat=mr-sensor-boards&subcat=mr-parking

5.2.8. Radiation Library

The `com.libelium.radiation` is the responsible of managing the use of the sensor board called Smart Metering.

Note: *The necessary documentation to make the correct assembly of the sensors into the board and to know where must they be connected can be found in the technical guide of the board:*

http://www.libelium.com/development/waspmote/documentation/radiation-board-technical-guide/

Please note that this guide is made for the original Waspmote system and the code that there appears will not work on Mote Runner.

For proper use of this library, the common assembly is needed to be loaded into the Waspmote. See "Common Library" section.

To build the library, this set of commands must be executed in the Mote Runner shell (mrsh):

```
>  cd /API path/com/libelium/radiation
> mrc --assembly=radiation-1.0 --system=waspmote --ref=../common/common-1.0 Radiation.java
```

To upload the compiled assembly to the Waspmote, run the following commands:

```
> cd/API path/com/libelium/radiation
> moma-load radiation
```

This library offers a method to obtain the count of pulses of radiation in an interval.

The callback is a method that will be called when the <timeMillisRadiation> time has elapsed:

```
getRadiation(byte mode, Callback cb, long timeMillisRadiation)
```

A complete example of the use of this library to get the radiation count in ten seconds:

```
static{

  rad.ON();

  rad.getRadiation((byte)0, new Callback(null) {
        public int invoke(byte[] buf, int len) {
              return interrupt(buf, len);
        }
  }, 10000);
}
static int interrupt(byte[] buf, int len) {

  long result = Util.get32(buf, 0);
  Logger.appendString(csr.s2b("Radiacion en usv/h: "));
  Logger.appendHexLong(Float.toLong(result,(byte)2));
  Logger.flush(Mote.INFO);
}
```

Further documentation of the methods can be found embedded within the code and in a generated javadoc delivered with the libraries.

See more examples at:

http://www.libelium.com/development/waspmote/examples/?cat=mr-sensor-boards&subcat=mr-radiation

5.2.9. Prototyping Library

The `com.libelium.prototyping` is the responsible of managing the use of the prototyping sensor board.

Note: *The necessary documentation to make the correct assembly of the sensors into the board and to know where must they be connected can be found in the technical guide of the board:*

http://www.libelium.com/development/waspmote/documentation/prototyping-board-technical-guide/

Please note that this guide is made for the original Waspmote system and the code that there appears will not work on Mote Runner.

For proper use of this library, the common assembly is needed to be loaded into the Waspmote. See "Common Library" section.

To build the library, this set of commands must be executed in the Mote Runner shell (mrsh):

```
> cd /API path/com/libelium/prototyping
> mrc --assembly=prototyping-1.0 --system=waspmote --ref=../common/common-1.0 Prototyping.java
```

To upload the compiled assembly to the Waspmote, run the following commands:

```
> cd/API path/com/libelium/prototyping
> moma-load prototyping
```

Here are the necessary steps to read a value from a sensor:

1. Create the object of the Prototyping library class and switch on the same:

```
static Prototyping proto = new Prototyping();
proto.ON();
```

2. Call the read method and store the value returned:

```
long result = proto.readAnalogSensor(byte pin);
```

To read a value from the ADC that the board has attached:

1. Create the object of the Prototyping library class and switch on the same:

```
static Prototyping proto = new Prototyping();
proto.ON();
```

2. Call the reading method providing the callback that should be used when the read is done:

```
readADC(new Callback(null){
  public int invoke(byte[] buf, int len) {
     return <Class>.callback(buf,len);
  }
});
```

A simple complete example of the use of this library for reading a sensor:

```
static Prototyping proto = new Prototyping();
proto.ON();
long valRead  = proto.readAnalogSensor(byte pin);

Logger.appendString(csr.s2b("Sensor measurement: "));
Logger.appendHexLong(Float.toLong(valRead, (byte)2));
Logger.flush(Mote.INFO);
```

Further documentation of the methods can be found embedded within the code and in a generated javadoc delivered with the libraries.

See more examples at:

http://www.libelium.com/development/waspmote/examples/?cat=mr-sensor-boards&subcat=mr-prototyping

6. Mote Runner SDK

The latest Mote Runner SDK for Windows, Linux (and Linux 64-bit) and Mac iOS can be downloaded from:
http://www.zurich.ibm.com/moterunner

6.1. SDK Installation

6.1.1. Requirements

6.1.1.1. Firefox

The Firefox browser is required to operate the IBM Mote Runner Web GUI and for reading the documentation. These features will not work correctly under Internet Explorer or other browsers. You should have at least version 23.x installed. You can download Firefox **here**.

6.1.1.2. Java JRE/SDK

You need a Java JRE to operate the IBM Mote Runner tool chain. If you are planning to write IBM Mote Runner applications in Java then you need a Java JDK containing a Java compiler. You should have installed a Java JRE/JDK version 6 or higher (download **here**). If in doubt install the JDK.

Make sure the tools directory of the JDK/JRE is added to the **PATH** environment variable so that the IBM Mote Runner compiler can find the **java** and **javac** programs.

6.1.1.3. Optional: C# Language Compiler

You only need a C# compiler if you plan to develop IBM Mote Runner applications in the C# programming language. Mono is available **here**.

6.1.2. Windows

6.1.2.1. Quick Start

- Download and unzip the corresponding moterunner-*.win32.zip file
- Execute the setup.hta file and follow the instructions
- Set-up the PATH environment variable

6.1.2.2. PATH Environment Variable

The PATH environment variable should contain all the directories holding tools relevant for IBM Mote Runner. In addition, you should add the IBM Mote Runner tools directory `<YOUR_INSTALLATION_DIRECTORY>/moterunner/win32/bin` to the PATH.

For example assuming you have installed IBM Mote Runner under C:\moterunner:

```
PATH=C:\moterunner\win32\bin;%PATH%
```

You can double click on the file mrcmd.bat in the moterunner directory to get a Windows command shell with the path set.

6.1.2.3. Optional: C# development

Nowadays, all Windows operating systems have the .NET framework installed by default. Any version (v1, v2, or v3) will do. You can explicitly select the version with an option for the IBM Mote Runner compiler. Verify that the framework is installed in the default place (under C:/WINDOWS/Microsoft.NET/Framework). If it is not in this location make sure that the directory is listed in the PATH environment variable so that the **csc.exe** and other tools can be found by the IBM Mote Runner compiler. You can also use Mono on Windows (see above).

6.1.2.4. Optional: Cygwin

Furthermore, you might consider installing Cygwin. Although not mandatory, it might make running some sample code out of the box easier. Cygwin emulates a Unix like environment on Windows. Some examples and sample code use small makefiles and tool usage throughout the documentation assumes a Unix like tool chain (GNU make, bash shell etc.).

Sample code and makefiles are kept very simple and can be easily changed into your preferred tool chain with little effort. If desired, Cygwin can be downloaded here.

6.1.3. Linux / Linux (64-bit)

6.1.3.1. Quick Start

- Download and unzip the corresponding moterunner-*.linux*.zip file
- Execute the setup script and follow the instructions
- Move the moterunner directory to a location of your own choice, e.g. ~/moteruner
- Set-up the PATH and LD_LIBRARY_PATH environment variables

6.1.3.2. PATH and LD_LIBRARY_PATH Environment Variable

The PATH environment variable should contain all the directories holding tools relevant for IBM Mote Runner. In addition, you should add the IBM Mote Runner tools to the PATH environment variable.

For example assuming you have installed IBM Mote Runner under ~/moterunner you can execute the following commands in your shell:

Linux (32-bit)

```
export PATH=~/moterunner/linux/bin:$PATH
export LD_LIBRARY_PATH=~/moterunner/linux/bin:$PATH
```

Linux (64-bit)

```
export PATH=~/moterunner/linux64/bin:$PATH
export LD_LIBRARY_PATH=~/moterunner/linux64/bin:$PATH
```

6.1.3.3. Optional: C# development

Mono is available at **Mono Project**.

6.1.4. MAC OSX

6.1.4.1. Quick Start

Download and unzip the corresponding moterunner-*.macosx*.zip file

Install the dmg file and follow the onscreen instructions

Set-up the PATH and LD_LIBRARY_PATH environment variables

6.1.4.2. PATH Environment Variable

The PATH environment variable should contain all the directories holding tools relevant for IBM Mote Runner. In addition, you should add the IBM Mote Runner tools directory to the PATH environment variable.

For example, assuming you have installed IBM Mote Runner under ~/moterunner:

```
export PATH="~/moterunner/macosx/bin:$PATH"
```

6.1.4.3. Optional: C# development

Mono is available **here**.

6.1.5. Optional: Eclipse Integration

The Mote Runner compiler tool can be integrated with the Eclipse SDK for Java.

> See complete Eclipse integration at:
> **http://localhost:5000/doc/system/index.html#tools_ideIntegration_0**
>
> Note that mrsh should be running in order to access the mentioned link

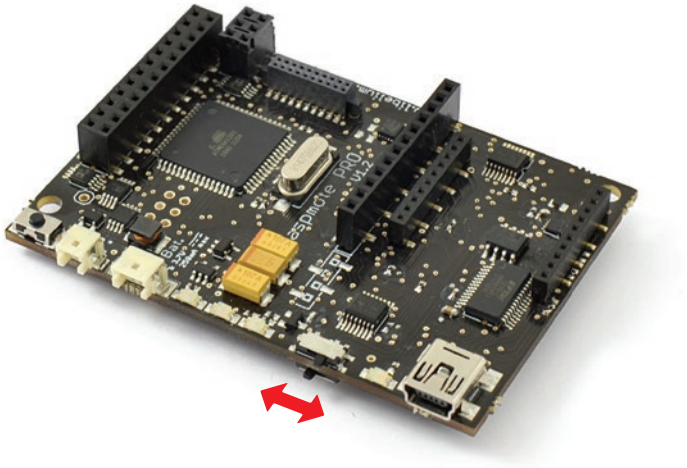
6.2. Firmware Installation for Waspmote

In order to use Mote Runner with the Waspmote the Mote Runner firmware must be installed on the Waspmote hardware. The following hardware and software are required:

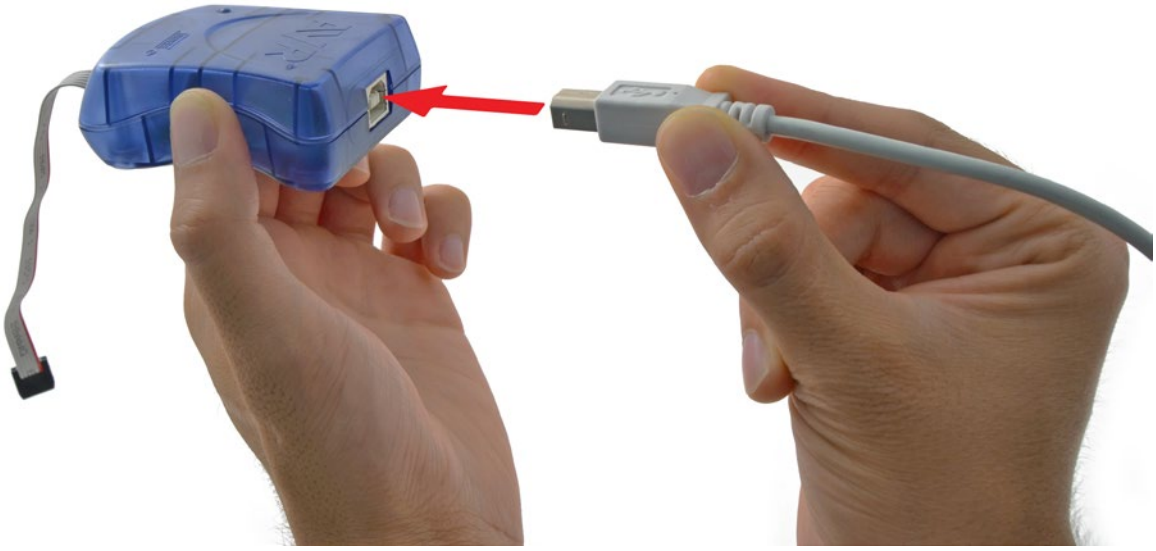
- A Waspmote PRO from Libelium with: a mini USB cable, an extension RF23x radio board, an extension Ethernet board
- An ISP (or JTAG) programmer for AVR from Atmel
- Flash writing tools like the Atmel Studio from Atmel or avrdude
- The Mote Runner firmware `<mote_runner_directory>/firmware/waspmote.hex`

6.2.1. Flashing the hex image

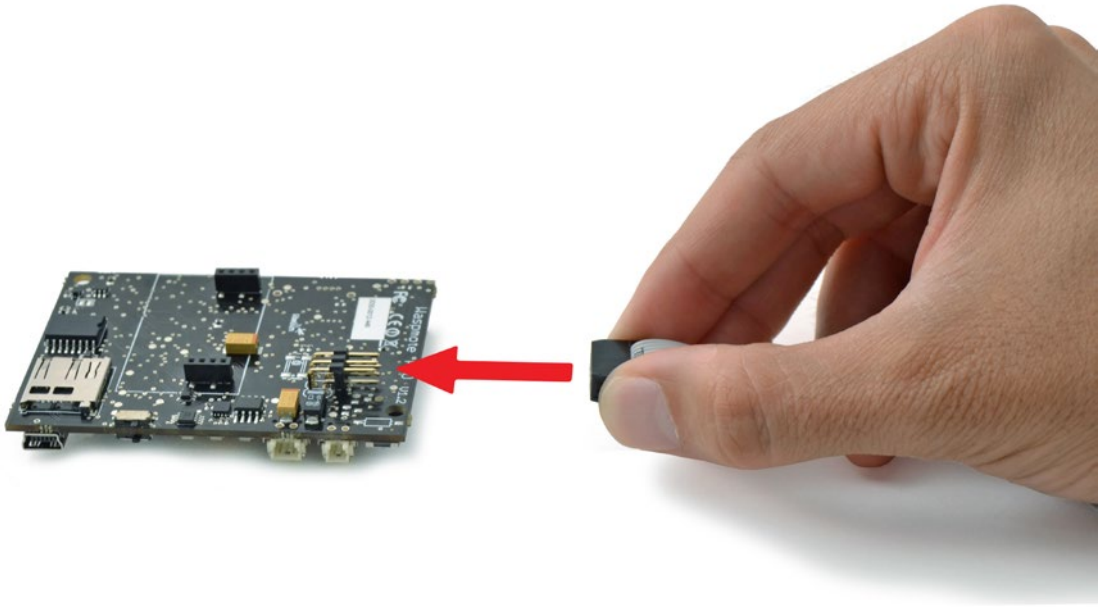
1. First of all connect the Waspmote to a power supply (for example to a computer) using the micro-USB cable. The Waspmote should be powered on (on the image move the switch to the left).



2. The next step is to connect the AVR programmer to the PC using the USB cable.



3. Now the AVR programmer can be connected to the ICSP connector on the back of the Waspmote.



4. The last step is to load the firmware on the waspmote using avrdude or Atmel Studio as explained in the following sections.

6.2.1.1. Loading the firmware using avrdude

Just type on a terminal the following commands.

Set the fuses:

```
sudo avrdude -c avrispmkII -p m1281 -P usb -b 115200 -e -u -U efuse:w:0xFF:m -U
hfuse:w:0xD0:m -U lfuse:w:0xFF:m
```

Load the firmware image:

```
sudo avrdude -c avrispmkII -p m1281 -P usb -b 115200 -U flash:w:waspmote.hex
```

6.2.1.2. Loading the firmware using Atmel Studio

Start the Atmel Studio and connect to the programmer. To check that the programmer and the SPI interface is operating correctly, select the ATmega1281 as the device and read the voltage and signature.

