

# **ESCUELA POLITÉCNICA NACIONAL**

## **FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA**

### **OPTIMIZACIÓN DEL FLUJO DE DATOS EN UN PROTOTIPO DE RED 6LowPAN CON TOPOLOGÍA LINEAL**

**PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN  
ELECTRÓNICA Y REDES DE INFORMACIÓN**

**LENIN XAVIER VELÁSTEGUI ALMEIDA**

lenin\_17x@yahoo.com

**DIRECTOR: ING. CARLOS EGAS, MSc.**

cegas@ieee.org

**Quito, Marzo 2016**

## DECLARACIÓN

Yo, Lenin Xavier Velástegui Almeida, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

---

Lenin Xavier Velástegui Almeida

## **CERTIFICACIÓN**

Certifico que el presente trabajo fue desarrollado por Lenin Xavier Velástegui Almeida, bajo mi supervisión.

---

**Ing. Carlos Egas, Msc.**  
**DIRECTOR DEL PROYECTO**

## **AGRADECIMIENTOS**

Mi agradecimiento es a mi madre por el apoyo incondicional durante mis estudios, a mi hermano por ayudarme en mis dificultades.

También un agradecimiento a la Escuela Politécnica Nacional y a mis maestros, por todas las enseñanzas compartidas.

Finalmente un agradecimiento especial al MSc. Carlos Egas, director del proyecto, por todo el tiempo dedicado, por la gestión realizada.



## **DEDICATORIA**

Este proyecto de titulación les dedico a mi madre y a mi hermano.

## CONTENIDO

DECLARACIÓN .....	I
CERTIFICACIÓN .....	II
AGRADECIMIENTOS .....	III
DEDICATORIA.....	IV
CONTENIDO.....	V
ÍNDICE DE FIGURAS .....	VIII
ÍNDICE DE TABLAS .....	X
ÍNDICE DE CÓDIGO.....	XI
RESUMEN .....	XII
PRESENTACIÓN .....	XIII
CAPÍTULO I .....	1
1 MARCO TEÓRICO .....	1
1.1 IEEE 802.15.4 .....	1
1.1.1 CAPA FÍSICA ( <i>PHYSICAL LAYER</i> , PHY) .....	4
1.1.2 IEEE 802.15.4 MAC .....	6
1.1.2.1 Estructura Supertrama .....	9
1.1.2.2 Otras características MAC .....	10
1.2 6LowPAN.....	11
1.2.1 IPv6 SOBRE IEEE 802.15.4.....	12
1.2.2 CAPA DE ADAPTACIÓN 6LOWPAN .....	14
1.2.3 DIRECCIONAMIENTO .....	16
1.2.4 FORMATO DE CABECERA.....	17
1.2.5 PROCESO DE INICIO.....	18
CAPÍTULO II .....	20
2 ALGORITMO DE OPTIMIZACIÓN DE FLUJO DE DATOS .....	20

2.1	TOPOLOGÍAS LINEALES CON WSN.....	20
2.2	WSN 6LowPAN .....	21
2.3	ALGORITMO .....	21
2.3.1	PRIMERA FASE.....	22
2.3.2	SEGUNDA FASE .....	23
2.4	PARÁMETROS DE COMPROBACIÓN.....	24
2.4.1	FUNCIONAMIENTO ÓPTIMO DE LAS 2 REDES LÓGICAS.....	24
2.4.2	FUNCIONAMIENTO DEL MÉTODO DE RECUPERACIÓN DE LA RED LÓGICA CON FALLA .....	25
CAPÍTULO III .....		26
3	IMPLEMENTACIÓN Y PRUEBAS DEL ALGORITMO.....	26
3.1	IMPLEMENTACIÓN DEL PROTOTIPO DE RED 6LowPAN .....	26
3.1.1	ANÁLISIS DE REQUERIMIENTOS PARA LA IMPLEMENTACIÓN DEL PROTOTIPO.....	26
3.1.1.1	Kit de desarrollo Wasmote Pro V1.2 .....	26
3.1.1.2	Ubuntu 12.04LTS .....	27
3.1.1.3	MonoDevelop .....	27
3.1.1.4	Mote Runner SDK.....	27
3.1.1.4.1	Características No Soportadas.....	27
3.1.1.4.2	Mote Runner Compiler .....	29
3.1.1.4.3	Mote Runner Shell.....	29
3.1.1.4.4	MRV6 .....	29
3.1.1.5	AVRdude 5.11.1-1.....	30
3.2	IMPLEMENTACIÓN Y CONFIGURACIÓN DEL PROTOTIPO DE RED 6LowPAN .....	30
3.2.1	INSTALACIÓN DEL FIRMWARE EN CADA NODO WASPMOTE PRO V1.2	30
3.2.2	SERVIDOR WEB MRSH ( <i>MOTE RUNNER SHELL</i> ).....	32

3.2.3	INSTALACIÓN DE LA LIBRERÍA MRV6 .....	33
3.2.3.1	Gateway .....	33
3.2.3.2	Nodo .....	34
3.2.4	CONFIGURACIÓN RED 6LOWPAN .....	35
3.3	IMPLEMENTACIÓN DEL ALGORITMO EN EL PROTOTIPO DE RED 6LowPAN .....	38
3.3.1	CÓDIGO DEL ALGORITMO .....	38
3.4	PRUEBAS Y RESULTADOS DEL ALGORITMO .....	44
CAPÍTULO IV .....		50
4	CONCLUSIONES Y RECOMENDACIONES .....	50
4.1	CONCLUSIONES .....	50
4.2	RECOMENDACIONES .....	51
REFERENCIAS BIBLIOGRÁFICAS .....		52
ANEXOS .....		53

## ÍNDICE DE FIGURAS

### CAPÍTULO I

Figura 1.1 Topología de una LR-WPAN.....	3
Figura 1.2 Estructura del paquete PHY IEEE 802.15.4 .....	6
Figura 1.3 Formato general de la trama MAC .....	9
Figura 1.4 Estructura de la supertrama IEEE 802.15.4 .....	10
Figura 1.5 Stack de Protocolos de IP y 6LowPAN .....	11
Figura 1.6 Stack de Protocolos en el nodo Gateway .....	12
Figura 1.7 Ejemplo de compresión de cabecera 6LowPAN (L = cabecera LowPAN) .....	18
Figura 1.8 Compresión de cabeceras 6LowPAN/UDP (6 bytes) .....	18
Figura 1.9 Cabeceras estándar IPv6/UDP (48 bytes) .....	19

### CAPÍTULO II

Figura 2.1 Ejemplo de topología lineal con WSN .....	20
Figura 2.2 Ejemplo de envío de un paquete en la topología lineal .....	20
Figura 2.3 Diagrama de Flujo Primera Fase.....	22
Figura 2.4 Diagrama de Flujo de la División de Red Lógica.....	25

### CAPÍTULO III

Figura 3.1 Prototipo de red 6LowPAN.....	26
Figura 3.2 Mote Runner Shell.....	29
Figura 3.3 Conectando el programador AVR para instalar Firmware.....	31
Figura 3.4 Conectando el programador AVR al nodo Waspnote Pro V1.2.....	31
Figura 3.5 Comando para establecer y quemar los fusibles.....	31
Figura 3.6 Comando para instalar el firmware en los nodos .....	32
Figura 3.7 Script para ejecutar el Servidor Web MRSH .....	32
Figura 3.8 Ejecución del script para poner a funcionar el Servidor Web MRSH... ..	32
Figura 3.9 Página inicial de Mote Runner .....	33
Figura 3.10 Comandos para la conexión USB con el nodo Gateway .....	33
Figura 3.11 Comando moma-list y cambio de directorio .....	34
Figura 3.12 Cargando librería mrv6-edge y configuración IPv4 nodo Gateway ...	34
Figura 3.13 Comandos para la conexión USB con el nodo .....	34
Figura 3.14 Cargando la librería mrv6-lib en el nodo .....	35
Figura 3.15 Directorio donde se encuentran los archivos del túnel .....	35