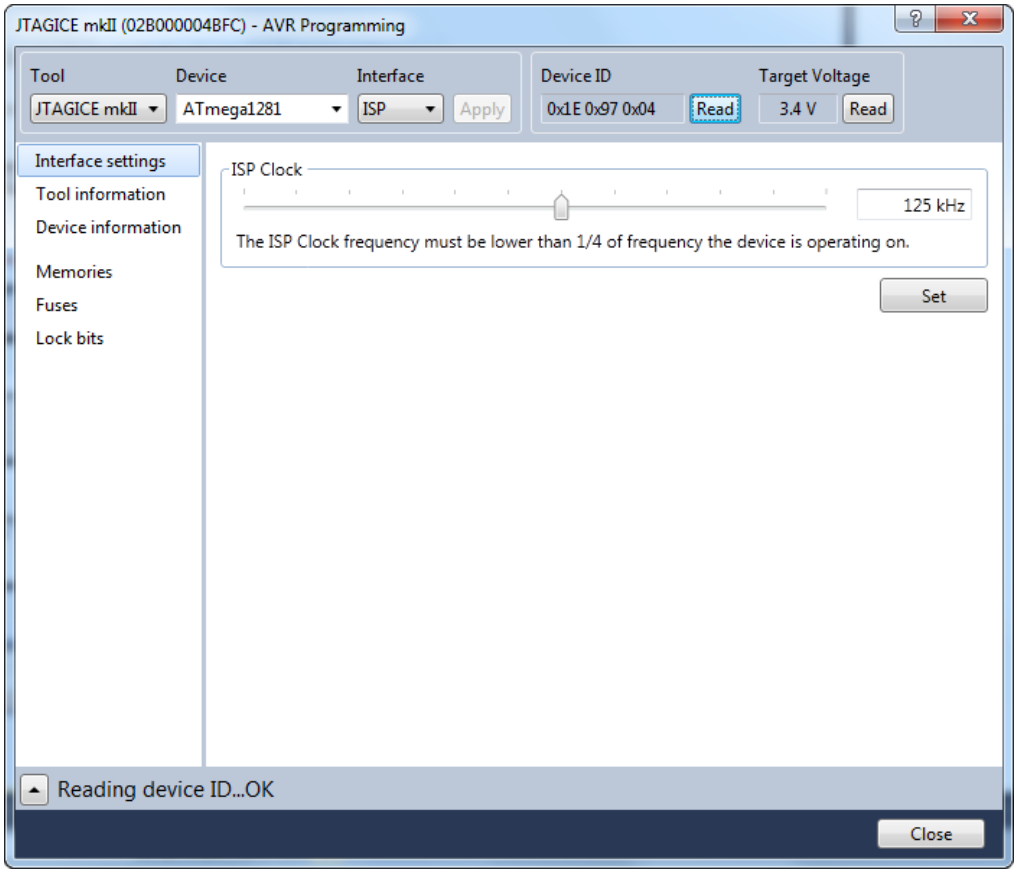


Figure 30: Atmel Studio AVR Programming Interface settings

Now, erase the entire chip. Set the fusses as described below. And finally, flash the Mote Runner firmware hex image.

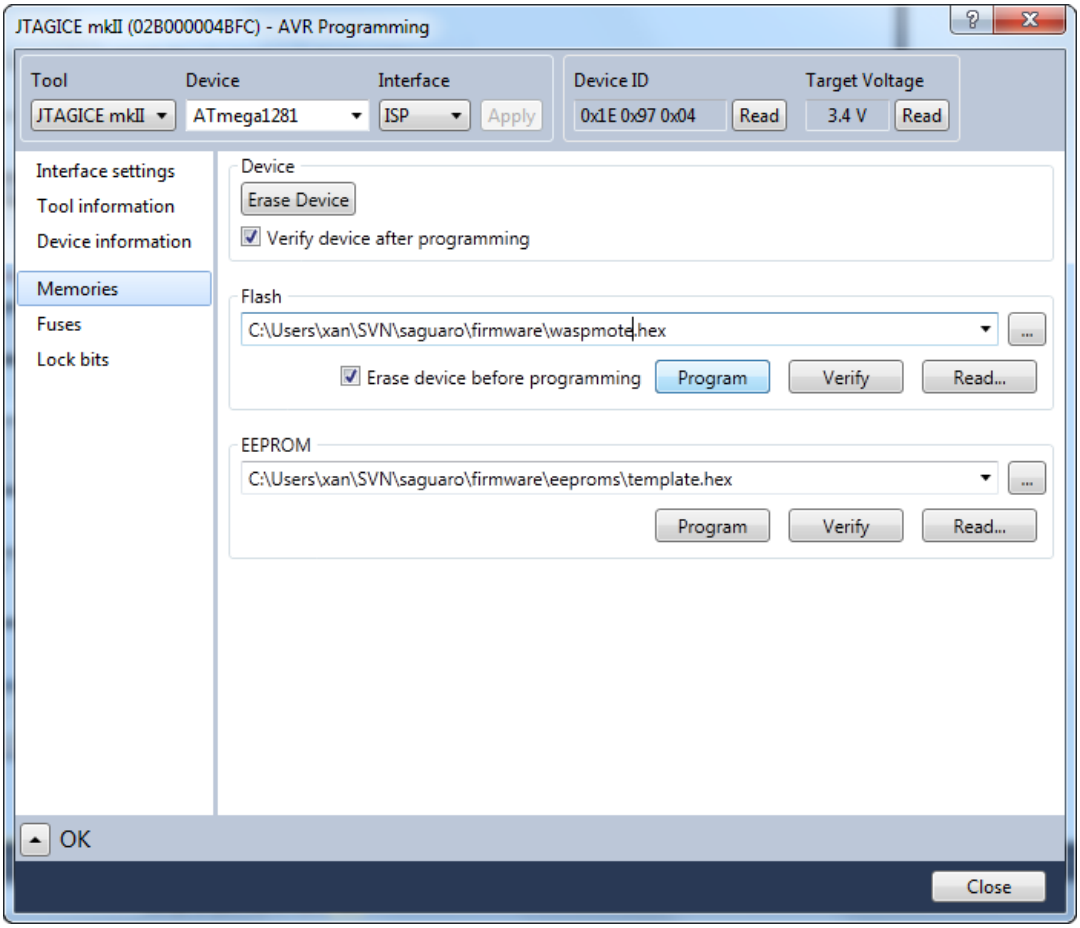


Figure 31: Atmel Studio AVR Programming Memories

6.3. Application development

Mote Runner applications can be developed using Java or C#.

Once an application has been written it must be compiled using the Mote Runner Compiler (**mrc**).

This tool can be found on the /bin folder inside the Mote Runner folder. Is useful to have this and other Mote Runner tools in the PATH environment variable.

The compiler generates an *assembly*. An assembly is bytecode that the VM inside the mote can execute.

6.3.1. Compiling applications

The basic Mote Runner compiler invocation is a follows:

```
mrc --assembly=<name>-<major>.<minor> --system=waspmote <code_file>
```

For example an application code on the file `TestADC.java` can be compiled as follows:

```
mrc --assembly=testadc-1.0 --system=waspmote TestADC.java
```

The compilation process will create two files: binary assembly (sba) and exchange file (sxp). The sxp file describes the public API of the assembly and it is needed in order to allow other assemblies to call its methods.

Public methods in Java applications and non private in case of C# will be visible by other assemblies on the mote and could be called by them.

In order to compile an application that calls other assembly methods the sxp file should be referenced:

```
mrc --assembly=<name>-<major>.<minor> --system=waspmote -r:<ref_sba>-<major>.<minor> <code_file>
```

For example if we want to reference the assembly `temperature-1.0` from the application `TestADC.java` we should do:

```
mrc --assembly=testadc-1.0 --system=waspmote -r:temperature-1.0 TestADC.java
```

6.3.2. Major.Minor Version Numbers

To facilitate application development and evolving APIs, the Mote Runner tool chain keeps manages binary compatibility of evolving assemblies. Java and C# runtime systems use strings to describe the signature of a method in detail. Changing the signature of a method will break linking. A Mote Runner platform uses small integer values for linking to save space. These integer values have a well defined meaning in the context of a specific assembly major version. The compiler manages the assignment if it identifies links across minor versions. If a programmer changes the public API of an assembly the compiler will ensure compatibility to the current or last minor version.

> See complete Mote Runner compiler documentation at:
> **http://localhost:5000/doc/system/index.html#tools_mrc**
>
> Note that mrsh should be running in order to access the mentioned link

6.3.3. Uploading applications

Communication between waspmotes (real ones or simulated) and the computer is established through Sonoran.

We can create a sonoran process running the Mote Runner Shell (mrsh).

6.3.3.1. Uploading assemblies to simulated motes

Inside mrsh:

```
# We start a saguaro process. We could specify a host and a port with saguaro-start -h lo-
calhost:44044 for example.

        saguaro-start

# Now we get connected to the saguaro process. If there is more than one saguaro process
we can specify to which one we want to connect by preceding the command with the saguaro-
process ID.

        saguaro-connect

# We create a mote on the process we are connected. With -d option we specify the platform

        mote-create -d waspmote

# An assembly can be loaded on this simulated mote. For examplo we load the assembly test-
1.0

        moma-load test-1.0

# For removing that assembly from the mote.

        moma-delete test-1.0
```

6.3.3.2. Uploading assemblies to real motes

Establishing connection with the mote through USB cable (Inside mrsh):

1. If we know the port to which it is connected (e.g. /dev/ttyUSB0):

```
mote-create -p /dev/ttyUSB0
```

2. If we don't know it:

```
#With the waspmote unplugged:
  lip-enumerate -w –waspmote -t30s

  #Now we plug the waspmote to the computer.
  #We can see a list of mrsh environment variables available with
     the list-vars command.
         list-vars
  #One of the variables should be WASPMOTE_MR_PORT that
     references the port to which the waspmote is connected.
  #In order to establish the connection we run:
  mote-create -p${WASPMOTE_MR_PORT}
```

Once a connection between the computer and the waspmote is established we can upload and remove assemblies.

```
# Loading the assembly test-1.0 on the mote.

moma-load test-1.0

# Removing test-1.0 from the mote.

moma-delete test-1.0
```

See complete assembly life cycle documentation at:
http://localhost:5000/doc/system/index.html#system_rtenv_1

Note that mrsh should be running in order to access the mentioned link

6.3.4. Debugging applications

When using the Mote Runner simulation applications can be debugged using an Eclipse plug in as shown in the image below.

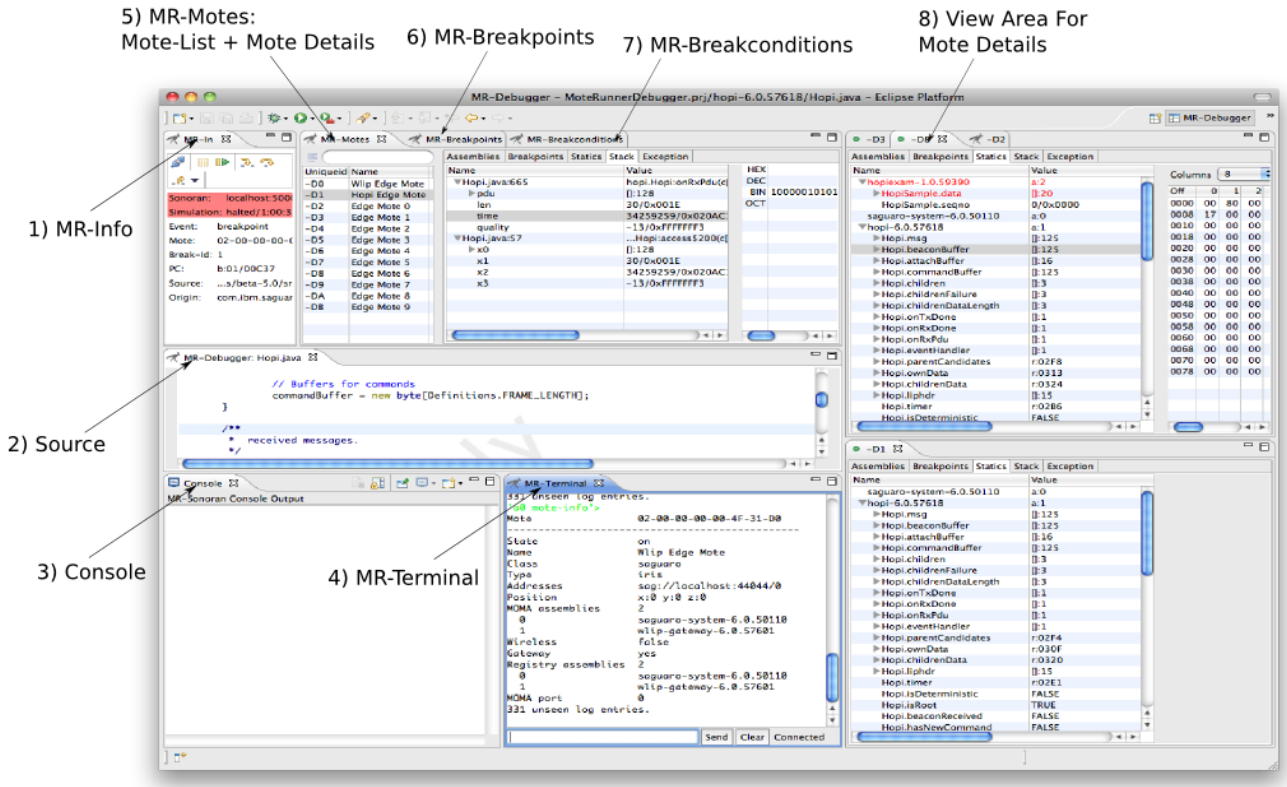


Figure 32: Mote Runner Debugger plugin for Eclipse

See details about installing the plug-in here:
http://localhost:5000/doc/system/index.html#start_startFirstSteps_PluginInstallation

See details about using the plug-in here:
http://localhost:5000/doc/system/index.html#tools_mrdebugger_0

Note that mrsh should be running in order to access the mentioned links

6.3.5. Simulating applications

The Mote Runner SDK comes with a rich simulation environment to facilitate application development and debugging. The environment accurately simulates the memory access, radio communication, sensor interaction, LED activity, ADC readings, GPIO operations, custom I2C sensors, position, and power consumption.

In general, it is good practice to first develop applications using the simulation environment before moving to actual hardware.

The mrsh commands provided by the moma-,mote-, device-, and feed- groups allows controlling most of the properties of simulated motes. For advanced control of I2C devices scripting using the Sonoran Java script environment is available.

To create a simulated Waspmote execute the following command in the **mrsh**:

```
> mote-create -dll waspmote
```

To list all available devices for the simulated mote can be listed:

```
> mote-info –devices
```

To turn on all LEDs of a Waspmote once can use the following command:

```
> moma-leds 3
```

To feed the simulation with data for the GPIO and ADC operations the feed-start and feed-stop commands can be used.

6.3.5.1. Timeline

All events (e.g., LED state changes, Radio messages) which occur in the simulation can be visualized using a Web-based Timeline (**http:localhost:5000/timeline**) as shown in the image below.

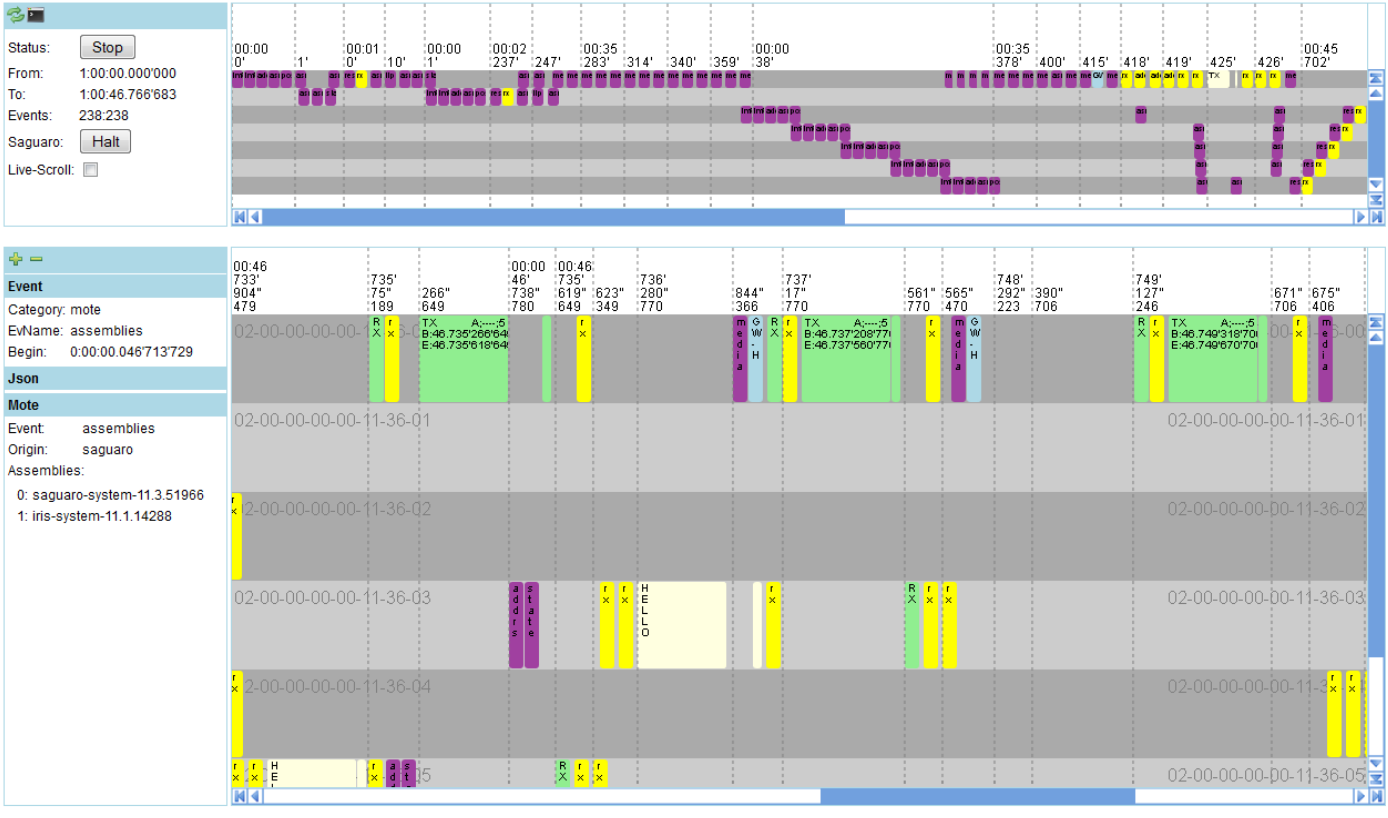


Figure 33: The web tool Timeline for Mote Runner

6.3.6. Managing Applications MOMA

The MOMA (Mote Manager) protocol specifies a number of LIP messages to configure a Mote Runner system. Any wireless or wired network protocol implementation can tunnel LIP messages to the system to initiate these MOMA management functions. The following 'mrsh' commands are wrappers to a LIP based communication using 'mrsh' sockets.

- moma-wlip: configure the persistent settings for a wireless WLIP mote such as channel.
- moma-list: list assemblies on a mote.
- moma-load: load an assembly.
- moma-delete: delete an assembly on a mote.
- moma-factory-reset: reset the mote and bring it back to factory state, e.g. deleting all installed applications
- moma-leds: switch on/off LEDs on motes.

Use 'help moma' in 'mrsh' to see the list of moma commands and 'help moma-xxx' for a command specific help message.

6.4. Libraries

The standard libraries available on the Mote Runner platform are available in the global assembly cache (GAC) folder, which can be found in ~/moterunner/gac. The gac directory contains the following

For each assembly version (name-major.minor the GAC holds the following types of files:

- a set of sba files: compiled binary versions of the assembly
- a set of sdb files: debugging information for each sba files (optional)
- sxp file: the API definition of the assembly
- jar file: the Java API of the assembly (optional)
- dll file: the C# API of the assembly (optional)

Note: *to use any libraries with Eclipse and code completion, the corresponding .jar file would need to be added to your Java project built path.*

See complete GAC documentation at:
http://localhost:5000/doc/system/index.html#tools_mrgac_0

Note that mrsh should be running in order to access the mentioned link

6.4.1. Saguaro System

Mote Runner system library which contains all hardware independent APIs such as for example the LIP, ADC, I2C, and Radio class as well as corresponding callbacks.

Note that the saguaro-system library is pre-loaded onto all simulated and hardware motes and cannot be deleted.

See Saguaro API documentation at:
http://localhost:5000/gac/saguaro-system-11.4.htm?fqn=com.ibm.saguaro.system

Note that mrsh should be running in order to access the mentioned link

6.4.2. Waspmote System

The waspmote-system library contains specific constants for pins, ADC settings, and device identifiers for the Waspmote hardware. The library is pre-loaded onto all Waspmote simulated and hardware motes and cannot be deleted.

To use the waspmote-system library simply use the import statement in Java (or the using statement in C#).

Java: import com.ibm.waspmote.*;

C#: using com.ibm.waspmote;

In addition, when compiling the application you need to specify the reference to the library. An example compiler invocation is as follows

```
mrc –ref:waspmote-system-#.# example.java
```

See Waspmote System API documentation at:
http://localhost:5000/gac/waspmote-system-13.1.htm

Note that mrsh should be running in order to access the mentioned link

6.4.3. Logger

Mote Runner provides a simple logging facility for applications attached via LIP to a host or via WLIP to a WLIP gateway. This facility is conveniently exported to user applications by the 'logger' assembly. A user application can link agains the logger assembly and use various static methods to create log messages and its 'flush' method to send the log messages to the host. The log messages are transported over WLIP and LIP, are picked up by 'mrsh' and appended to the shell log or dumped on the console.

Code using the Logger may look like this:

```
Logger.appendString(csr.s2b("value = "));
Logger.appendInt((int) 0);
Logger.flush(Mote.DEBUG);
```

This appends a string to the log buffer, appends a '0' and sends the log buffer contents to the host. The severity of the log message is 'DEBUG" (instead of INFO, WARN or ERROR). 'csr.s2b' is a helper method which is already executed when the code is compiled to a Mote Runner assembly: it maps the specified String to a byte array and adds that to the destination assembly.

The 'log-conf' command in the 'mrsh' allows to specify the logging behavior dependent on the category and severity of a message. Logging messages from motes appear under the cateogy 'MAPP'. To get all log messages from a mote starting with severity DEBUG immediately debugged on the console, the following 'mrsh' command may be used:

```
log-conf -i MAPP DEBUG
```

Log messages are kept by 'mrsh' in an internal buffer, the following prints the last thirty messages received by 'mrsh':

```
log-show -c 30
```

Note that log messages are discarded by the system when they are issued on a wireless mote running a custom network protocol.

6.4.4. WLIP Gateway

WLIP is a simple single channel star based wireless built-in network protocol. When a wireless mote starts it it sends out 'HELLO' messages listening for an acknowledgement by a gateway. If no WLIP gateway answers, an application can take over the radio and start a custom network protocol. If a WLIP gateway picks up the 'HELLO' it records the address of the mote and uses that for the future identification of and communication with the mote. Once connected, LIP messages can be exchanged between WLIP gateway and mote and especially the 'moma' commands in the 'mrsh' are applicable to manage wireless motes and upload new applications.

The WLIP gateway functionality is not built-in but provided by the WLIP gateway assembly. It is downloaded using the `moma-load` command in the 'mrsh'. It is configured and managed using LIP messages where a number of 'mrsh' commands simplify the handling of WLIP.

The following command uploads the most recent version of the assembly on the simulated mote 'a0' and starts the WLIP gateway protocol on it:

```
a0 wlip-setup
```

Use 'help wlip-setup' to see the options for the command, e.g. to modify the channel used by the WLIP gateway.

```
a0 wlip-appeal
```

A 'wlip-appeal' lets the WLIP gateway request 'HELLO' messages from the wireless motes and is useful to gather a list of wireless motes still connected to the gateway.

WLIP is not a battery efficient protocol as wireless motes never switch off the receiver. Messages are sent using CSMA/CA and are retransmitted on error for a configurable number of times. The destination mote for a unicast message is addresses using its EUI-64 (extended address). Broadcast messages are received by all WLIP motes in range.

6.4.5. 6LoWPAN / MRv6

> Documentation about 6LoWPAN library can be found at section "6LoWPAN Network"

6.5. Examples and demos

Mote Runner comes with a large number of examples and demo applications, including using the generic Mote Runner API to implement sensor drivers, as well as controlling the radio for custom wireless protocols. The examples folder from the distribution contains the source code and further documentation for each example.

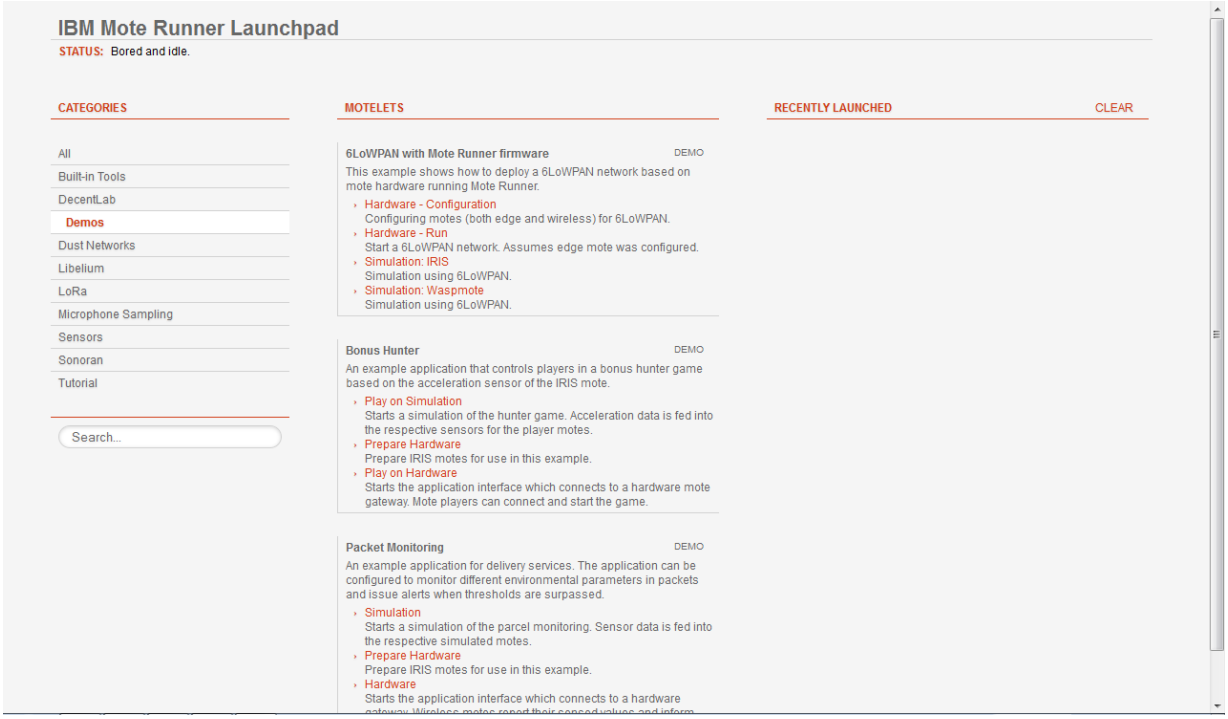


Figure 34: Mote Runner Launchpad

A good starting point with Mote Runner examples are the interactive tutorials. To access the interactive tutorials, simply start the Mote Runner shell (**mrsh**) process, then open Firefox, and go to **http://localhost:5000**. Access the **Demos** category, as highlighted in the picture bellow.

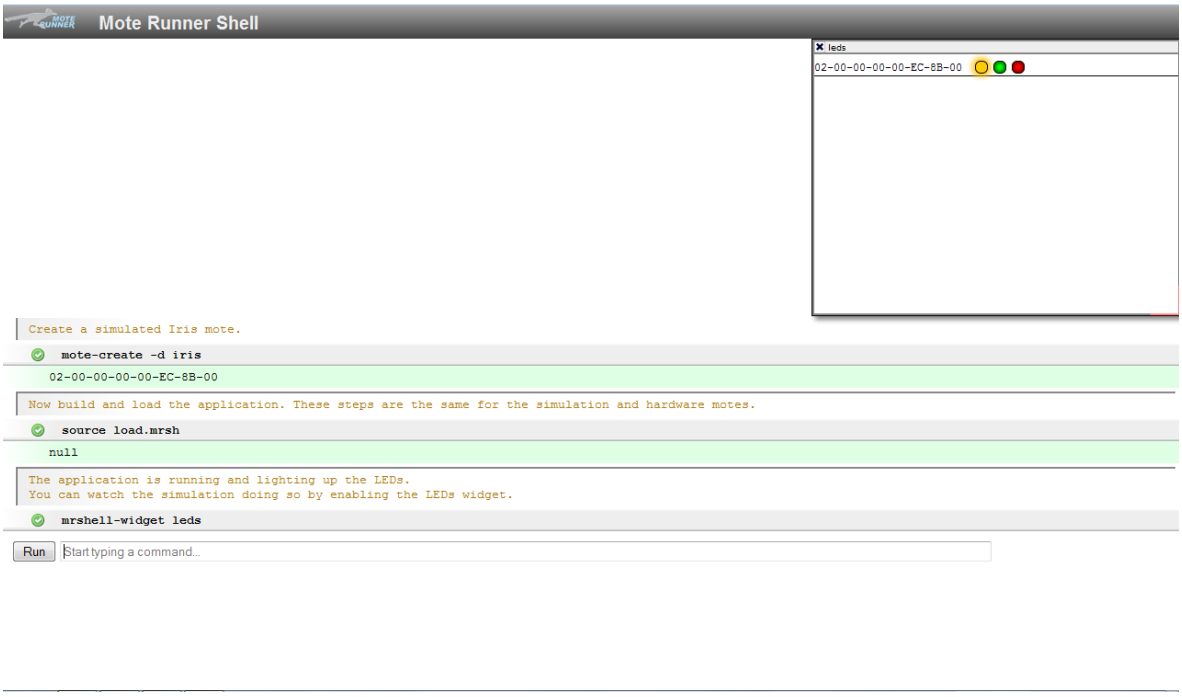


Figure 35: Web interface for Mote Runner Shell (mrsh)

The tutorial use the Web-based Mote Runner shell shown below. The Web mrsh can be accessed at **http://localhost:5000/mrshell**. Compared to the standard command-line interface, it brings a number of additional features such as an LED widget. The LEDs widget which can be loaded and activated using the **mrshell-widget leds** command.

The Web-based shell is extensible and additional widgets can be created using HTML and Javascript.

6.6. Mote Runner API Documentation

More details about the architecture can be found in the online documentation (**http://localhost:5000/doc**).

Note that the Mote Runner Shell (**mrsh**) must be started to access the online documentation. Also note that the supported Web browser is **Firefox**.