Figura 3.16 Archivo route_setup_linux_ipv6.sh.....	35
Figura 3.17 Configuración de la tarjeta Ethernet.....	36
Figura 3.18 Comandos para la configuración de la red 6LowPAN.....	36
Figura 3.19 Parámetros configurados en la red 6LowPAN.....	37
Figura 3.20 Dirección del Gateway sin asignar prefijo de red IPv6.....	37
Figura 3.21 Dirección del Gateway asignado prefijo de red IPv6.....	38
Figura 3.22 Prototipo de red 6LowPAN.....	45
Figura 3.23 Topología lineal con 2 sensores inalámbricos agregados a la red 6LowPAN.....	46
Figura 3.24 Visualización de los 2 nodos sensores agregados a la red 6LowPAN.....	46
Figura 3.25 Topología lineal con 4 sensores inalámbricos agregados a la red 6LowPAN.....	47
Figura 3.26 Visualización de los 4 nodos sensores agregados a la red 6LowPAN.....	47
Figura 3.27 Último funcionamiento de 4 nodos sensores en la WSN 6LowPAN..	48
Figura 3.28 Topología lineal con falla de 1 sensor inalámbrico en la red 6LowPAN.....	48
Figura 3.29 WSN 6LowPAN recuperada después del fallo.....	49

## ÍNDICE DE TABLAS

### **CAPÍTULO I**

Tabla 1.1 Canalización del estándar IEEE.802.15.4 .....	5
---	---

### **CAPÍTULO III**

Tabla 3.1 Cuadro Comparativo de Tiempo de Recuperación de la Red con y sin Algoritmo .....	49
--	----

## ÍNDICE DE CÓDIGO

### CAPÍTULO III

Código 3.1 Socket 1 para el envío de paquetes y recepción de confirmación .....	39
Código 3.2 Socket 2 para recepción de datos .....	40
Código 3.3 Socket 3 para recepción de datos .....	41
Código 3.4 Método para selección del puerto del siguiente salto .....	41
Código 3.5 Método para la división de red .....	42
Código 3.6 Método para envío de datos periódicos .....	44



## RESUMEN

El presente proyecto de titulación se encuentra dividido en cuatro capítulos. En el primer capítulo se realiza una explicación breve del estándar IEEE 802.15.4 y del estándar 6LowPAN (*IPv6 over Low power Wireless Personal Area Networks*).

En el segundo capítulo se hace una explicación de redes de sensores inalámbricos en topología lineal y sus problemas. Además se encuentra el algoritmo implementado en la red 6LowPAN y los parámetros de comprobación del funcionamiento del algoritmo en la red.

El tercer capítulo se enfoca en la creación y configuración de la red 6LowPAN y además se explica el código del algoritmo utilizado en la red. También se encuentran los resultados obtenidos de las pruebas de campo del algoritmo implementado.

En el cuarto capítulo se encuentra las conclusiones y recomendaciones obtenidas durante el proceso de implementación y la obtención del resultado del proyecto desarrollado.

Finalmente en los anexos está un manual sobre la instalación de requisitos previos para la implementación del prototipo de red 6LowPAN y se anexará un CD con el código del programa implementado.

## PRESENTACIÓN

Teniendo en cuenta el avance de las tecnologías de la información, la convergencia de las redes, la mejora en las prestaciones del internet y la amplia demanda de la sociedad a automatizar actividades cotidianas, por estos motivos se han desarrollado tecnologías orientadas al Internet de las Cosas (*Internet of Things*, IoT) las cuales se proyectan a tener conectividad total sobre las redes de sensores inalámbricos (*Wireless Network Sensor*, WSN).

Dada la popularidad y la tendencia de disminución de costos de tecnologías inalámbricas, en la actualidad se están desarrollando muchas aplicaciones de redes de sensores inalámbricos como por ejemplo: las redes WSN con topología lineal las cuales se pueden utilizar en el monitoreo de oleoductos, carreteras, fronteras, etc.

El incremento de las aplicaciones, se debe principalmente por la mejora en la calidad de servicio que las WSN están ofreciendo; pero en una topología lineal, al tener un solo camino para que los datos lleguen al nodo de borde (*Gateway*), el uso de tecnologías convencionales para dar calidad de servicio (*Quality of Service*, QoS) al flujo de datos no es aplicable; por lo que en este trabajo se proporciona una solución para mejorar la confiabilidad del flujo de datos en una red 6LowPAN con topología lineal, para lo cual se propone la implementación de dos redes lógicas con topología lineal, las cuales sirven de respaldo una con otra.

En la solución se utiliza las funciones de nivel de red y se realiza la configuración respectiva en cada nodo. Como resultado se obtiene que si una red lógica falla, automáticamente los datos son enviados por la otra red lógica.

# CAPÍTULO I

## 1 MARCO TEÓRICO

### 1.1 IEEE 802.15.4 [1]

Tras el desarrollo de la tecnología *Bluetooth*, se hizo evidente que ésta no era aplicable para todos los casos. Muchas aplicaciones industriales, agrícolas, médicas, y vehiculares, tales como: sensores, lectura de medidores, etiquetas o insignias inteligentes y automatización del hogar, requieren conectividad inalámbrica de corto alcance que es diferente de *Bluetooth*. Los dos parámetros más importantes para estas aplicaciones son: muy bajo consumo de energía y muy bajo costo. El consumo de energía debe ser bajo ya que las baterías pueden durar varios meses o más y la velocidad de los datos no es importante para estas aplicaciones, por lo que están dispuestos a negociar la velocidad para que la vida de la batería sea muy larga. *Bluetooth* o tecnología de redes inalámbricas de área personal de alta velocidad (*High Rate Wireless Personal Area Network*, HR-WPAN) no puede soportar estas aplicaciones, por lo que la IEEE 802 se dirigió a estas necesidades del mercado mediante el desarrollo de la tecnología de redes inalámbricas de área personal de baja velocidad (*Low Rate Wireless Personal Area Network*, LR-WPAN). El estándar fue llamado IEEE 802.15.4 y define las capas física y control de acceso al medio. Además de definir estas dos capas, la norma tiene como objetivo lograr un cierto nivel de convivencia con otros dispositivos inalámbricos.

En general los dispositivos LR-WPAN pueden ser fijos, portátiles o en movimiento, el principal objetivo del estándar IEEE 802.15.4 es realizar un protocolo simple, flexible y eficiente, para esto se decidió que una red IEEE 802.15.4 se componga de dos tipos de dispositivos diferentes: un dispositivo con funciones completas y un dispositivo con funciones reducidas.

El dispositivo con funciones completas puede operar en tres modos: un coordinador de la red (controlando la red), un coordinador, o un dispositivo de la red, por lo cual puede cambiarse dinámicamente su modo de operación. El dispositivo con funciones reducidas no puede controlar la red y puede ser

extremadamente simple, lo cual es apropiado para equipos tales como un conmutador de luz o un sensor pasivo infrarrojo. Este dispositivo no puede comunicarse con otros dispositivos con funciones reducidas por lo que solo pueden comunicarse con dispositivos con funciones completas. Dando como resultado, que el protocolo para dispositivos con funciones reducidas puede ser implementado usando mínimos recursos y mínima capacidad de memoria.

Los dispositivos LR-WPAN logran velocidades de datos de 250Kb/s, 40Kb/s, 20Kb/s y utilizan CSMA/CA (*Carrier Sense Multiple Access / Collision Advice*) para acceso al canal, además tienen administración de energía, y usan 16 canales en la banda ISM<sup>1</sup> de 2.4GHz.