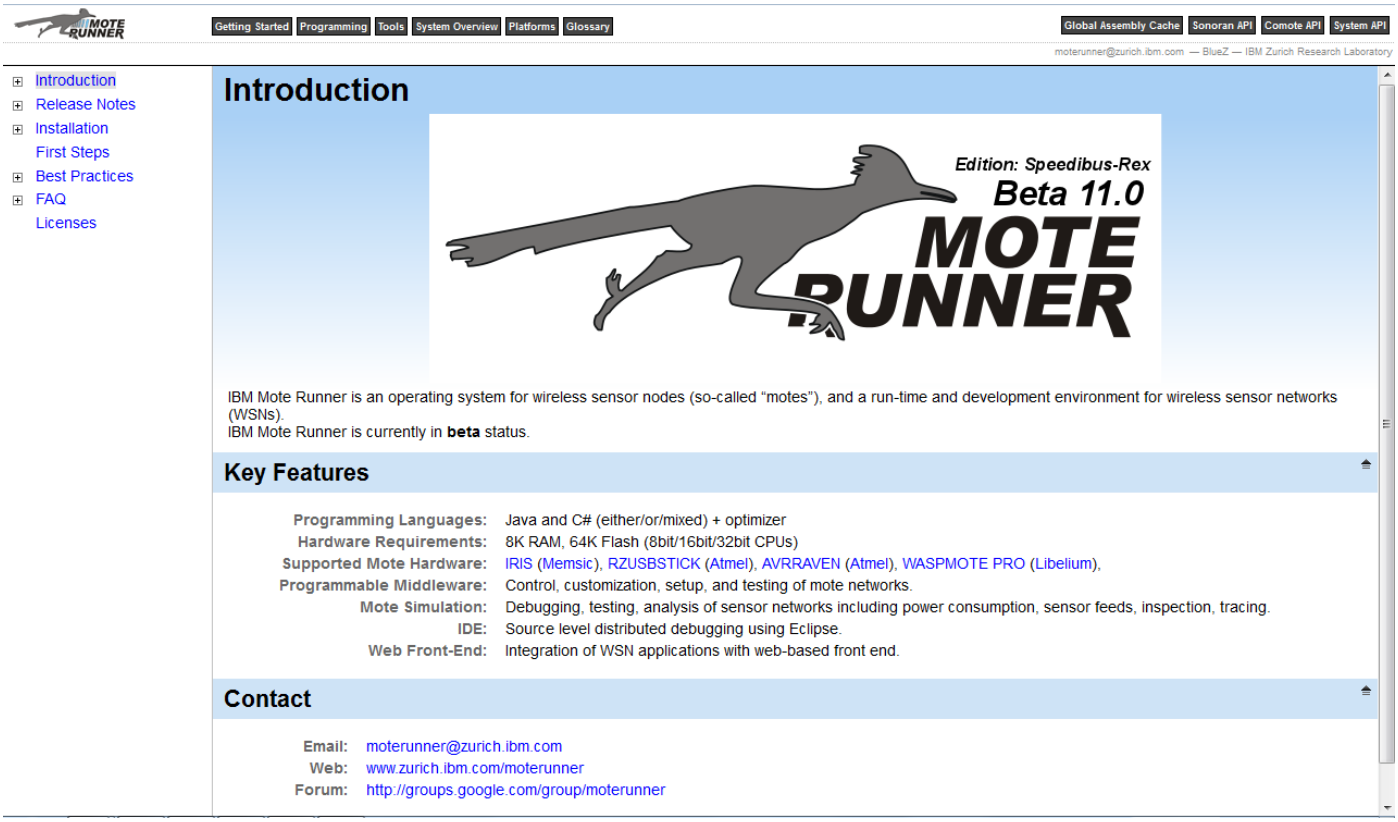


Figure 36: Mote Runner documentation entry point

7. 6LoWPAN / MRv6

7.1. Introduction

Mote Runner comes with a 6LoWPAN-based protocol, called MRv6, which can be installed on motes to provide IPv6-based connectivity.

MRv6 is a TDMA based multi-hop network which allows for an IPv6 based communication between hosts in the Internet and motes in the wireless network.

Datagram packets transferred in the wireless network make use of the IP header compression (IPHC) as specified in RFC4944 and RFC6282 for 6LoWPAN [1] [2]. In contrast to LIP and WLIP, MRv6 is not a built-in component, but fully implemented in C\#, packaged in assemblies and installed using the default Mote Runner upload mechanisms. The protocol is suited for data gathering applications where motes periodically send data to a remote host and are rarely subject to updates or requests from remote hosts. Communication between motes is expected to be rather uncommon, too.

The whole MRv6 network tree is managed by the edge mote and only very limited routing functionality is implemented in the wireless motes. The edge mote decides upon association requests, assigns communication schedules between wireless nodes, and determines the routes in the network. A host application can freely detach from or attach to an an established network without interfering with the tree management. Once attached, it can query the network, communicate with motes and retrieve data from them.

The format of application data packets exchanged in the network adheres to a subset of the 6LoWPAN specifications and constitutes a compressed representation of standard IPv6 packets. MRv6 features a tunnel program on OSX and Linux for hosts attached to an edge mote which maps IPv6 to 6lowpan packets and vice versa and forwards them between remote Internet hosts and the edge mote. As the **tunnel** implements a virtual network interface, the mapping occurs seamless for host and mote applications. Both wireless motes and hosts can be reached and identified using IPv6 addresses.

MRv6 supports a shell and programming interface to manage, debug, query and communicate with a wireless network, either physical or simulated. The tunnel is only required if communication with external host applications based on IPv6 is necessary.

7.2. 6LoWPAN Libraries

The 6LoWPAN library for mote runner is called MRv6.

In contrast to the default communication protocols LIP and WLIP, MRv6 is not a built in Mote Runner component, but fully implemented in C#, packaged in assemblies and installed using the default Mote Runner upload mechanisms. The protocol is suited for data gathering applications where motes periodically send data to a remote host and are rarely subject to updates or requests from remote hosts. Communication between motes is expected to be rather uncommon, too.

The whole MRv6 network tree is managed by the edge mote and only very limited routing functionality is implemented in the wireless motes. The edge mote decides upon association requests, assigns communication schedules between wireless nodes, and determines the routes in the network. A host application can freely detach from or attach to an an established network without interfering with the tree management. Once attached, it can query the network, communicate with motes and retrieve data from them.

The format of application data packets exchanged in the network adheres to a subset of the 6lowpan specifications and constitutes a compressed representation of standard IPv6 packets. MRv6 features a tunnel program for hosts attached to an edge mote which maps IPv6 to 6lowpan packets and vice versa and forwards them between remote Internet hosts and the edge mote. As the tunnel implements a virtual network interface, the mapping occurs seamless for host and mote applications. Both wireless motes and hosts can be reached and identified using IPv6 addresses.

MRv6 supports a shell and programming interface to manage, debug, query and communicate with a wireless network, either physical or simulated. The tunnel is only required if communication with external host applications based on IPv6 is necessary.

Files:

(Located on the <Mote runner folder>/examples/mrv6)

- **tunnel:** a simple program to setup a virtual IPv6 network interface which routes packets between the network interface and a hardware edge mote or between the network interface and Sonoran (and the simulation). Only supported on OSX and linux.
- **src:** directory with sources of the MRv6 protocol implementation, for both edge as well as wireless motes.
- **apps:** folder with sample applications.
- **java:** a library to be executed on a host. The library connects to the edge mote using UDP and allows for the event notification about mote appearances and disappearances. It is installed in lib/java.
- **lib:** contains the folder 'js' with the Javascript code for Sonoran to support the network protocol in the simulation and the compiled java library.

More examples can be found in examples/mrv6 inside the Mote Runner folder.

Complete API documentation can be found after generating it here:
http://localhost:5000/gac/mrv6-lib-1.0.htm?fqn=r:com.ibm.saguaro.mrv6.Address

In order to compile and generate MRv6 libraries documentation run the following commands on examples/mrv6/src:

```
make clean; make

make doc
```

7.3. Using MRv6 Libraries

On this example one mote will be configured as the edge-mote (gateway) and some other motes will be end-devices. Using the Linux or Mac OSX tunnel application a virtual ipv6 interface will be configured in order to route information between the edge mote and other ipv6 networks. Nodes of this WSN will be accessible using its IPv6 address (through the edge-mote that is connected to the tunnel application).

7.3.1. Setup

First of all MRv6 libraries must be compiled before using them.

In order to compile and generate MRv6 libraries documentation run the following commands on examples/mrv6/src folder:

```
make clean; make

make doc
```

1. Connect one Waspmote to a PC. Open *mrsh* on that PC and load the assembly *mrv6-edge* on the Waspmote. This assembly is located on the gac so there is no need to specify the path. This Waspmote will be the edge-mote. Once the assembly is loaded we can disconnect the Waspmote from the PC.

Instructions to load the assembly using USB cable (from mrsh):

```
> mote-create -p /dev/ttyUSB0

> moma-load mrv6-edge
```

This mote will be connected through an Ethernet interface. The network configuration can be set using *moma-ipv4* command. This configuration will be stored on the EEPROM memory so it will stay configured even if the mote is restarted. For example network parameters can be set as follows (inside mrsh):

```
> moma-ipv4 --ip 192.168.1.226/24 --gateway 192.168.1.1 --udp 9999
```

2. Load the mrv6-lib assembly on the end nodes. Connect them to a PC, this step can be applied one by one to every mote. This assembly is located on the gac so there is no need to specify the path.

Instructions to load the assembly using USB cable (from mrsh):

```
> mote-create -p /dev/ttyUSB0

> moma-load mrv6-lib
```

3. Load an application assembly that uses MRv6 libraries on every end node. For this example the application used will be reply located in *examples/mrv6/apps/reply* inside the Mote Runner folder. For compiling the application the following commands should be executed inside the *examples/mrv6/apps/reply* directory:

```
> make clean; make
```

Once the application is compiled we can load it on the motes. From mrsh inside *examples/mrv6/apps/reply* directory (this may be repeated with every mote):

```
> mote-create -p /dev/ttyUSB0

> moma-load sense
```

4. Launch the tunnel application on the PC

7.3.2. Running the example

1. Now all motes should be unplugged from the PC, USB connector is no longer needed. Radio modules and USB connector can't work at the same time, so battery should be used. [NOTE: you can have the battery recharged the whole time by plugging the USB cable to a power plug socket by using the 110/220V to 5V transformer provided by Libelium. See section "USB connection and Radio Modules" for more information]. The edge-mote is the only one that should be connected through Ethernet Module directly to the PC (using a crossover Ethernet cable) or to a network accessible by the PC (using Ethernet cable). This edge-mote should be accessible from the PC if network configuration on the mote (done in step 1) is suitable for your PC or network configuration. Lights on the Ethernet Module should blink when it is connected to the network. We can test if the mote is accessible doing a ping to its ipv4 address.

2. The end nodes must be powered only by battery (or USB to power plug) and the 6LoWPAN module should be attached to the mote.

3. In order to start the edge mote run the following commands in mrsh:

```
> source ../../lib/js/mrv6.js

> a0 v6-setup --MAX_DEPTH=12 --NUM_CHILDREN=5 --MAX_CHILDREN=5 --MAX_MOTES=6 --RECV_SAFETY_MILLIS=3 --R24_SLOT_RCV_MILLIS=12 --R24_SLOT_GAP_MILLIS=15 --R24_BEACON_GAP_MILLIS=20 --INFO_INTERVAL_CNT=0
```

4. As end-nodes are switched on they attach automatically to the WSN network so the edge-mote is now aware of new nodes. Different routes can be established automatically, maybe some nodes reach directly to the edge-node or maybe they are too far and they use other end-nodes to reach the edge-node.

5. In this example UDP packets can be sent to a mote using its IPv6 address and the mote send back the same UDP packet.

IPv6 addresses from nodes can be retrieved for example with the command v6-connect in mrsh:

```
> v6-connect
```

For example a node can have the EUI-64 IP address: `0200:0000:00AE:2F01`

Due to the tunnel application default configuration nodes can be reached using `fc00:db8:5::<EUI-64>` so for this example the node that is going to be used will be accessible with the IP `fc00:db8:5::0200:0000:00AE:2F01`

Information can be sent to the nodes from a machine capable to reach the PC that is running the tunnel application using IPv6. For simplicity it can be done from the same PC running the tunnel application. The information in this example will be sent using the Netcat tool (IPv6 version). Using it for sending UDP packets is as follows:

```
> nc6 -u fc00:db8:5::0200:0000:00AE:2F01 102
```

Now for example the phrase hello world can be sent just typing it and pressing enter.

This should return the same phrase:

```
 > nc6 -u fc00:db8:5::0200:0000:00AE:2F01 1024
      hello world
      hello world
```

7.4. Quick start development with the Waspmote Mote Runner Lab Kit

On section "Using MRv6 libraries" is explained how to configure WSN, how to compile MRv6 libraries and how to load applications using those libraries.

In this section Lab Kit motes will be running a custom application based on the reply application (*<Mote Runner folder>/examples/mrv6/apps/reply*).

This application will send back an analog reading from ADC channel 0 instead of replying the same information that it receives.

The value returned by the ADC will be converted by the application to ASCII characters representing the hexadecimal value of channel 0 lecture.

Information will be sent from a client using Netcat and it will reach the mote specified by its IPv6 address, the mote will read ADC 0 and send the value back to the client.

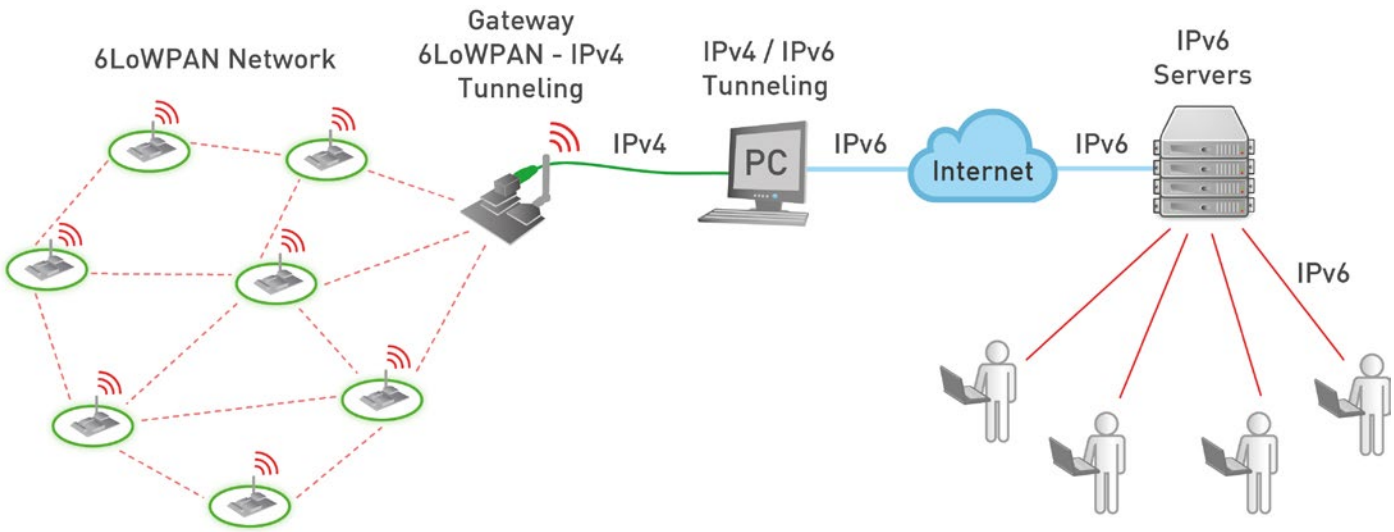


Figure 37: Clients running Netcat on laptop exchange information with WSN nodes

7.4.1. Application code

The application code is the following

adcread.cs:

```
namespace com.ibm.saguaro.gateway.generic {
    using com.ibm.saguaro.system;
    using com.ibm.saguaro.mrv6;
    using com.ibm.saguaro.util;

    public class ReplySocket : UDPSocket {

        internal static uint LOCAL_PORT = 1024;
        internal static ReplySocket socket = new ReplySocket();
        internal static ADC adc;

        public ReplySocket() {
            this.bind(LOCAL_PORT);
               adc = new ADC();
               adc.open(0x01,GPIO.NO_PIN,0, 0);
        }


        public override int onPacket(Packet packet) {
            // An UDP packet has reached this mote

            // Toggle LED 2
            LED.setState(2, (byte)(LED.getState(2)^1));

            // We read ADC channel 0
            uint adcRead = adc.readChannel(0);

            // We store the ADC reading on a byte array
            byte[] buf = new byte[2];
            Util.set16be(buf,0,adcRead);

            // ADC value hexadecimal representation is transformed to ASCII.
            // Doing so we can see the hexa value directly on Netcat
            buf = adc2Ascii(buf);

            uint len = packet.payloadLen;
            try {
                packet.swap(len);
            } catch {
                return 0;
            }

            Util.copyData(buf, 0, packet.payloadBuf, packet.payloadOff, 4);
            this.send(packet);
            return 0;

        }

        // Get ASCII representation of a value in hexadecimal
        private byte[] adc2Ascii(byte[] adcValue){

            byte[] aux = new byte[4];

            aux[0] = (byte)((adcValue[0] & 0xF0) >> 4);
            aux[1] = (byte)(adcValue[0] & 0x0F);
            aux[2] = (byte)((adcValue[1] & 0xF0) >> 4);
            aux[3] = (byte)(adcValue[1] & 0x0F);

            for(int i=0;i<4;i++){
                if(aux[i] < 9){
```

```
                    aux[i] += 48;
                }else{
                    aux[i] += 55;
                }
            }

            return aux;
        }
    }
}
```

This application must be compiled referencing the MRv6 library mrv6-lib:

```
> mrc –assembly=adcread-1.0 –ref=mrv6-lib adcread.cs
```

At this point the assembly adcread-1.0 has been created and it should be uploaded to WSN nodes.

7.4.2. Retrieving values from the WSN

Once the network is configured, tunnel is running and the application is uploaded to the end-nodes they can be queried to send back the ADC 0 channel reading.

IPv6 addresses from nodes can be retrieved for example with the command v6-connect in mrsh:

```
> v6-connect
```

For example a node can have the EUI-64 IP address: `0200:0000:00AE:2F01`

Due to the tunnel application default configuration nodes can be reached using `fc00:db8:5::<EUI-64>` so for this example the node that is going to be used will be accessible with the IP `fc00:db8:5::0200:0000:00AE:2F01`

Information can be sent to the nodes from a machine capable to reach the PC that is running the tunnel application using IPv6. For simplicity it can be done from the same PC running the tunnel application. The information in this example will be sent using the Netcat tool (IPv6 version). Using it for sending UDP packets is as follows:

```
> nc6 -u fc00:db8:5::0200:0000:00AE:2F01 102
```

Now 4 bytes need to be sent to the node. For example "0000". Typing 0000 and pressing enter will send those bytes to the node.

The value from the ADC will appear in screen:

```
> nc6 -u fc00:db8:5::0200:0000:00AE:2F01 1024

      0000

      03FF
```

In this case the node sent 03FF (1023 in decimal representation) which is the maximum value the ADC can return. With the default threshold configured on the ADC this means 3.3V.

Four bytes are sent to the node because the application previously loaded on the node reuse the same packet structure for sending the value back, and the ASCII representation of the ADC value needs four bytes.

8. Support

There is a specific thread in the Forum which will be supported by both Libelium and IBM teams.

- Libelium Dev Team will answer to the Hardware questions.

- IBM Dev Team will answer to the Software / Networking / Data related questions.

Go the Waspmote Mote Runner Forum. **http://www.libelium.com/forum/viewforum.php?f=34**

9. Documentation Changelog

From v4.0 to v4.1:

- Changed Weather Meters name to Weather Station WS-3000

10. Certifications

10.1. CE

In accordance with the 1999/05/CE directive, Libelium Comunicaciones Distribuidas S.L. declares that the Waspmote device conforms to the following regulations:

EN 55022:1998

EN 55022:1998/A1:2000

EN 55022:1998/A2:2003

EN 61000-4-3:2002

EN 61000-4-3/A1:2002

EN 61000-4-3:2006

UNE-EN 60950-1:2007

Compliant with ETSI EN 301 489-1 V1.6.1, EN 300 328, Date: March 26, 2009

If desired, the Declaration of Conformity document can be requested using the Contact section at:

http://www.libelium.com/contact

Waspmote is a piece of equipment defined as a wireless sensor capture, geolocalization and communication device which allows:

- short and long distance data, voice and image communication
- capture of analog and digital sensor data directly connected or through probes
- wireless access enablement to electronic communication networks as well as local networks allowing cable free connection between computers and/or terminals or peripheral devices
- geospatial position information
- interconnection of wired networks with wireless networks of different frequencies
- interconnection of wireless networks of different frequencies between each other
- output of information obtained in wireless sensor networks
- use as a data storage station
- capture of environmental information through interface interconnection, peripherals and sensors
- interaction with the environment through the activation and deactivation of electronic mechanisms (both analog and digital)

10.2. FCC

Waspmote models:

Model 1- FCC (XBee PRO series 1 OEM + SIM900 GSM/GPRS module)
FCC ID: XKM-WASP01 comprising
- FCC ID: OUR-XBEEPRO
- FCC ID: UDV-0912142009007

Model 2- FCC (XBee PRO ZB series 2 + SIM900 GSM/GPRS module)
FCC ID: XKM-WASP02 comprising
- FCC ID: MCQ-XBEEPRO2*
- FCC ID: UDV-0912142009007

Model 3 - FCC (XBee 900MHz + SIM900 GSM/GPRS module)
FCC ID: XKM-WASP03 comprising
- FCC ID: MCQ-XBEE09P
- FCC ID: UDV-0912142009007

Installation and operation of any Waspmote model must assure a separation distance of 20 cm from all persons, to comply with RF exposure restrictions.

Module Grant Restrictions

FCC ID OUR-XBEEPRO

The antenna(s) used for this transmitter must be installed to provide the separation distances, as described in this filing, and must not be co-located or operating in conjunction with any other antenna or transmitter. Grantee must coordinate with OEM integrators to ensure the end-users of products operating with this module are provided with operating instructions and installation requirements to satisfy RF exposure compliance. Separate approval is required for all other operating configurations, including portable configurations with respect to 2.1093 and different antenna configurations. Power listed is continuously variable from the value listed in this entry to 0.0095W

FCC ID MCQ-XBEEPRO2

OEM integrators and End-Users must be provided with transmitter operation conditions for satisfying RF exposure compliance. The instruction manual furnished with the intentional radiator shall contain language in the installation instructions informing the operator and the installer of this responsibility. This grant is valid only when the device is sold to OEM integrators and the OEM integrators are instructed to ensure that the end user has no manual instructions to remove or install the device.

FCC ID: UDV-0912142009007

This device is to be used in mobile or fixed applications only. For other antenna(s) not described in this filing the antenna gain including cable loss must not exceed 7.3 dBi in the 850 MHz Cellular band and 12.7 dBi in the PCS 1900 MHz band, for the purpose of satisfying the requirements of 2.1043 and 2.1091. The antenna used for this transmitter must be installed to provide a separation distance of at least 20 cm from all persons, and must not be co-located or operating in conjunction with other antennas or transmitters within a host device, except in accordance with FCC multi- transmitter product procedures. Compliance of this device in all final product configurations is the responsibility of the Grantee. OEM integrators and end-users must be provided with specific information required to satisfy RF exposure compliance for all final host devices and installations.

10.3. IC

Waspmote models:

Model 1- IC (XBee PRO series 1 OEM + SIM900 GSM/GPRS module)
IC: 8472A-WASP01 comprising
- IC: 4214A-XBEEPRO
- IC: 8460A-20100108007

Model 2- IC (XBee PRO ZB series 2 + SIM900 GSM/GPRS module)
IC: 8472A-WASP02 comprising
- IC: 1846A-XBEEPRO2
- IC: 8460A-20100108007

Model 3- IC (XBee 900MHz + SIM900 GSM/GPRS module)
IC: 8472A-WASP03 comprising
- IC: 1846A-XBEE09P
- IC: 8460A-20100108007

The term "IC:" before the equipment certification number only signifies that the Industry Canada technical specifications were met.

Installation and operation of any Waspmote model must assure a separation distance of 20 cm from all persons, to comply with RF exposure restrictions.

10.4. Use of equipment characteristics

- Equipment to be located in an area of restricted access, where only expert appointed personnel can access and handle it.
- The integration and configuration of extra modules, antennas and other accessories must also be carried out by expert personnel.

10.5. Limitations of use

The ZigBee/IEEE 802.15.4 module has a maximum transmission power of 20dBm.

It is regulated according to EN 301 489-1 v 1.4.1 (202-04) and EN 301 489-17 V1.2.1 (2002-08). The configuration software must be used to limit to a maximum power of 12'11dBm (PL=0).

The 868MHz XBee module has a maximum transmission power of 27dBm. This module is regulated only for use in Europe.

The 900MHz XBee module has a maximum power of 20dBm. This module is regulated only for use in the United States.

The GSM/GPRS module has a power of 2W (Class 4) for the 850MHz/900MHz band and 1W (Class 1) for the 1800MHz and 1900MHz frequency band.

The 3G/GPRS module has a power of 0,25W for the UMTS 900MHz/1900MHz/2100MHz band, 2W for the GSM 850MHz/900MHz band and 1W DCS1800MHz/PCS1900MHz frequency band.

Important: *In Spain the use of the 850MHz band is not permitted. For more information contact the official organisation responsible for the regulation of power and frequencies in your country.*

The cable (pigtail) used to connect the radio module with the antenna connector shows a loss of approximately 0.25dBi for GSM/GPRS.

The broadcast power at which the WiFi, XBee 2.4GHz, XBee 868MHz, XBee 900MHz operate can be limited through the configuration software. It is the responsibility of the installer to choose the correct power in each case, considering the following limitations:

The broadcast power of any of the modules added to that of the antenna used minus the loss shown by the pigtail and the cable that joins the connector with the antenna (in the event of using an extra connection cable) must not exceed 20dBm (100mW) in the 2.4GHz frequency band and 27dBm for the 868MHz band, according to the ETSI/EU regulation.

It is the responsibility of the installer to configure the different parameters of the equipment correctly, whether hardware or software, to comply with the pertinent regulation of each country in which it is going to be used.

Specific limitations for the 2.4GHz band.

- In Belgium, outdoor use is only on channels 11(2462MHz), 12(2467MHz) and 13(2472MHz) only. It can be used without a licence if it is for private use and at a distance less than 300m. Over longer distances or for public use, an I'IBPT licence is required.
- In France the use of channels 10(2457MHz), 11(2462MHz), 12(2467MHz) and 13(2472MHz) is restricted. A licence is required for any use both indoors and outdoors. Contact ARCEP (**http://www.arcep.fr**) for further information.
- In Germany a licence is required for outdoor use.
- In Italy a licence is required for indoor use. Outdoor use is not permitted.
- In Holland a licence is required to outdoor use.
- In Norway, use near Ny-Alesund in Svalbard is prohibited. For further information enter Norway Posts and Telecommunications (**http://www.npt.no**).

Specific limitations for the 868MHz band.

- In Italy the maximum broadcast power is 14dBm.
- In the Slovakian Republic the maximum broadcast power is 10dBm.

IMPORTANT

It is the responsibility of the installer to find out about restrictions of use for frequency bands in each country and act in accordance with the given regulations. Libelium Comunicaciones Distribuidas S.L does not list the entire set of standards that must be met for each country. For further information go to:

CEPT ERC 70-03E - Technical Requirements, European restrictions and general requirements: **http://www.ero.dk**

R&TTE Directive - Equipment requirements, placement on market: **http://www.ero.dk**

11. Maintenance

- In this section, the term "Waspmote" encompasses both the Waspmote device itself as well as its modules and sensor boards.
- Take care when handling Waspmote, do not let it fall, knock it or move it suddenly.
- Avoid having the devices in high temperature areas as it could damage the electronic components.
- The antennas should be connected carefully. Do not force them when fitting them as the connectors could be damaged.
- Do not use any type of paint on the device, it could harm the operation of the connections and closing mechanisms.

12. Disposal and recycling

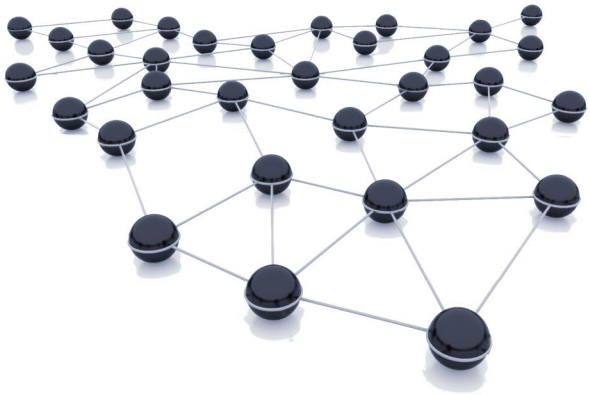
- In this section, the term "Waspmote" encompasses both the Waspmote device itself as well as its modules and sensor boards.
- When Waspmote reaches the end of its useful life, it must be taken to an electronic equipment recycling point.
- The equipment must be disposed of in a selective waste collection system, and not that for urban solid residue. Please manage its disposal properly.
- Your distributor will inform you about the most appropriate and environmentally friendly disposal process for the used product and its packaging.

ANEXO III

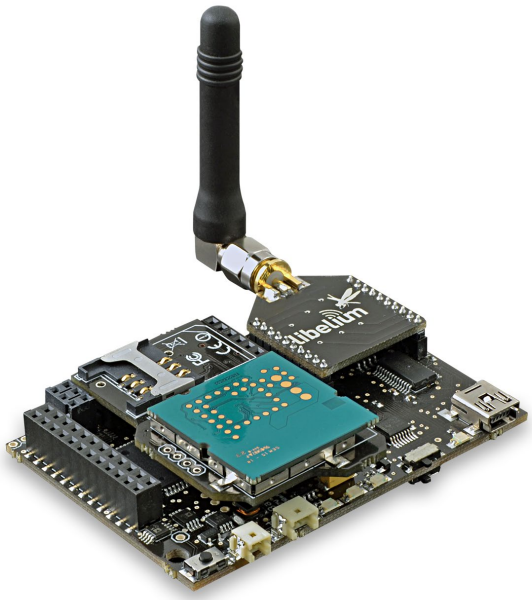
WASPMOTE DATASHEET

Waspmote

Datasheet



Waspmote



General data:

Microcontroller:	ATmega1281
Frequency:	14.7456 MHz
SRAM:	8KB
EEPROM:	4KB
FLASH:	128KB
SD Card:	2GB
Weight:	20gr
Dimensions:	73.5 x 51 x 13 mm
Temperature Range:	[-10ºC, +65ºC]
Clock:	RTC (32KHz)

Consumption:

ON:	15mA
Sleep:	55µA
Deep Sleep:	55µA
Hibernate:	0.07µA

Operation without recharging: 1 year *

** Time obtained using the Hibernate mode as the energy saving mode*

Inputs/Outputs:

7 Analog (I), 8 Digital (I/O), 1 PWM,
2 UART, 1 I2C, 1USB, 1SPI

Electrical data:

Battery voltage:	3.3 V - 4.2V
USB charging:	5 V - 100mA
Solar panel charging:	6 - 12 V - 280mA

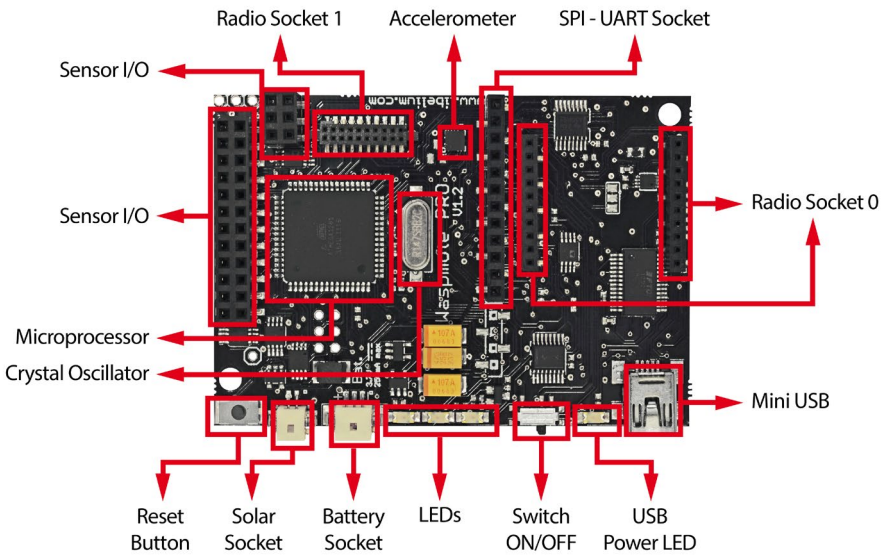


Figure: Waspmote Board Top

Built-in sensors on the board:

Temperature (+/-): -40ºC , +85ºC. Accuracy: 0.25ºC
Accelerometer: ±2g/±4g/±8g
Low power: 0.5 Hz/1 Hz/2 Hz/5 Hz/10 Hz
Normal mode: 50 Hz/100 Hz/400 Hz/1000 Hz

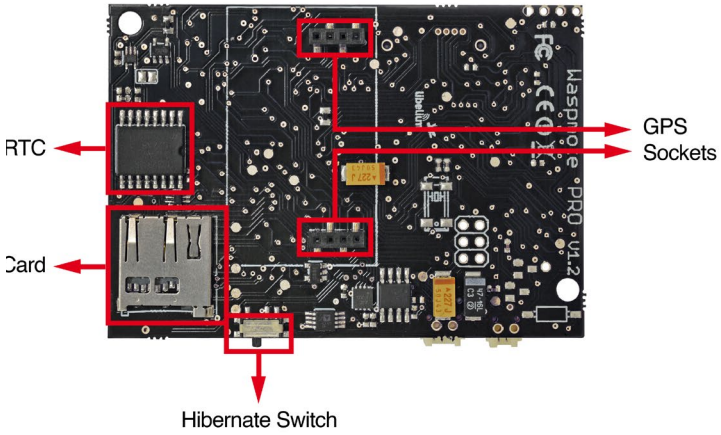


Figure: Waspmote Board Bottom

802.15.4/ZigBee

Model	Protocol	Frequency	txPower	Sensitivity	Range *
XBee-802.15.4-Pro	802.15.4	2.4GHz	100mW	-100dBm	7000m
XBee-ZB-Pro	ZigBee-Pro	2.4GHz	50mW	-102dBm	7000m
XBee-868	RF	868MHz	315mW	-112dBm	12km
XBee-900	RF	900MHz	50mW	-100dBm	10km