Hay dos arquitecturas básicas de red para las LR-WPAN las cuales son: estrella y *peer-to-peer*<sup>2</sup>, como se muestra en la Figura 1.1. En la topología estrella los nodos de la red solo pueden comunicarse con el único controlador central por ejemplo el coordinador de la red. Un dispositivo de red tiene algunas funciones asociadas en las cuales puede ser el punto de inicio o el punto de terminación de la comunicación.

Un coordinador de la red puede ser usado para iniciar, terminar o enrutar una comunicación alrededor de la red, el cual no necesita un host PC. Una diferencia con otras tecnologías como por ejemplo Bluetooth es que mientras en estos los nodos son generalmente bidireccionales, en LR-WPAN los nodos unidireccionales son permitidos con acceso extremadamente limitado a la red. Los mensajes entre dos dispositivos de la red pueden ser intercambiados a través de enlaces virtuales *peer-to-peer*, donde las tramas de mensaje se enrutan a través del coordinador de la red. La topología estrella es apropiada para periféricos de PC y productos de automatización del hogar. Adicionalmente a los dos tipos básicos de red, se puede hacer arquitecturas de red más compleja, tales como las redes *cluster-tree*. Por lo tanto, IEEE 802.15.4 soporta los requisitos arquitectónicos de las redes de sensores inalámbricos.

---

<sup>1</sup> ISM (*Industrial, Scientific and Medical*): Son bandas reservadas internacionalmente para uso no comercial de radiofrecuencia electromagnética en áreas industrial, científica y médica.

<sup>2</sup> *peer-to-peer*: Es una red de computadoras en la que todos o algunos aspectos funcionan sin clientes ni servidores fijos, sino una serie de nodos que se comportan como iguales entre sí.

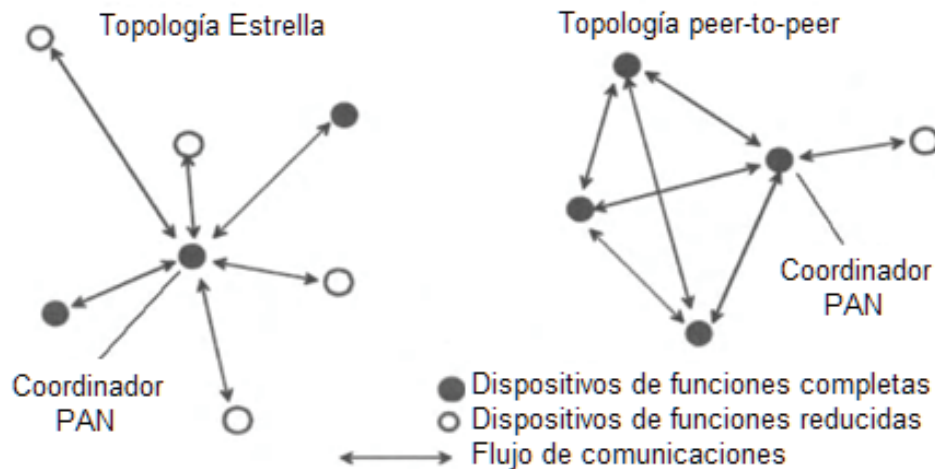


Figura 1.1 Topología de una LR-WPAN  
Fuente: [1]

En las topologías *peer-to-peer*, cualquier nodo puede comunicarse con cualquier otro nodo, siempre y cuando estén en el rango de uno al otro, por lo tanto, las topologías *peer-to-peer* requieren solo de dispositivos de funciones completas, por lo que en este modo son viables las topologías de red más complejas, tales como la topología *cluster-tree*<sup>3</sup>.

Las aplicaciones tales como: agricultura inteligente, monitoreo, control industrial, redes de sensores inalámbricos, seguimiento de activos e inventario, y seguridad se beneficiarían de dicha topología de red; su ventaja es que es *ad-hoc*, permite auto-organización y auto-recuperación, además permite multi-saltos para encaminar mensajes desde cualquier nodo en la red para cualquier otro nodo en la red. Mientras 802.15.4 permite ese tipo de redes, la formación y gestión de una red *cluster-tree* se realizan por una capa encima de la capa de control de acceso al medio (*Medium Access Control*, MAC) de los estándares 802.

Además, tanto las arquitecturas estrella y *peer-to-peer* se pueden conectar a otras redes. Por ejemplo, si el coordinador de la red de área personal (*Personal Area Network*, PAN) es una computadora, ésta puede estar conectada a otras redes inalámbricas o cableadas.

<sup>3</sup> *Cluster-tree*: Esta denominación corresponde a un híbrido que se forma aplicando una estructura de árbol a varias estrellas. Cada estrella es un *cluster* (grupo), y un *cluster tree* es un árbol de *clusters*.

El estándar LR-WPAN consiste de una capa MAC y una capa física, por lo cual un dispositivo LR-WPAN se compone de una capa física, que contiene el transceptor de radio frecuencia y cualquier control de bajo nivel del transceptor, y una subcapa de control de acceso al medio.

El consumo de energía es de suma importancia en esta tecnología, debido a que, en muchas aplicaciones que utilizan IEEE 802.15.4, los dispositivos se alimentan de la batería, por ende el consumo de energía reducido se consigue de dos maneras:

- En primer lugar, el protocolo IEEE 802.15.4 se desarrolla de modo que los dispositivos requieren poca energía para funcionar.
- En segundo lugar, en la implementación física de IEEE 802.15.4, se pueden utilizar técnicas adicionales de administración de energía.

Las técnicas adicionales de administración de energía son un área para la diferenciación de proveedores en el mercado, por lo tanto hay que tener en cuenta que algunos dispositivos podrían ser alimentados por una red eléctrica. Los nodos que funcionan con baterías normalmente utilizarán un ciclo de trabajo para reducir el consumo de energía, de hecho, estos dispositivos pasan la mayor parte de su vida útil en un estado de sueño, eso quiere decir que no participan en la red. Sin embargo, cada dispositivo de red debe escuchar periódicamente al canal de radio frecuencia para detectar el *beacon*<sup>4</sup> de red con el fin de determinar si un mensaje está pendiente desde un coordinador de la red u otro dispositivo, y para permanecer sincronizado con la red. Este mecanismo permite que el diseñador de la aplicación decida sobre el equilibrio entre el consumo de batería y la latencia de retardo del mensaje. Los dispositivos LR-WPAN alimentados por la red eléctrica tienen la opción de escuchar el canal de radio frecuencia de forma continua.

### 1.1.1 CAPA FÍSICA (*PHYSICAL LAYER, PHY*) [1],[2]

Las características de la capa física son la activación y desactivación del transceptor de radio frecuencia, ajuste de canal, y la evaluación de un canal libre.

---

<sup>4</sup> *Beacon*: Es una trama que le permite al coordinador manejar las comunicaciones en la LR-WPAN, este puede estar habilitado o deshabilitado en estas redes.

Para lograr el bajo costo, los dispositivos operan en las bandas de frecuencias 2.4GHz, 868MHz y 915 MHz, como se indican en la Tabla 1.1, con un total de 27 canales, numerados 0-26 y que están disponibles en estas tres bandas de frecuencia.

<b>Banda</b>	<b>Velocidad de bits</b>	<b>Número de canales</b>	<b>Localización</b>
868-868.6 MHz	20 Kb/s	1	Europa
902-928 MHz	40 Kb/s	10	América
2400-2483.5 MHz	250 Kb/s	16	Mundial

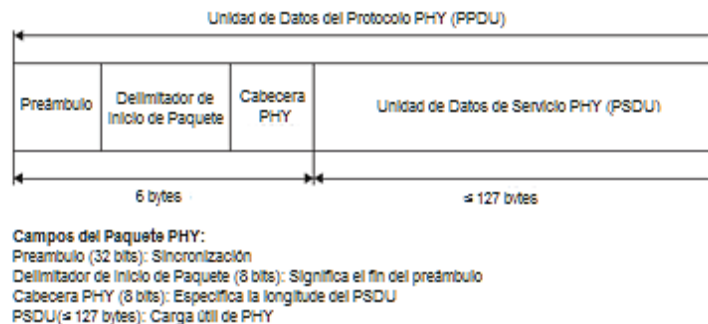
**Tabla 1.1 Canalización del estándar IEEE.802.15.4**  
Fuente: [2]

La capa física debe ser capaz de realizar la evaluación de un canal libre, para lo cual hay tres maneras de hacerlo: umbral de energía, detección de la señal IEEE 802.15.4, y una combinación de ambos (IEEE 802.15.4 de detección de señal con energía por encima de un umbral).

Las sensibilidades del receptor son: -85dBm para 2.4GHz, -92dBm para 868 y 915 MHz. El rango de alcance es una función de la sensibilidad del receptor, así como de la potencia de transmisión; por lo que el estándar especifica que cada dispositivo debe ser capaz de transmitir 1mW. Dependiendo de las necesidades de la aplicación, la potencia de transmisión real puede ser inferior (dentro de los límites regulatorios), por lo tanto se espera que los dispositivos típicos (1mW) puedan cubrir un rango de 10 a 20 m, con buena sensibilidad y un aumento moderado de la potencia de transmisión.

Para mantener una sencilla interfaz común con la subcapa de control de acceso al medio, se comparte una sola estructura de paquetes como se indica en la Figura 1.2. Cada unidad de datos del protocolo PHY (*PHY Protocol Data Unit*, PPDU) contiene una cabecera de sincronización (inicio más preámbulo de delimitador de paquetes) y una cabecera de la capa física para indicar la longitud del paquete, y la carga útil o unidad de datos de servicio de la capa física (*PHY Service Data Unit*, PSDU). El preámbulo de 32 bits está diseñado para la adquisición de símbolo y chip de sincronización y en algunos casos se puede utilizar para el

ajuste de frecuencia. La ecualización de canal no es necesaria para cualquier capa física debido a la combinación del área de cobertura pequeña y las relativamente bajas tasas de chips. Dentro de la cabecera de la capa física, se utilizan 7 bits para especificar la longitud de la carga útil (en bytes), por ende soporta paquetes de longitud 0-127 bytes, aunque, debido a la sobrecarga de la subcapa de control de acceso al medio, los paquetes de longitud cero no ocurrirán en la práctica.



**Figura 1.2 Estructura del paquete PHY IEEE 802.15.4**  
 Fuente: [2]

### 1.1.2 IEEE 802.15.4 MAC [1],[2]

Las principales características de la subcapa MAC son la gestión por *beacon*, mecanismo de acceso al canal, selección dinámica de canales, tramas de recepción y acuse de recibo, asociación, disociación y seguridad. Las funciones de seguridad son necesarias en IEEE 802.15.4 para tener éxito en el mercado, por lo tanto, son compatibles con las listas de control de acceso y criptografía simétrica.

Las LR-WPAN usan dos tipos de acceso al canal, dependiendo de la configuración de la red. Las redes sin *beacon* usan CSMA/CA como mecanismo de acceso al canal, este mecanismo es muy similar al mecanismo en el estándar IEEE 802.11.

Las redes con *beacon* habilitado utilizan el protocolo de acceso al canal CSMA/CA para acceder a una ranura de tiempo en el cual pueden transmitir. El mecanismo de ranura CSMA/CA funciona de la siguiente manera: una estación que quiere transmitir detecta el canal, y si el canal está desocupado va a esperar por un número aleatorio de ranuras y empezará a transmitir en el siguiente límite



de ranura disponible; las ranuras de *backoff*<sup>5</sup> están alineadas con el inicio del *beacon*. El *beacon* en sí y el acuse de recibo son enviados sin CSMA/CA.

Hay tres tipos de transferencia de datos: El primero es desde un dispositivo de funciones reducidas a un dispositivo de funciones completas. La segunda desde un dispositivo de funciones completas a un dispositivo de funciones reducidas. El tercero desde un dispositivo de funciones completas a un dispositivo de funciones completas. El primer tipo de transferencia es desde un dispositivo al coordinador de la red, el segundo tipo de transferencia es desde el coordinador a un dispositivo de la red, y el tercer tipo de transferencia es desde un coordinador a otro coordinador en la red. El mecanismo para cada tipo de transferencia depende si en la red está habilitado el *beacon* o no.

Cuando un dispositivo necesita transferir datos al coordinador en una red con *beacon* habilitado, este dispositivo primero escucha el *beacon*, que es necesario para que el dispositivo pueda determinar cuándo puede transmitir.

En cambio, en una red con *beacon* deshabilitado, el dispositivo usa directamente el protocolo CSMA (*Carrier Sense Multiple Access*) para transmitir sus datos al coordinador. Por último, el coordinador puede reconocer los dos tipos de transmisión como exitosa. Una transmisión exitosa no necesariamente implica un mensaje de acuse de recibo.

El coordinador de la red puede enviar datos a un dispositivo de red en una de las siguientes dos maneras: En una red con *beacon* habilitado, el coordinador indica los datos pendientes para un dispositivo en su *beacon*; el dispositivo solicita los datos y luego el coordinador envía el paquete de datos adecuado, y el dispositivo reconoce transmisión exitosa. En una red de *beacon* deshabilitado, el coordinador debe mantener los datos hasta que el dispositivo adecuado hace contacto y solicita los datos.

Un dispositivo de red puede contactar con el coordinador de la red por dos formas:

---

<sup>5</sup> *Backoff*: es un algoritmo utilizado para la contención en el uso de una red, es utilizado cuando se necesita una retransmisión por colisiones.

- La primera, el dispositivo puede sondear al coordinador a una velocidad específica; y si hay datos pendientes, el coordinador transmite los datos.
- La segunda forma es la transferencia de datos desde el dispositivo al coordinador, y a continuación el coordinador de la red puede indicar en su acuse de recibo de que tiene datos para el dispositivo de red.

La transferencia de datos *peer-to-peer* sigue el esquema CSMA.

Las estructuras de los paquetes han sido diseñadas para que su complejidad sea mínima, y a su vez los hace lo suficientemente robustos para transmitir en un canal ruidoso; el formato general de una trama de MAC se muestra en la Figura 1.3.

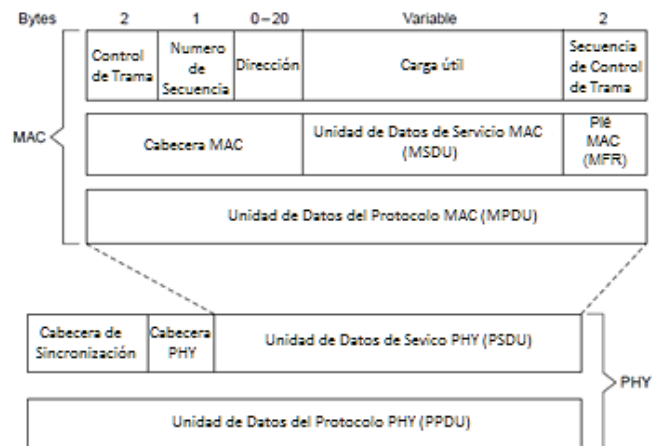
La unidad de datos de protocolo MAC (*MAC Protocol Data Unit*, MPDU) consiste en la cabecera MAC (*MAC Header*, MHR), la unidad de datos de servicio MAC (*MAC Service Data Unit*, MSDU), y el pie MAC (*MAC Footer*, MFR). El primer campo de la cabecera MAC es el campo de control de trama, que indica el tipo de trama MAC que está transmitiendo, especifica el formato del campo de dirección, y controla el acuse de recibo.