** Line of sight and Fresnel zone clearance with 5dBi dipole antenna*

Figure: XBee

Antennas: 2.4GHz: 2dBi / 5dBi
868/900MHz: 0dBi / 4.5dBi
Connector: RPSMA
Encryption: AES 128b
Control Signal: RSSI
Standards: XBee-802.15.4 - 802.15.4 Compliant / XBee-ZB - ZigBee-Pro v2007 Compliant
Topologies: star, tree, mesh

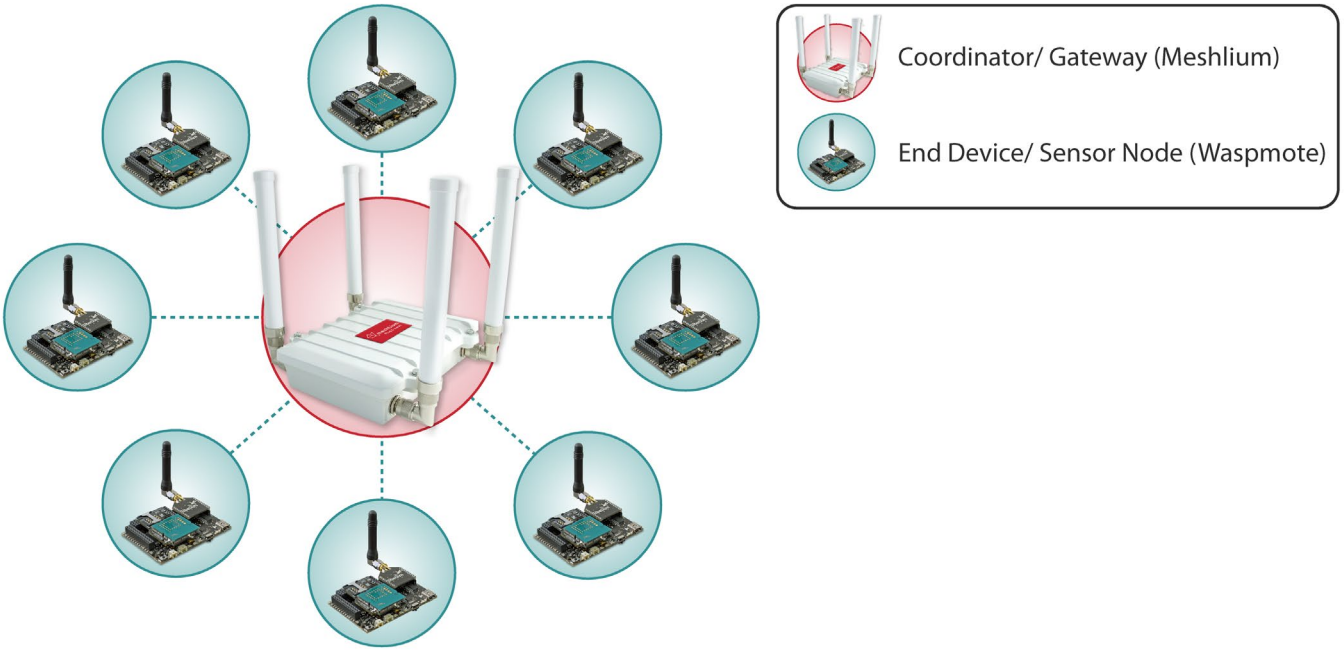


Figure: Star

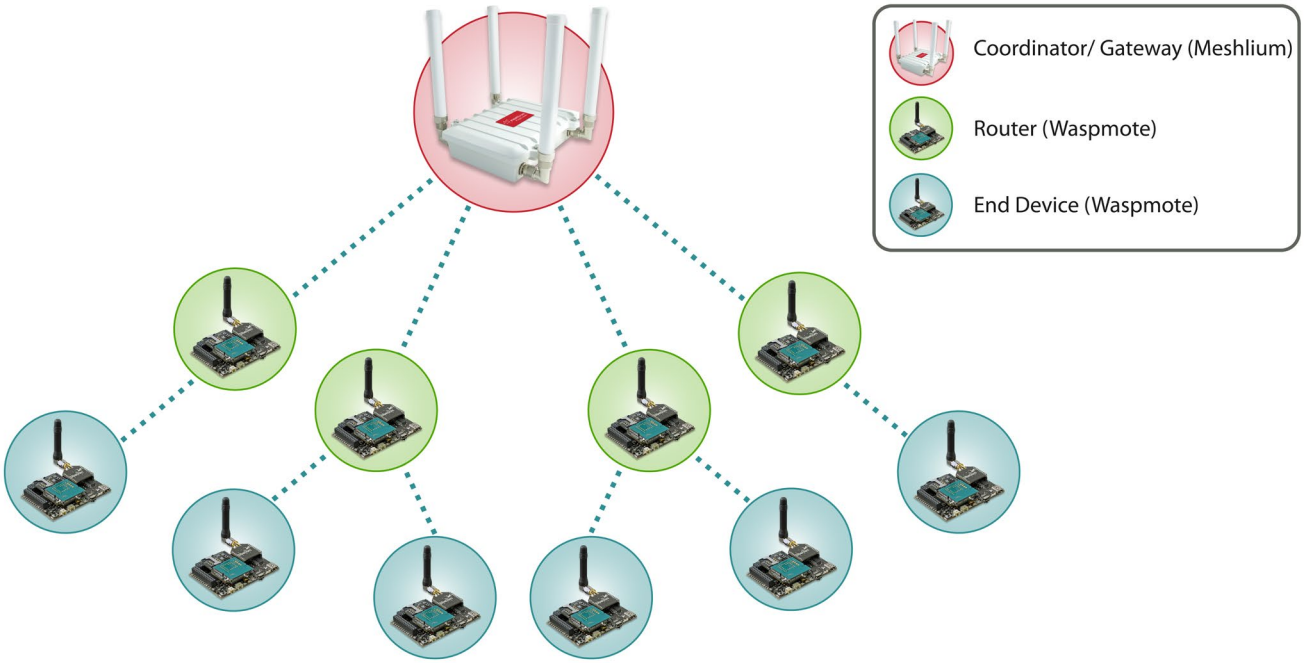


Figure: Tree

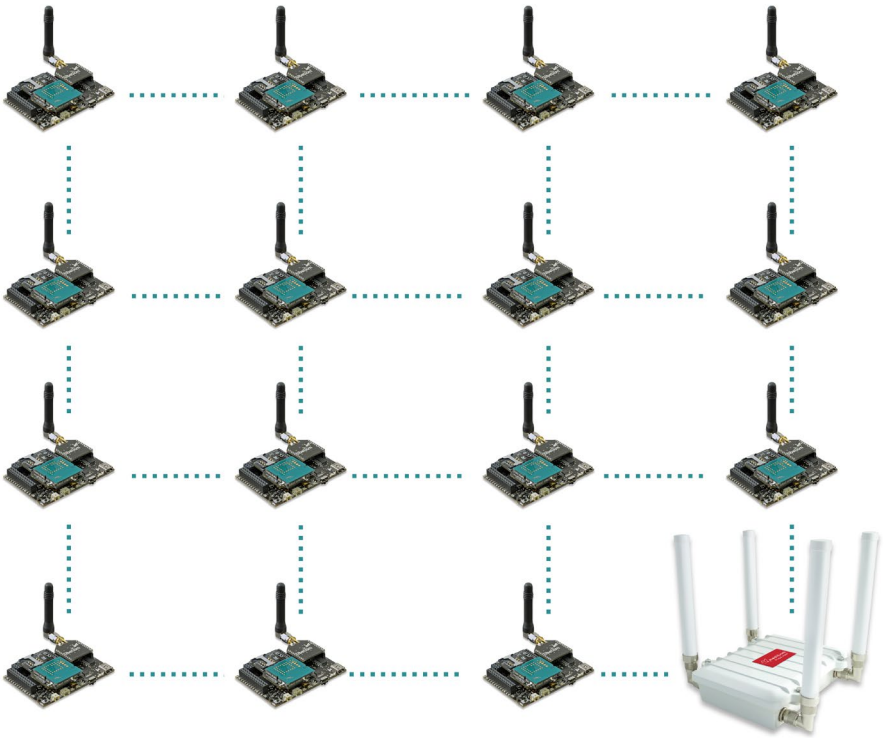


Figure: Mesh

Sigfox module

Frequency: ISM 868 MHz

TX Power: 14 dBm

ETSI limitation: 140 messages of 12 bytes, per module per day

Range: Typically, each base station covers some km. Check the Sigfox Network

Chipset consumption: TX: 49 mA @ +14 dBm

Radio Data Rate: 100 bps

Receive sensitivity: -126 dBm

Sigfox certificated: Class 0u (the highest level)

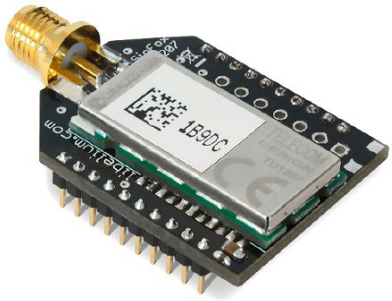


Figure: Sigfox module

Figure: Sigfox network

LoRa module

Model: Semtech SX1272

Frequencies available: 860-1000 MHz, fits both 868 (Europe) and 915 MHz (USA) ISM bands

Max TX power: 14 dBm

Sensitivity: -137 dBm

Range:

Line of Sight: 21+ km / 13.4+ miles (LoS and Fresnel zone clearance)

Non Line of Sight: 2+ km / 1.2+ miles (nLoS going through buildings, urban environment)

Figure: LoRa module

Antenna:

868 / 915 MHz: 0 / 4.5 dBi

Connector: RPSMA

Encryption: AES 128/192/256b (performed by Waspmote API)

Control Signal: RSSI

Topology: Star

Receiver/Central node: Meshlium LoRa, special Gateway LoRa (SPI) or another Waspmote or Plug & Sense! unit

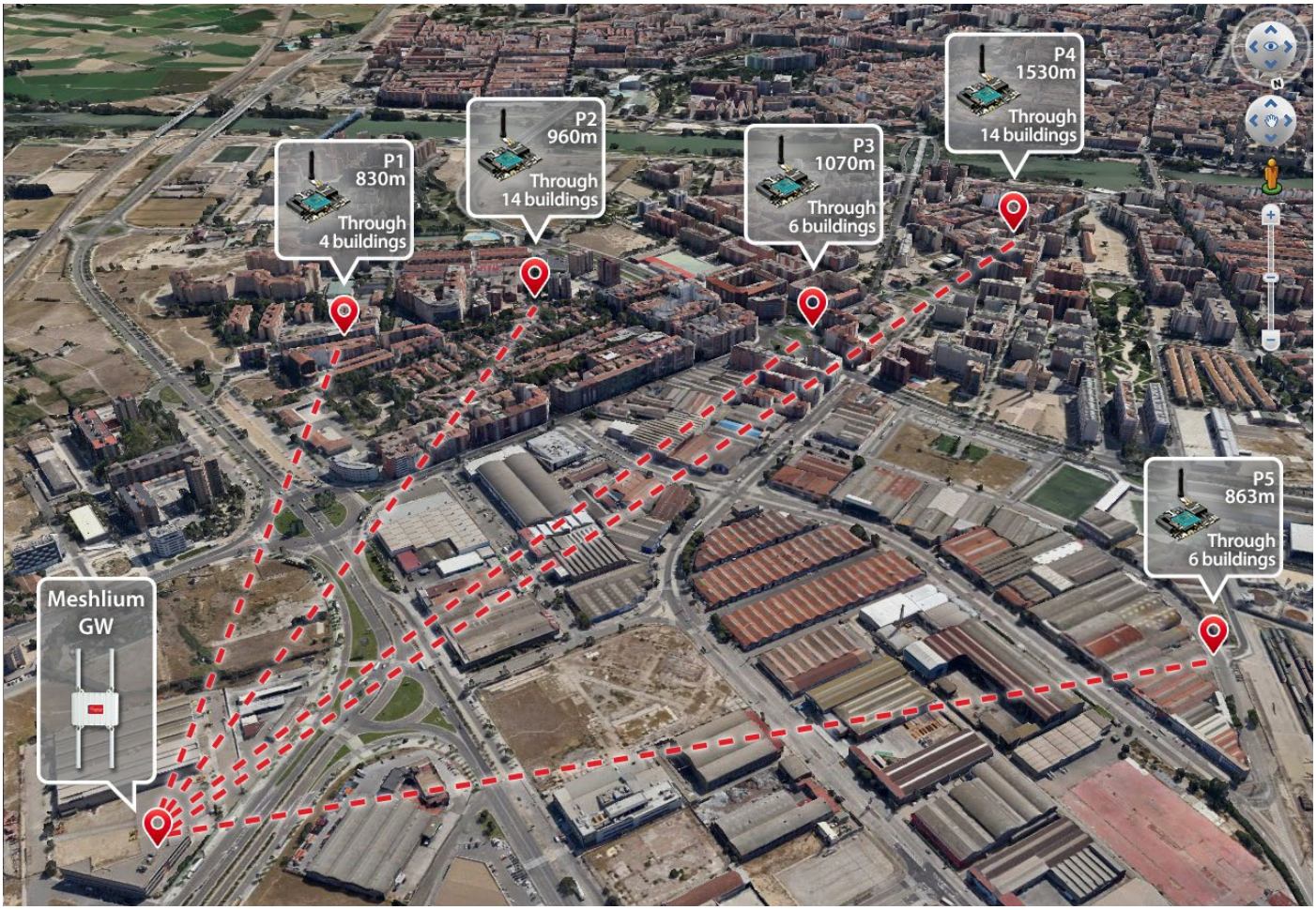


Figure: Star topology

Over the Air Programming (OTA)

There are two different OTA methodologies:

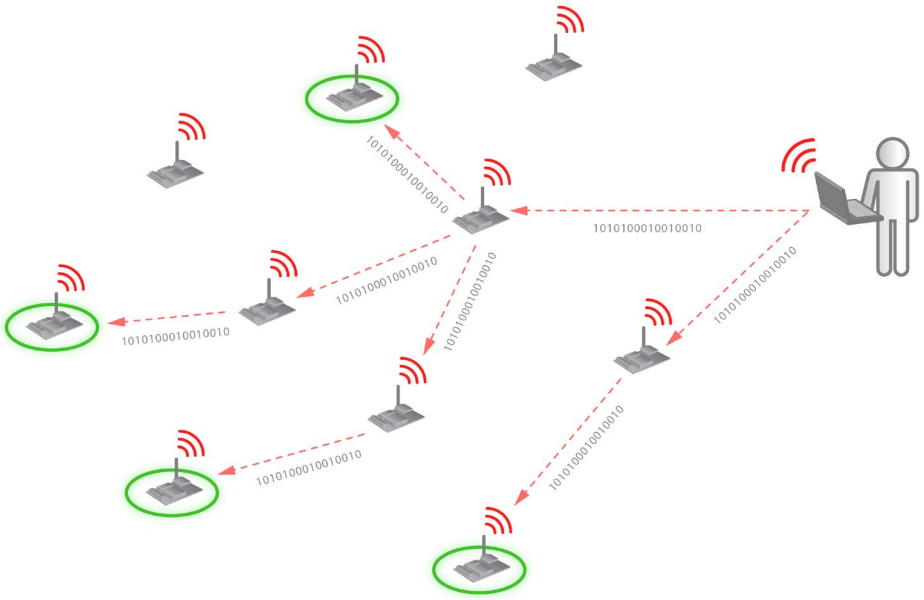
- OTA with 802.15.4/ZigBee modules
- OTA with 3G/GPRS/WiFi modules via FTP

OTA with 802.15.4/ZigBee modules

Benefits:

- Enables the upgrade or change of firmware versions without physical access
- Discover nodes in the area just sending a broadcast discovery query
- Upload new firmware in few minutes
- No interferences: OTA is performed using a change of channel between the programmer and the desired node so no interferences are generated to the rest of the nodes

Over The Air Programming with 802.15.4 / ZigBee



Topologies:

- Direct access: when the nodes are accessed in just one hop (no forwarding of the packets is needed).
- Multihop: when the nodes are accessed in two or more hops. In this mode some nodes have to forward the packets sent by the Gateway in order to reach the destination.

Modes:

- Unicast: Reprogram an specific node
- Multicast: Reprogram several nodes at the same time sending the program just once
- Broadcast: Reprogram the entire network sending the program just once

OTA with 3G/GPRS/WiFi modules via FTP

Benefits:

- Enables the upgrade or change of firmware versions without physical access.
- Upgrades the new firmware by querying a FTP server which helps to keep battery life.
- Upload new firmware in few minutes.

Topologies:

- Protocols which support FTP transmissions are directly connected to the Network Access Point.

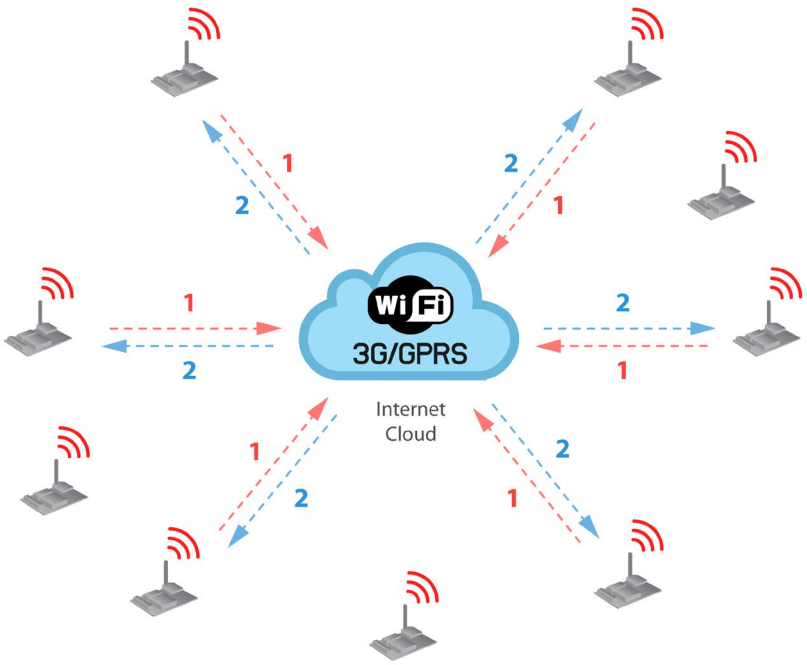


Figure: OTA with GPRS/3G/WiFi fundamentals

Encryption Libraries

The new Encryption Libraries are designed to add to the Waspmote sensor platform the capabilities necessary to protect the information gathered by the sensors. To do so **two cryptography layers** are defined:

- **Link Layer**: In the first one all the nodes of the network share a common **preshared key** which is used to encrypt the information using **AES 128**. This process is carried out by specific hardware integrated in the same 802.15.4/ZigBee radio, allowing the maximum efficiency of the sensor nodes energy consumption. This first security layer ensures no third party devices will be able to even connect to the network (access control).
- **Secure Web Server Connection:** The third security technique is carried out in Meshlium -the Gateway- where **HTTPS** and **SSH** connections are used to send the information to the Cloud server located on the Internet.

A third optional encryption layer allows each node to encrypt the information using the Public key of the Cloud server. Thus, the information will be kept confidentially all the way from the sensor device to the web or data base server on the Internet.

Transmission of sensor data:

Information is encrypted in the application layer via software with **AES 256** using the key shared exclusively between the origin and the destination. Then the packet is encrypted again in the link layer via hardware with **AES 128** so that only trusted packets be forwarded, ensuring access control and improving the usage of resources of the network.

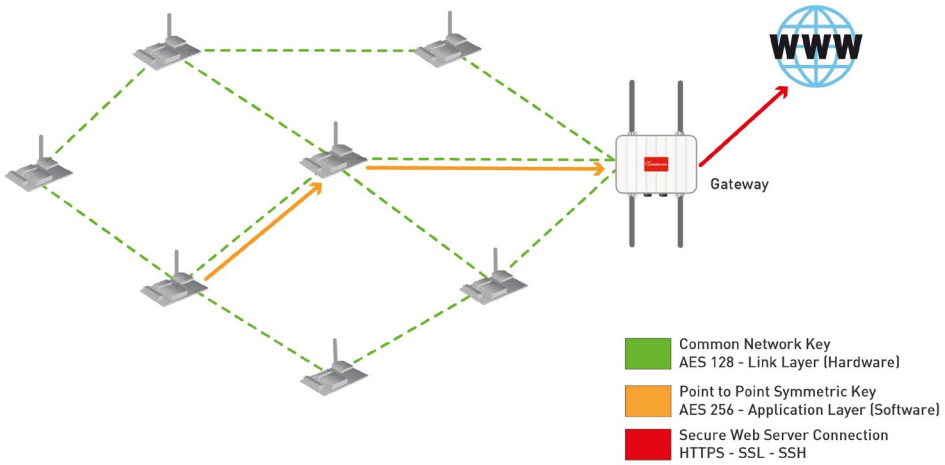


Figure: Communication diagram

WiFi

Protocols: 802.11b/g - 2.4GHz
TX Power: 0dBm - 12dBm (variable by software)
RX Sensitivity: -83dBm
Antenna connector: RPSMA
Antenna: 2dBi/5dBi antenna options
Security: WEP, WPA, WPA2
Topologies: AP
802.11 roaming capabilities

Actions:

- TCP/IP - UDP/IP socket connections
- HTTP web connections
- FTP file transfers
- Direct connections with iPhone and Android
- Connects with any standard WiFi router
- DHCP for automatic IP assignation
- DNS resolution enabled

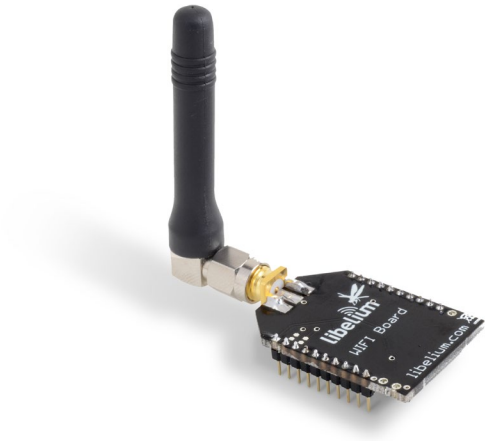


Figure: WiFi Module

GSM/GPRS

Model: SIM900 (SIMCom)
Quadband: 850MHz/900MHz/1800MHz/1900MHz
TX Power: 2W(Class 4) 850MHz/900MHz, 1W(Class 1) 1800MHz/1900MHz
Sensitivity: -109dBm
Antenna connector: UFL
External Antenna: 0dBi
Consumption in sleep mode: 1mA
Consumption in power off mode: 0mA

Figure: GSM/GPRS

Actions:

- Making/Receiving calls
- Making 'x' tone missed calls
- Sending/Receiving SMS
- Single connection and multiple connections TCP/IP and UDP/IP clients
- TCP/IP server
- HTTP Service
- FTP Service (downloading and uploading files)

GPRS + GPS

Figure: GPRS+GPS

Model: SIM928 (SIMCom)

GPRS features:
Quadband: 850MHz/900MHz/1800MHz/1900MHz
TX Power: 2W (Class 4) 850MHz/900MHz, 1W (Class 1) 1800MHz/1900MHz
Sensitivity: -109dBm
Antenna connector: UFL
External Antenna: 0dBi
Consumption in sleep mode: 1mA
Consumption in power off mode: 0mA

GPS features:

Time-To-First-Fix: 30s (typ.)
Sensitivity:

- Tracking: -160 dBm
- Adquisition: -147 dBm

Accuracy horizontal position : <2.5m CEP
Power consumption (GSM engine in idle mode):

- Acquisition : 72mA
- Tracking : 67mA

Actions:

- Making/Receiving calls
- Making 'x' tone missed calls
- Sending/Receiving SMS
- Single connection and multiple connections TCP/IP and UDP/IP clients
- TCP/IP server
- HTTP Service
- FTP Service (downloading and uploading files)
- GPS receiver

3G + GPS module

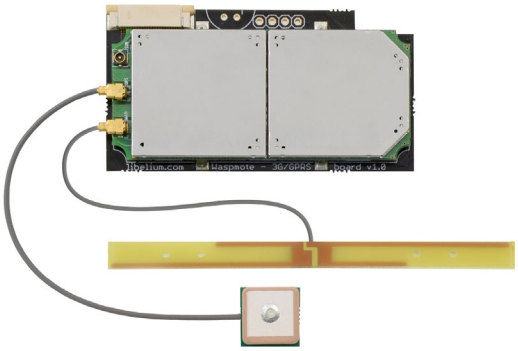


Figure: 3G/GPRS board

Model: SIM5218E (SIMCom)
Tri-Band UMTS 2100/1900/900MHz
Quad-Band GSM/EDGE, 850/900/1800/1900 MHz
HSDPA up to 7.2Mbps
HSUPA up to 5.76Mbps
TX Power:

- UMTS 900/1900/2100 0,25W
- GSM 850MHz/900MHz 2W
- DCS1800MHz/PCS1900MHz 1W

Sensitivity: -106dBm
Antenna connector: UFL
External Antenna: 0dBi
Consumption in sleep mode (RF circuits power off previously): 1mA

Actions:

- WCDMA and HSPA 3G networks compatibility
- Videocall using 3G network available with Video Camera Sensor Board
- Record video (res. 320 x 240) and take pictures (res. 640 x 480) available with Video Camera Sensor Board
- Support microSD card up to 32GB
- 64MB of internal storage space
- Making/Receiving calls
- Making 'x' tone missed calls
- MS-assisted (A-GPS), MS-based (S-GPS) or Stand-alone GPS positioning
- Sending/Receiving SMS
- Single connection and multiple connections TCP/IP and UDP/IP clients
- TCP/IP server.
- HTTP and HTTPS service
- FTP and FTPS Service (downloading and uploading files)
- Sending/receiving email (SMTP/POP3)

Bluetooth low energy module

Protocol: Bluetooth v.4.0 / Bluetooth Smart
Chipset: BLE112
RX Sensitivity: -103dBm
TX Power: [-23dBm, +3dBm]
Antenna: 2dBi/5dBi antenna options
Security: AES-128
Range: 100 meters (at maximum TX power)

Figure: Bluetooth Low Energy module

Actions:

- Send broadcast advertisements (iBeacons)
- Connect to other BLE devices as Master / Slave
- Connect with Smartphones and Tablets
- Set automatic cycles sleep / transmission
- Calculate distance using RSSI values
- Perfect for indoor location networks (RTLS)
- Scan devices with maximum inquiry time
- Scan devices with maximum number of nodes
- Scan devices looking for a certain user by MAC address

Bluetooth module for device discovery

Protocol: Bluetooth 2.1 + EDR. Class 2
TX Power: 3dBm
Antenna: 2dBi
Max Scan: Up to 250 unique devices in each inquiry
Power levels: 7 [-27dBm, +3dBm]

Application:

- Vehicular and pedestrian traffic monitoring

Features:

- Received Strength Signal Indicator (RSSI) for each scanned device

Scan devices with maximum inquiry time

- Scan devices with maximum number of nodes
- Scan devices looking for a certain user by MAC address
- Class of Device (CoD) for each scanned device

Figure: Bluetooth module for device discovery

RFID/NFC

13.56MHz

- **Compatibility:** Reader/writer mode supporting ISO 14443A / MIFARE / FeliCaTM / NFCIP-1
- **Distance:** 5cm
- **Max capacity:** 4KB
- **Tags:** cards, keyrings, stickers

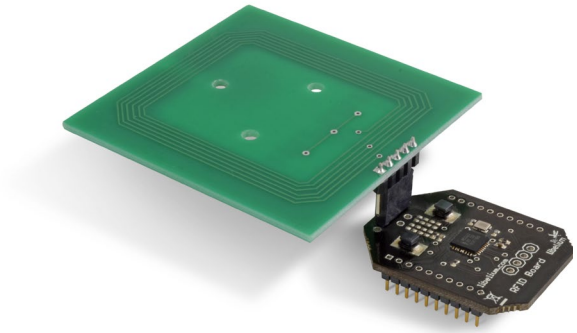


Figure: 13.56MHz RFID/NFC module

Applications:

- Located based services (LBS)
- Logistics (assets tracking, supply chain)
- Access management
- Electronic prepaid metering (vending machines, public transport)
- Smartphone interaction (NFCIP-1 protocol)

125KHz

- **Compatibility:** Reader/writer mode supporting ISO cards - T5557 / EM4102
- **Distance:** 5cm
- **Max capacity:** 20B
- **Tags available:** cards, keyrings

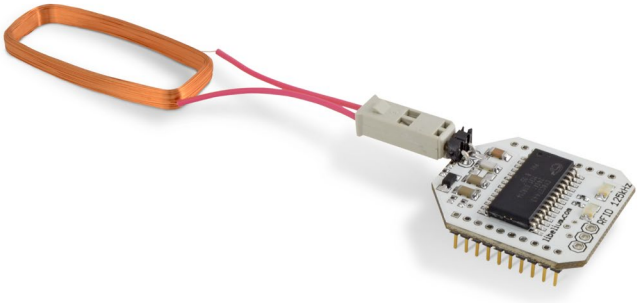


Figure: 125KHz RFID module

Applications:

- Located based services (LBS)
- Logistics (assets tracking, supply chain)
- Product management
- Animal farming identification

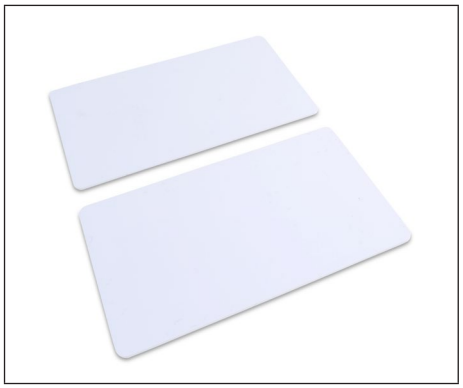


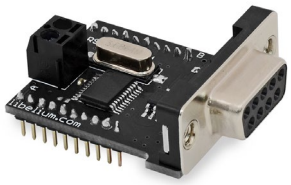
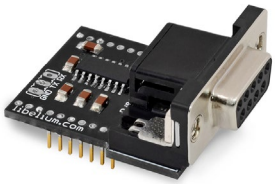
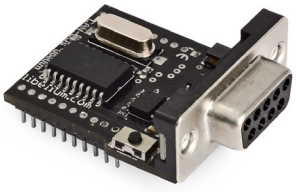
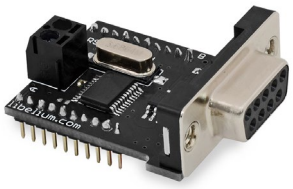
Figure: RFID cards

Figure: RFID keyrings

Figure: RFID sticker

Industrial Protocols

RS-485, RS-232, CAN Bus and Modbus are widely used standards in the industrial and automation market. Waspmote can be interfaced with standard devices and sensors thanks to the Industrial Protocols modules.