El campo de control de trama especifica cómo el resto de la trama se ve y lo que contiene. El tamaño del campo de dirección puede variar entre 0 y 20 bytes, por lo que la estructura flexible del campo de dirección ayuda a aumentar la eficiencia del protocolo, manteniendo los paquetes cortos.

El campo de carga útil es de longitud variable; sin embargo, la trama MAC completa no podrá exceder de 127 bytes de longitud y los datos contenidos en la carga útil dependen del tipo de trama.

El IEEE 802.15.4 MAC tiene cuatro tipos de tramas diferentes y estas son: la trama *beacon*, trama de datos, trama de acuse de recibo y trama de comando MAC; donde sólo las tramas de datos y de *beacon* en realidad contienen información enviada por las capas superiores.

Las tramas de acuse de recibo y de comando MAC se originan en la MAC y se utilizan para comunicación MAC en *peer-to-peer*.



**Figura 1.3 Formato general de la trama MAC**  
Fuente: [2]

Otros campos en la trama MAC son el número de secuencia y la secuencia de chequeo de trama (*Frame Check Sequence*, FCS). El número de secuencia en el encabezado MAC coincide con la trama de acuse de recibo de la transmisión anterior. El FCS ayuda a verificar la integridad de la trama MAC.

### 1.1.2.1 Estructura Supertrama [2]

Algunas aplicaciones pueden requerir un ancho de banda dedicado para lograr bajas latencias; para lograr este objetivo, el IEEE 802.15.4 LR-WPAN puede operar en un modo opcional de supertrama. En una supertrama (ver Figura 1.4), un coordinador PAN dedicado transmite *beacons* de supertramas en intervalos predeterminados; estos intervalos pueden ser tan cortos como 15 ms o tan largos como 245 segundos. El tiempo entre dos *beacons* se divide en 16 ranuras de tiempo iguales independientes de la duración de la supertrama. La trama *beacon* se envía en la primera ranura de cada supertrama. Las *beacons* se utilizan para sincronizar los dispositivos conectados, para identificar la PAN, y describir la estructura de las supertramas. Un dispositivo puede transmitir en cualquier momento durante la ranura, pero debe completar su transacción antes de la próxima *beacon* de la supertrama. El acceso al canal en intervalos de tiempo se basa en contención; sin embargo, el coordinador de la PAN podrá ceder espacios de tiempo a un único dispositivo que requiere un ancho de banda dedicado o transmisiones de baja latencia. Estas ranuras de tiempo asignadas se llaman ranuras garantizadas de tiempo (*Guaranteed Time Slot*, GTS) y juntos forman un

periodo libre de contención (*Contention Free Period*, CFP) situado inmediatamente antes de la siguiente *beacon*. El tamaño de la CFP puede variar dependiendo de la demanda de los dispositivos asociados a la red. Cuando se utilizan intervalos de tiempo garantizados, todos los dispositivos deben completar sus transacciones basadas en contención antes de que comience la CFP. El comienzo de la CFP y la duración de la supertrama se comunican a los dispositivos conectados a la red por el coordinador de la PAN. El coordinador de la PAN puede asignar hasta siete de los GTS y un GTS puede ocupar más de una ranura de tiempo.

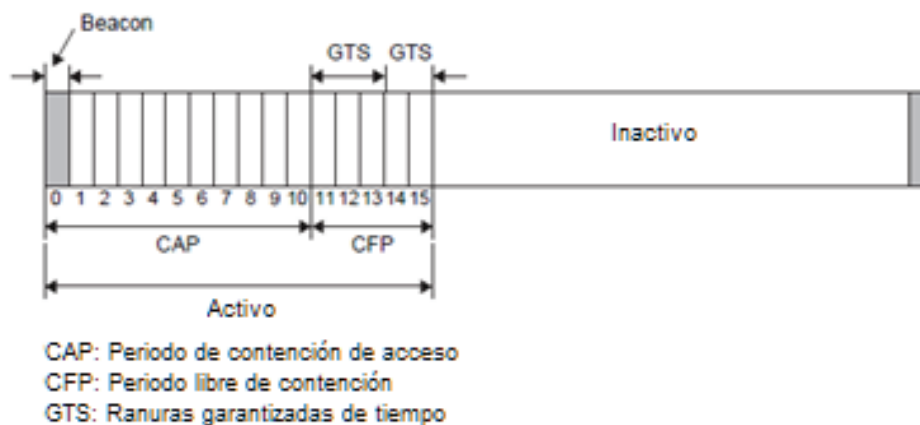


Figura 1.4 Estructura de la supertrama IEEE 802.15.4  
Fuente: [2]

### 1.1.2.2 Otras características MAC [2]

Una función importante de la MAC es confirmar la recepción satisfactoria de una trama recibida. Una exitosa recepción y validación de un dato o una trama de comando MAC son confirmados con un acuse de recibo. Si el dispositivo receptor no es capaz de manejar el mensaje entrante por cualquier motivo, la recepción no hace un acuse de recibo.

El campo de control de la trama MAC indica si se espera un acuse de recibo; la trama de acuse de recibo se envía inmediatamente después de la validación exitosa de la trama recibida.

La MAC proporciona tres niveles de seguridad y son: no hay seguridad de ningún tipo; listas de control de acceso (seguridad no criptográfica); y la seguridad de clave simétrica utilizando AES-128.

## 1.2 6LowPAN [3]

El *stack* de protocolos de IPv6 con 6LowPAN en comparación con el *stack* de protocolos de IP y con sus correspondientes capas dentro de la arquitectura TCP/IP<sup>6</sup>, separando la capa física de la capa de enlace son casi idénticos como se indica en la Figura 1.5, pero con las siguientes diferencias. La primera diferencia es que 6LowPAN solo soporta IPv6, mediante una capa de adaptación llamada LoWPAN la cual ha sido definida para el uso de 6LowPAN sobre IEEE 802.15.4. El protocolo de capa transporte más utilizado por 6LowPAN es el protocolo UDP<sup>7</sup>, el cual también es comprimido usando el formato 6LowPAN; el protocolo TCP<sup>8</sup> no es utilizado, ya que degrada el desempeño, la eficiencia de la red y por razones de complejidad. El protocolo ICMPv6 es utilizado para mensajes de control.

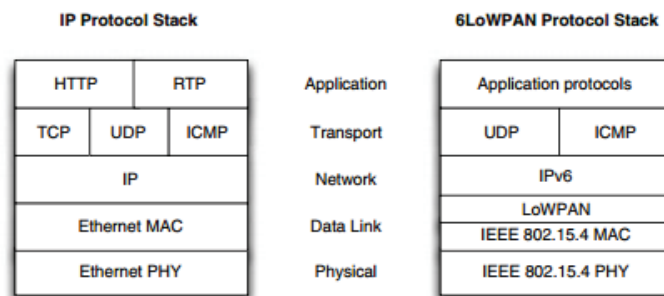


Figura 1.5 Stack de Protocolos de IP y 6LowPAN  
Fuente: [3]

En el nodo *Gateway* (o Coordinador) tendremos a los dos *stack* de protocolos tanto de 6LowPAN como de IP como se muestra en la Figura 1.6. Para las redes 6LowPAN se han definido mecanismos para la conexión a nivel de enlace y en especial se ha utilizado el estándar IEEE 802.15.4 en el cual se desarrolló esta tecnología.

<sup>6</sup> TCP/IP (*Transport Control Protocol/ Internet Protocol*): Es una arquitectura que está compuesta por cuatro capas o niveles, cada nivel se encarga de determinados aspectos de la comunicación y a su vez brinda un servicio específico a la capa superior. Estas capas son: aplicación, transporte, internet, acceso al canal.

<sup>7</sup> UDP (*User Datagram Protocol*): Es un protocolo no orientado a conexión de la capa de transporte del modelo TCP/IP. Este protocolo es muy simple ya que no proporciona detección de errores.

<sup>8</sup> TCP (*Transport Control Protocol*): Es un protocolo orientado a conexión, es decir, que permite que dos máquinas que están comunicadas controlen el estado de la transmisión.

IPv6	
Ethernet MAC	LoWPAN adaptation
	IEEE 802.15.4 MAC
Ethernet PHY	IEEE 802.15.4 PHY

Figura 1.6 Stack de Protocolos en el nodo Gateway  
Fuente: [3]

### 1.2.1 IPv6 SOBRE IEEE 802.15.4 [4]

El protocolo IPv6 está diseñado como el sucesor de IPv4 y permite el crecimiento del Internet para las próximas décadas ya que ante la escases de direcciones IPv4 y en previsión de que los aparatos e instrumentos en red serán mucho más numerosos que los ordenadores convencionales, IPv6 amplía el espacio de direcciones IP de 32 a 128 bits reconociendo el crecimiento del ancho de banda del enlace; además IPv6 aumenta el requisito mínimo de MTU<sup>9</sup> de 576 a 1280 bytes. Para simplificar los routers<sup>10</sup> y aumentar el rendimiento, IPv6 implementa la fragmentación en los puntos finales, en lugar de que se haga en los routers intermedios.

Para aumentar la eficiencia del protocolo y eliminar la necesidad de servicios *ad hoc* a nivel de enlace en el proceso de inicio de una subred, IPv6 incluye el alcance *multicast*<sup>11</sup> como parte integral de su arquitectura.

Los componentes del núcleo de IPv6 tales como el descubrimiento de vecinos (*Neighbor Discovery*, ND), utilizan *multicast* en el alcance de enlace local para la resolución de direcciones, detección de direcciones duplicadas, y el descubrimiento de enrutadores. La autoconfiguración de direcciones *stateless*<sup>12</sup>

<sup>9</sup> MTU (*Maximum Transmission Unit*): Expresa el tamaño en bytes de la unidad de datos más grande que puede enviarse usando un protocolo de comunicaciones.

<sup>10</sup> Router: es un dispositivo de interconexión de redes informáticas que permite asegurar el enrutamiento de paquetes entre redes o determinar la ruta que debe tomar el paquete de datos.

<sup>11</sup> *Multicast*: Es el envío de la información en múltiples redes o a múltiples destinos simultáneamente.

<sup>12</sup> *Stateless*: El host utiliza el prefijo (la dirección de red y la máscara de subred), el cual es "publicado" por los dispositivos de red, como parte de la creación de la dirección. Posteriormente los clientes pueden utilizar su dirección física (MAC *address*) para completar la identificación del dispositivo y así asignar la dirección IPv6.

(*Stateless Address Autoconfiguration, SAA*), simplifica la configuración y gestión de dispositivos IPv6 al permitir a los nodos asignarse a sí mismos direcciones significativas. IPv6 también refleja los avances en tecnologías de enlace utilizados en el Internet; por lo que Ethernet se ha impuesto como el enlace dominante, y su rendimiento se ha incrementado a un ritmo extraordinario.

Las tecnologías actuales WLAN (*Wireless Local Area Network*), tales como Wi-Fi tienen las mismas capacidades de Ethernet por medio del soporte de similares tamaños de MTU y de altas tasas de velocidad en los enlaces. Estos enlaces operan en el contexto de una gran potencia y dispositivos de alta capacidad. Las tecnologías WPAN, por otro lado, operan con menor potencia; por lo que IEEE 802.15.4 fue diseñado específicamente para ámbitos de aplicaciones de larga duración que requieren numerosos nodos de bajo costo, y estas restricciones limitan la capacidad de los enlaces LowPAN y los microcontroladores a los que están unidos. El *throughput*<sup>13</sup> está limitado a 250kbps. La longitud de la trama está limitada a 128 bytes para asegurar razonablemente bajas tasas de error de paquetes, pero cuando las tasas de errores de bits no son insignificantes refleja la limitada funcionalidad de los buffers de los microcontroladores, esto hace que IEEE 802.15.4 defina direcciones cortas de enlace de 16 bits, además de direcciones IEEE EUI-64, para reducir la sobrecarga de cabecera y los requisitos de memoria.

El rango de comunicación es corto (decenas de metros), ya que aumenta la potencia de transmisión con el rango. A diferencia de la mayoría de las instalaciones típicas WPAN y WLAN, las LowPAN se comunican a través de múltiples saltos; y por último, los microcontroladores asociados suelen tener alrededor de 8 Kbytes de memoria de acceso aleatorio (*Random Access Memory, RAM*) de datos y 64 Kbytes de memoria de solo lectura (*Read Only Memory, ROM*) del programa.

Debido a estas limitaciones de recursos y la naturaleza multisalto de las LowPAN, el soporte de IPv6 sobre redes LowPAN presenta varios desafíos los cuales son:

---

<sup>13</sup> *Throughput*: Es la tasa promedio de éxito en la entrega de un mensaje sobre un canal de comunicación.

- Los datagramas IPv6 no son aptos para LowPAN, ya que LowPAN tiene bajo *throughput*, limitado *buffering*<sup>14</sup> y las tramas son una décima parte del tamaño mínimo de la MTU de IPv6, por lo tanto para una operación eficiente es necesario hacer fragmentación y compresión de datagramas.
- IEEE 802.15.4 debido a su bajo consumo y bajo rendimiento, es más propenso a la interferencia espuria, fallos de enlace, cualidades dinámicas del enlace y a los enlaces asimétricos; estas características requieren que la capa de red sea sensible y adaptable sin dejar de ser energéticamente eficiente, ya que afectan a todos los aspectos de la creación de redes, incluyendo la fragmentación, la compresión, el reenvío y el enrutamiento.
- La topología esperada en una LowPAN es una malla de conexiones de corto alcance, por lo que niega el supuesto de que el enlace es un único dominio de *broadcast* sobre el cual un núcleo de componentes arquitectónicos IP tales como: IPv6 ND y SAA se cimientan.

### 1.2.2 CAPA DE ADAPTACIÓN 6LOWPAN [4]

El formato 6LowPAN define como la comunicación IPv6 es llevada en tramas IEEE 802.15.4 y especifica los elementos claves de la capa de adaptación. 6LowPAN tiene tres elementos primarios:

- Compresión de cabecera: Los campos de la cabecera IPv6 son comprimidos asumiendo el uso de valores comunes, por lo que los campos de cabecera son omitidos de un paquete cuando la capa de adaptación puede derivarlos de la información a nivel de enlace que lleva la trama IEEE 802.15.4 o basados en suposiciones simples de contexto compartido.
- Fragmentación: Los paquetes IPv6 son fragmentados en varias tramas de nivel de enlace para dar cabida a la exigencia mínima de MTU de IPv6.
- Reenvío en capa dos: Para soportar reenvío de datagramas IPv6 en capa dos, la capa de adaptación puede llevar direcciones de nivel de enlace por los extremos de un salto IP. Alternativamente, el *stack* IP podría lograr que la *intra-PAN (Intra Personal Area Network)* haga el enrutamiento a través de la capa

---

<sup>14</sup> *Buffering*: Es un espacio de la memoria reservado para el almacenamiento temporal de información digital, mientras que está esperando ser procesada.



tres para el reenvío de información, en el que cada salto de radio IEEE 802.15.4 es un salto IP.

El concepto clave aplicado por toda la capa de adaptación 6LowPAN es el uso de *stateless* o la compresión de contexto compartido para omitir los campos de las cabeceras de las siguientes capas: capa transporte, capa de red y capa de adaptación, comprimiendo las tres capas hasta unos pocos bytes, combinados. Es posible comprimir los campos de cabecera a unos pocos bits cuando se observa que a menudo llevan valores comunes, reservando un valor de escape para cuando aparecen los menos comunes. Los valores comunes se producen debido al uso frecuente de un subconjunto de la funcionalidad de IPv6 (tales como UDP, TCP, y ICMPv6 como valores de siguiente cabecera (*Next Header*)) y las hipótesis simples de contexto compartido (por ejemplo, un prefijo de red común asignado a la totalidad de la LowPAN). 6LoWPAN también omite la información de cabecera redundante a través de las capas de protocolo (por ejemplo, campos de longitud UDP e IPv6 y las direcciones IPv6 se derivan de las cabeceras de capa inferior).