MODULE	MAIN APPLICATIONS	
RS-485 / Modbus module	• Industrial Equipment • Machine to Machine (M2M) communications • Industrial Control Systems, including the most common versions of Modbus and Profibus • Programmable Logic Controllers • RS-485 is also used in building automation • Interconnect security control panels and devices	 *Figure: RS-485 module*
RS-232 Serial / Modbus module	• Dial-up modems • GPS receivers (typically NMEA 0183 at 4,800 bit/s) • Bar code scanners and other point of sale devices • LED and LCD text displays • Satellite phones, low-speed satellite modems and other satellite based transceiver devices • Flat-screen (LCD and plasma) monitors to control screen functions by external computer, other AV components or remotes • Test and measuring equipment such as digital multimeters and weighing systems • Updating firmware on various consumer devices • Some CNC controllers • Uninterruptible power supply • Stenography or Stenotype machines • Software debuggers that run on a 2nd computer • Industrial field buses	 *Figure: RS-232 module*
CAN Bus module	• Automotive applications • Home automation • Industrial Networking • Factory automation • Marine electronics • Medical equipment • Military uses	 *Figure: Can Bus module*
Modbus software layer	• Modbus is a software layer which can be run over the RS-485 or RS-232 modules • Multiple master-slave applications • Sensors and instruments • Industrial Networking • Building and infrastructure • Transportation and energy applications	 *Figure: RS-485 module*

Expansion Radio Board

The Expansion Board allows to connect two communication modules at the same time in the Waspmote sensor platform. This means a lot of different combinations are possible using any of the wireless radios available for Waspmote: 802.15.4, ZigBee, DigiMesh, 868 MHz, 900 MHz, LoRa, Bluetooth Pro, Bluetooth Low Energy, RFID/NFC, WiFi, GPRS Pro, GPRS+GPS and 3G/GPRS. Besides, the following Industrial Protocols modules are available: RS-485/Modbus, RS-232 Serial/Modbus and CAN Bus.

Some of the possible combinations are:

- LoRa - GPRS
- 802.15.4 - Bluetooth
- 868 MHz - RS-485
- RS-232 - WiFi
- DigiMesh - 3G/GPRS
- RS-232 - RFID/NFC
- WiFi - 3G/GPRS
- CAN bus - Bluetooth
- etc.

Remark: *GPRS Pro, GPRS+GPS and 3G/GPRS modules do not need the Expansion Board to be connected to Waspmote. They can be plugged directly in the socket1.*

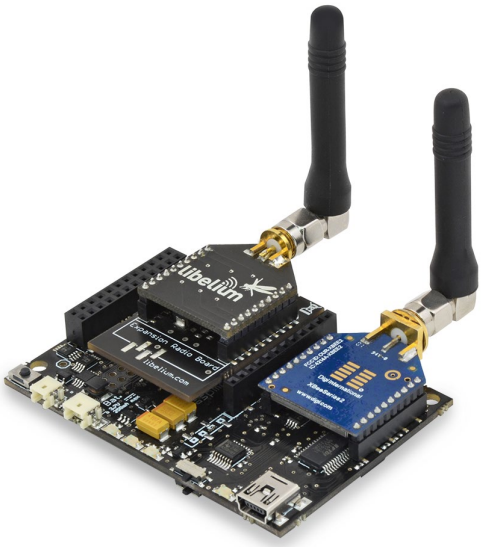


Figure: Expansion Radio Board

Applications:

- Multifrequency Sensor Networks: (2.4GHz - 868/900MHz)
- Bluetooth - ZigBee hybrid networks
- NFC (RFID) applications with 3G/GPRS
- ZigBee - WiFi hybrid networks

GPS

Model: JN3 (Telit)
Sensitivity :
- Acquisition: -147 dBm
- Navigation: -160 dBm
- Tracking: -163 dBm

Hot Start Time: <1s
Cold Start Time: <35s
Antenna connector: UFL
External antenna: 26dBi
Possitional accuracy error < 2.5 m
Speed accuracy < 0.01 m/s
EGNOS, WAAS, GAGAN and MSAS capability

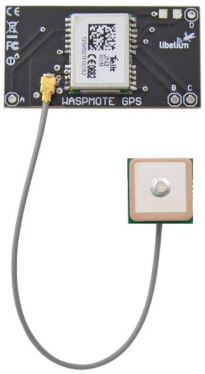


Figure: GPS

Available information: latitude, longitude, altitude, speed, direction, date/time and ephemerids management.

Programmable interruptions

- **Asynchronous**
  - Sensors (programmable threshold)
  - Accelerometer: Free-fall, impact (programmable threshold)
  - XBee (DigiMesh)
- **Synchronous:**
  - Watchdog: programmable alarms: from 32ms to 8s
  - RTC: programmable alarms: from 1s to days

Sensor Boards

GASES	APPLICATIONS	SENSORS
 *Figure: Gases Board*	• **City pollution** CO, CO_2, NO_2, O_3 • **Emissions from farms and hatcheries** CH_4 ,H_2S , NH_3 • **Control of chemical and industrial processes** C_4H_{10} ,H_2, VOC • **Forest fires** CO, CO_2	• Carbon Monoxide – CO • Carbon Dioxide – CO_2 • Oxygen – O_2 • Methane – CH_4 • Hydrogen – H_2 • Ammonia – NH_3 • Isobutane – C_4H_{10} • Ethanol – CH_3CH_2OH • Toluene – $C_6H_5CH_3$ • Hydrogen Sulfide – H_2S • Nitrogen Dioxide – NO_2 • Ozone – O_3 • Hydrocarbons – VOC • Temperature • Humidity • Atmospheric pressure

GASES PRO	APPLICATIONS	SENSORS
 *Figure: Gases PRO Board*	• **City pollution** CO, NO, NO_2, O_3, SO_2, Particle Matter - Dust • **Air Quality Index calculation** SO_2, NO_2, Particle Matter - Dust, CO, O_3, NH_3 • **Emissions from farms and hatcheries** CH_4, H_2S, NH_3 • **Greenhouse management** CO_2, CH_4, Humidity • **Control of chemical and industrial processes** H_2, HCl, CH_4, SO_2, CO_2 • **Indoor air quality** CO_2, CO, Particle Matter - Dust, O_3 • **Forest fires** CO, CO_2	• Carbon Monoxide – CO • Carbon Dioxide – CO_2 • Molecular Oxygen – O_2 • Ozone – O_3 • Nitric Oxide – NO • Nitric Dioxide – NO_2 • Sulfur Dioxide – SO_2 • Ammonia – NH_3 • Methane – CH_4 – and other combustible gases • Molecular Hydrogen – H_2 • Hydrogen Sulfide – H_2S • Hydrogen Chloride – HCl • Hydrogen Cyanide – HCN • Phosphine – PH_3 • Ethylene Oxide – ETO • Chlorine – Cl_2 • Particle Matter (PM1 / PM2.5 / PM10) – Dust Sensor [only for Plug & Sense!] • Temperature, Humidity and Pressure

EVENTS

Figure: Events Board

APPLICATIONS

- **Security**

  Hall effect (doors and windows), person detection PIR

- **Emergencies**

  Presence detection and water level sensors, temperature

- **Control of goods in logistics**

SENSORS

- Pressure/Weight
- Bend
- Hall Effect
- Temperature (+/-)
- Liquid Presence
- Liquid Flow
- Luminosity
- Presence (PIR)
- Stretch

SMART WATER

Figure: Smart Water Board

APPLICATIONS

- **Potable water monitoring**

  pH, ORP, Dissolved Oxygen (DO), Nitrates, Phosphates

- **Chemical leakage detection in rivers**

  Extreme pH values signal chemical spills , Dissolved Oxygen (DO)

- **Swimming pool remote measurement**

  pH, Oxidation-Reduction Potential (ORP)

- **Pollution levels in the sea**

  Temperature, Conductivity (Salinity), pH, Dissolved Oxygen (DO) and Nitrates

SENSORS

- pH
- Oxidation-Reduction Potential (ORP)
- Dissolved Oxygen (DO)
- Conductivity
- Temperature
- Turbidity

SMART WATER IONS

Figure: Smart Water Ions Board

APPLICATIONS

- **Drinking water quality control**

  Calcium (Ca^{2+}), Iodide (I^-), Chloride (Cl^-), Nitrate (NO_3^-), pH

- **Agriculture water monitoring**

  Calcium (Ca^{2+}), Nitrate (NO_3^-), pH

- **Swimming pools**

  Bromide (Br^-), Chloride (Cl^-), Fluoride (F^-), pH

- **Waste water treatment**

  Cupric (Cu^{2+}), Silver (Ag^+), Lead (Pb^{2+}), Fluoroborate (BF_4^-), pH

SENSORS

- Calcium (Ca^{2+})
- Fluoride (F^-)
- Fluoroborate (BF_4^-)
- Nitrate (NO_3^-)
- Bromide (Br^-)
- Chloride (Cl^-)
- Cupric (Cu^{2+})
- Iodide (I^-)
- Lead (Pb^{2+})
- Silver (Ag^+)
- pH
- Temperature

SMART CITIES

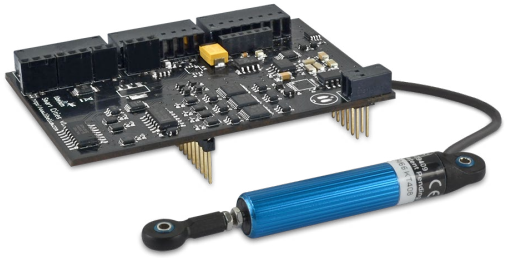


Figure: Smart Cities Board

APPLICATIONS

- **Noise maps**

  Monitor in real time the acoustic levels in the streets of a city

- **Structural health monitoring**

  Crack detection

- **Air quality**

  Detect the level of particulates and dust in the air

- **Waste management**

  Measure the garbage levels in bins to optimize the trash collection routes

SENSORS

- Microphone (dBA)
- Crack detection gauge
- Linear displacement
- Dust
- Ultrasound (distance measurement)
- Temperature
- Humidity
- Luminosity

SMART PARKING

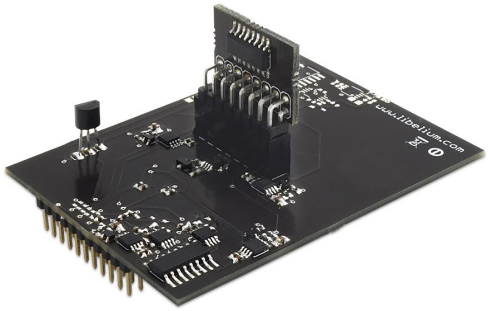


Figure: Smart Parking Board

APPLICATIONS

- Car detection for available parking information
- Detection of free parking lots outdoors
- Parallel and perpendicular parking slots control

SENSORS

- Magnetic Field
- Temperature

AGRICULTURE

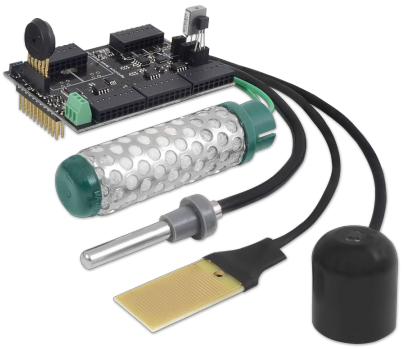


Figure: Agriculture Board

APPLICATIONS

- **Precision Agriculture**

  Leaf wetness, fruit diameter

- **Irrigation Systems**

  Soil moisture, leaf wetness

- **Greenhouses**

  Solar radiation, humidity, temperature

- **Weather Stations**

  Anemometer, wind vane, pluviometer

SENSORS

- Air Temperature / Humidity
- Soil Temperature / Moisture
- Leaf Wetness
- Atmospheric Pressure
- Solar Radiation - PAR
- Ultraviolet Radiation - UV
- Trunk Diameter
- Stem Diameter
- Fruit Diameter
- Anemometer
- Wind Vane
- Pluviometer
- Luminosity

4-20 mA CURRENT LOOP

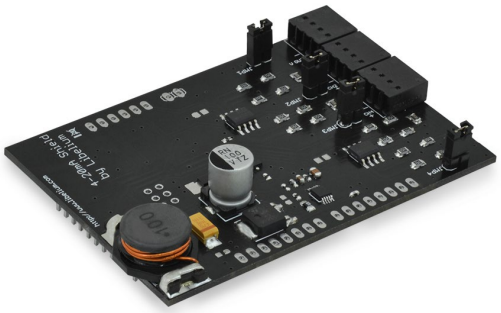


Figure: 4-20 mA Current Loop Board

APPLICATIONS

- Sensors and Instruments
- Remote transducers
- Monitoring processes
- Data transmission in industrial ambients

FEATURES

- **Type:** Analog
- **Media:** Twisted Pair
- **No. of devices:** 1
- **Distance:** 900m
- **Supply:** 5-24V

The user can choose among a wide variety of standard sensors

VIDEO CAMERA

Figure: Video Camera Sensor Board

APPLICATIONS

- Security and surveillance
- Take photos (640 x 380)
- Record video (320 x 240)
- Realtime Videocall using 3G network
- Night Vision mode available

SENSORS

- Image sensor
- Luminosity
- Infrared
- Presence (PIR)

RADIATION

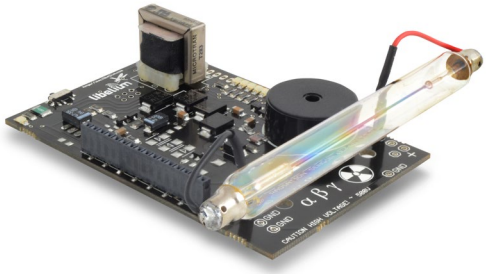


Figure: Radiation Board

APPLICATIONS

- Monitor the radiation levels wirelessly without compromising the life of the security forces
- Create prevention and control radiation networks in the surroundings of a nuclear plant
- Measure the amount of Beta and Gamma radiation in specific areas autonomously

SENSORS

- Geiger tube [β, γ] (Beta and Gamma)

SMART METERING	APPLICATIONS	SENSORS
	• Energy measurement • Water consumption • Pipe leakage detection • Liquid storage management • Tanks and silos level control • Supplies control in manufacturing • Industrial Automation • Agricultural Irrigation	• Current • Water flow • Liquid level • Load cell • Ultrasound • Distance Foil • Temperature • Humidity • Luminosity

Figure: Smart Metering Board

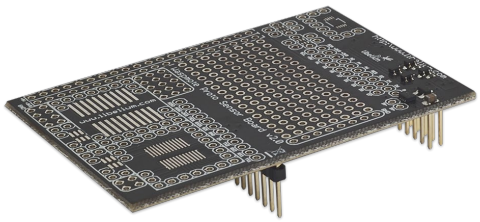
PROTOTYPING SENSOR	APPLICATIONS	SENSORS
	• Prepared for the **integration of any kind of sensor**.	• Pad Area • Integrated Circuit Area • Analog-to-Digital Converter (16b)

Figure: Prototyping Sensor Board

Power supplies

- 6600mAh Li-Ion rechargeable // 13000 /26000/52000mAh **non - rechargeable**
- Solar Panel: rigid (7V – 500mA) and flexible (7.2V – 100mA)
- USB (220V-USB, car lighter USB)

USB-PC interface

Model: Waspmote Gateway *
Communication: 802.15.4/ZigBee - USB PC
Programmable buttons and leds
** Included in the developers Kit*

Compiler:

- IDE-Waspmote (open source)
- Language: C++
- Versions Windows, Linux and Mac-OS

Figure: Waspmote Gateway

Waspmote vs Waspmote Plug & Sense!

Waspmote is the original line in which developers have a total control over the hardware device. You can physically access to the board and connect new sensors or even embed it in your own products as an electronic sensor device.

The new Waspmote Plug & Sense! line allows developers to forget about electronics and focus on services and applications. Now you can deploy wireless sensor networks in an easy and scalable way ensuring minimum maintenance costs. The new platform consists of a robust waterproof enclosure with specific external sockets to connect the sensors, the solar panel, the antenna and even the USB cable in order to reprogram the node. It has been specially designed to be scalable, easy to deploy and maintain.

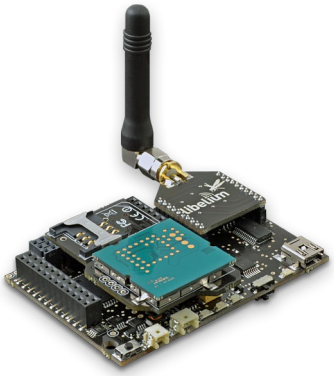


Figure: Waspmote

Figure: Waspmote Plug & Sense!

For more information about Waspmote Plug & Sense! go to:

http://www.libelium.com/plug_&_sense

Certifications

- CE (Europe)
- FCC (USA)
- IC (Canada)

Document version: v5.5 - 10/2015