Las técnicas de compresión de encabezado IP tradicionales son *statefull*<sup>15</sup> y generalmente se centran en la optimización de los flujos individuales sobre un enlace muy limitado. Estos métodos suponen que el compresor y descompresor están en comunicación directa y exclusiva ya que comprimen las cabeceras tanto de la capa de red y de capa transporte; por lo que se optimizan para flujos de larga vida mediante la explotación de las redundancias a través de los paquetes dentro de un flujo a través del tiempo, requiriendo de los puntos finales para enviar inicialmente paquetes sin comprimir. Las técnicas de compresión basadas en flujos son poco adecuadas para las LowPAN, ya que el tráfico en muchas aplicaciones LowPAN es impulsado por las lecturas poco frecuentes o notificaciones, en lugar de los flujos de larga vida; debido a esto la comunicación a través de múltiples saltos requiere compresión salto a salto y la descompresión y el estado de cada flujo en cada nodo intermedio. Muchos protocolos de

---

<sup>15</sup> *Statefull*: Es la asignación de la dirección y sus parámetros correspondientes, siempre y cuando sean configurados con antelación, por medio de un servicio de asignación dinámico, mejor conocido como DHCPv6.

enrutamiento LowPAN obtienen diversidad de receptor a través de cambios de ruta, lo que requeriría la migración del estado del flujo de datos y la reducción de la eficacia de la compresión. En contraste, la compresión *stateless* y de contexto compartido en 6LowPAN no requiere ningún estado por flujo y permite que los protocolos de enrutamiento elijan dinámicamente rutas sin afectar la eficiencia de compresión. En cuanto a aspectos específicos de 6LowPAN, la compresión *stateless* está ampliamente difundida.

### 1.2.3 DIRECCIONAMIENTO [3]

El direccionamiento IP con 6LoWPAN funciona igual que en cualquier red IPv6, y es similar al direccionamiento sobre redes Ethernet; ya que las direcciones IPv6 se forman típicamente de manera automática desde el prefijo del LowPAN y la dirección de capa de enlace de las interfaces inalámbricas; pero la diferencia en una LowPAN es con la forma en que las tecnologías inalámbricas de bajo consumo soportan el direccionamiento de capa de enlace; por lo que se utiliza una asignación directa entre la dirección de capa de enlace y la dirección IPv6 para lograr la compresión.

Los radio enlaces inalámbricos de bajo consumo normalmente hacen uso del direccionamiento a nivel de capa de enlace para todos los dispositivos y para soportar ambas direcciones: las direcciones únicas largas (por ejemplo EUI-64) y las direcciones cortas configurables (generalmente 8-16 bits de longitud). El estándar IEEE 802.15.4, por ejemplo, soporta direcciones únicas EUI-64 llevados en sus chips de radio, junto con las direcciones cortas configurables de 16 bits. Estas redes, por naturaleza, también soportan *broadcast*<sup>16</sup> (dirección 0xFFFF en IEEE 802.15.4), pero no soportan *multicast* nativo.

Las direcciones IPv6 son de 128 bits de longitud, y una parte se compone de un prefijo de 64 bits y un identificador de interfaz de 64 bits (IID). La configuración automática de direcciones *stateless* (SAA) se utiliza para formar el identificador de interfaz IPv6 de la dirección de capa de enlace de la interfaz inalámbrica ya que,

---

<sup>16</sup> *Broadcast*: Es una forma de transmisión de información donde un nodo emisor envía información a todos los nodos receptores de manera simultánea, sin necesidad de reproducir la misma transmisión nodo por nodo.

por simplicidad y compresión, las redes 6LowPAN asumen que el IID tiene una asignación directa a la dirección de capa de enlace, por lo tanto, evitando la necesidad de resolución de direcciones. El prefijo IPv6 es adquirido a través de mensajes de descubrimiento de vecinos por anuncio de rutas como un enlace IPv6 normal; por lo que la construcción de direcciones IPv6 en 6LowPAN comienza desde que se conoce la información del prefijo y se conocen las direcciones de capa de enlace, esto es lo que nos permite una relación de compresión alta en la cabecera.

#### 1.2.4 FORMATO DE CABECERA [3]

La principal funcionalidad de 6LowPAN está en su capa de adaptación LowPAN, lo que permite la compresión IPv6 y de las siguientes cabeceras tales como UDP junto con la fragmentación y las características de direccionamiento en malla. Las cabeceras 6LowPAN se definen en el RFC4944, que ha sido posteriormente mejorado y ampliado por ID-6LoWPAN-hc; por lo que la compresión de 6LowPAN es *stateless*, y consecuentemente muy simple y fiable. Se basa en la información conocida compartida por todos los nodos desde su participación en LowPAN, y el espacio de direcciones IPv6 jerárquicas que le permite direcciones IPv6 que se deben evitar la mayoría de veces.

La cabecera LowPAN consiste en un valor de despacho que identifica el tipo de encabezado, seguido por un byte de compresión de cabeceras IPv6 que indica qué campos se comprimen, y luego cualquier campo IPv6 en línea. Si, por ejemplo, las cabeceras de extensión UDP o IPv6 siguen a IPv6, entonces, estas cabeceras pueden también ser comprimidas utilizando lo que se denomina compresión de próxima cabecera. Un ejemplo de compresión 6LowPAN se da en la Figura 1.7; en el paquete superior un valor de despacho LowPAN de un byte se incluye para indicar IPv6 completo sobre IEEE 802.15.4. Al contrario en la Figura 1.8 muestra un ejemplo de 6LowPAN / UDP en su forma más simple (equivalente al paquete inferior en la Figura 1.7), con un valor de despacho y de compresión de cabecera IPv6 (LOWPAN\_IPHC) según la ID-6LoWPAN-hc (2 bytes), todos los campos IPv6 comprimidos, a continuación, seguidas de un byte de una compresión de siguiente cabecera de UDP(LOWPAN\_NHC) con compresión de campos de puerto origen y destino y la suma de comprobación UDP (4 bytes).

Consecuentemente, en el mejor de los casos probablemente la cabecera 6LowPAN/UDP será sólo de 6 bytes de longitud en comparación a un encabezado estándar IPv6 / UDP que es de 48 bytes de longitud como se muestra en la Figura 1.9. Por lo que, si se considera que en el peor de los casos IEEE 802.15.4 tiene sólo 72 bytes de carga útil disponible después de las cabeceras de capa de enlace, demostrando que la compresión es importante.

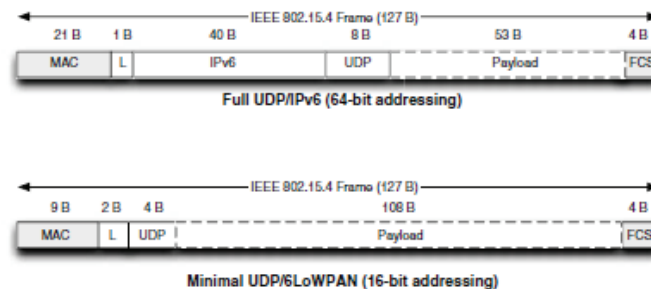


Figura 1.7 Ejemplo de compresión de cabecera 6LowPAN (L = cabecera LowPAN)  
Fuente: [3]

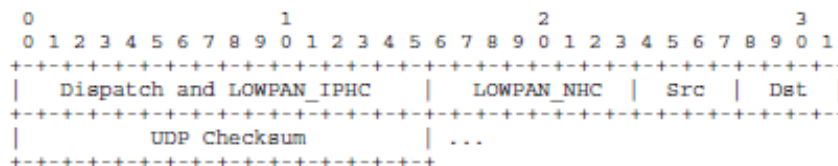


Figura 1.8 Compresión de cabeceras 6LowPAN/UDP (6 bytes)  
Fuente: [3]

### 1.2.5 PROCESO DE INICIO [3]

Las aplicaciones de 6LowPAN por lo general implican dispositivos y redes completamente autónomos, que deben auto configurarse a sí mismos sin la intervención humana; por ende el proceso de inicio primero necesita ser realizado por la capa de enlace, con el fin de permitir la comunicación básica entre los nodos dentro del alcance de radio. La configuración básica de capa de enlace por lo general implica la configuración de ajuste de canal, clave de seguridad por defecto y dirección. Una vez que la capa de enlace está funcionando y las comunicaciones de un solo salto entre dispositivos es posible, el descubrimiento de vecinos de 6LowPAN (ID-6LoWPAN-nd) se utiliza para arrancar toda la LowPAN.

El descubrimiento de vecinos es una característica clave de IPv6, ya que se ocupa de lo más básico del proceso de inicio y de los problemas de mantenimiento de los enlaces IPv6 entre los nodos. El descubrimiento de vecino básico de IPv6 no es adecuado para su uso con 6LowPAN; por lo que el grupo de trabajo 6LowPAN ha definido el descubrimiento de vecinos 6LowPAN (6LowPAN-ND) optimizado para redes inalámbricas de baja potencia y para 6LowPAN en particular (ID-6LowPAN-nd). La especificación 6LowPAN-ND describe la configuración automática de la red y la operación de los hosts, routers y routers de borde en LowPAN. Un registro de los nodos en cada LowPAN se mantiene en un router de borde correspondiente, lo que simplifica la operación de IPv6 a través de la red y reduce la cantidad de inundaciones *multicast*; además, 6LowPAN-ND permite LowPAN que cubren muchos enrutadores de borde conectados por un enlace de *backbone*<sup>17</sup> común (por ejemplo, Ethernet), y la generación única de direcciones cortas de capa de enlace.



Figura 1.9 Cabeceras estándar IPv6/UDP (48 bytes)

Fuente: [3]

<sup>17</sup> *Backbone*: Es el enlace principal de una red.

## CAPÍTULO II

### 2 ALGORITMO DE OPTIMIZACIÓN DE FLUJO DE DATOS

#### 2.1 TOPOLOGÍAS LINEALES CON WSN

Las topologías lineales son redes en las cuales se debe colocar los sensores inalámbricos de tal forma que conformen con sus enlaces una línea recta como se indica en la Figura 2.1. Estas redes constan de nodos sensores y un *Gateway*, el cual permitirá el envío de información fuera de la red de sensores inalámbricos y también se encargará de iniciar y configurar la red de sensores inalámbricos.

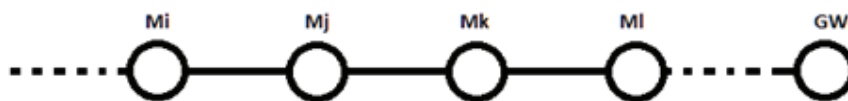


Figura 2.1 Ejemplo de topología lineal con WSN

Estas topologías funcionan de la siguiente manera:

- Un nodo tiene información que enviar.
- El nodo genera un paquete, dicho paquete pasa por todos los nodos de la topología que se denominan nodos intermedios hasta llegar al dispositivo *Gateway*.

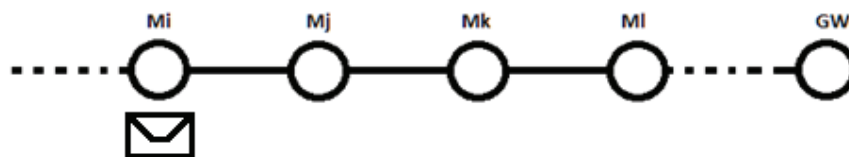


Figura 2.2 Ejemplo de envío de un paquete en la topología lineal

- Como se observa Figura 2.2, el nodo Mi genera un paquete y lo envía hacia el *Gateway*, este paquete debe ser procesado por los nodos intermedios: Mj, Mk, MI y así sucesivamente.

Por ello estos nodos tienen que procesar la información lo que influye en la duración de las baterías.

Uno de los principales problemas con estas topologías lineales con red de sensores inalámbricos es el flujo de datos en ella, ya que, al tener que pasar los datos por todos los nodos intermedios, estos necesitan un mayor procesamiento, lo cual influye en la confiabilidad de la red de sensores inalámbricos, esto es debido a que si falla un nodo intermedio se perderá conectividad de algunos nodos de la red de sensores inalámbricos y se perderá la completa funcionalidad de la red de sensores inalámbricos.

## **2.2 WSN 6LowPAN**

Estas redes permiten la comunicación mediante el estándar IEEE 802.15.4, el cual tiene sus limitaciones en relación al alcance y la sensibilidad de los dispositivos. Estos varían dependiendo del hardware y sus prestaciones. Antes de formar la red 6LowPAN primero se realiza la configuración a nivel de capa de enlace, con lo que se permite la comunicación entre nodos cercanos o en el rango de alcance de la señal del nodo sensor. Una vez realizado la configuración a nivel de capa de enlace comienza a ejecutar el descubrimiento de vecinos de 6LowPAN, en el cual se le asigna una dirección corta al nodo sensor. Estas redes utilizan una dirección asignada por el *Gateway* para la comunicación interna, esta dirección tiene un tamaño de 2 bytes, lo que nos permite tener un máximo de 65536 nodos sensores, enumerados desde el 0, el cual es asignado al nodo *Gateway*. La asignación de la dirección depende del orden de encendido, y van desde el 1 hasta el número máximo que se puede asignar. Por lo que, la asignación es una limitante de la red de sensores inalámbricos, ya que, nos condicionan al momento de ejecutar el algoritmo.

## **2.3 ALGORITMO**

El algoritmo utiliza las funciones del nivel de red con IPV6, específicamente 6LowPAN, para crear redes lógicas en una topología lineal, ejecutando el algoritmo propuesto en cada nodo sensor. El algoritmo está dividido en dos fases, la primera fase es la agregación del nodo sensor a la red 6LowPAN, y la segunda fase es la división de la red lógica. Cabe recalcar que la primera fase se ejecuta una sola vez, ya que este procedimiento ocurre al momento de encender el nodo sensor.

### 2.3.1 PRIMERA FASE

Al encender el nodo sensor, este se agrega a la red 6LowPAN; el tiempo de asociación del nodo a la red dependerá de su distancia al *Gateway*, ya que los nodos más cercanos se agregan en menos tiempo a la red y el último en agregarse será el más alejado; por lo mencionado anteriormente, las comunicaciones en esta fase serán salto a salto (*multi-hop*). Este procedimiento comienza con el envío de un mensaje *node registration* (NR) al nodo de borde (*Gateway*), este nodo responde con un *node confirmation* (NC), con el cual se confirma la agregación del nodo sensor a la red 6LowPAN. Una vez agregado el nodo sensor a la red 6LowPAN, el *Gateway* asignará una dirección corta identificando al nodo, que se almacenará en la variable *IdNodo*, la dirección corta. El uso de este tipo de direcciones, permite que exista una comunicación entre los nodos sensores dentro de la red WSN 6LowPAN, minimizando el procesamiento y consumo de energía. Además se asigna el valor de 0 a la variable *Acknexthop*, el cual será su valor inicial. La variable *Acknexthop* permite el manejo de los acuses de recibo de los datos enviados. La Figura 2.3 indica cómo se asignan valores iniciales a las variables *Idnodo* y la asignación de *Acknexthop*.

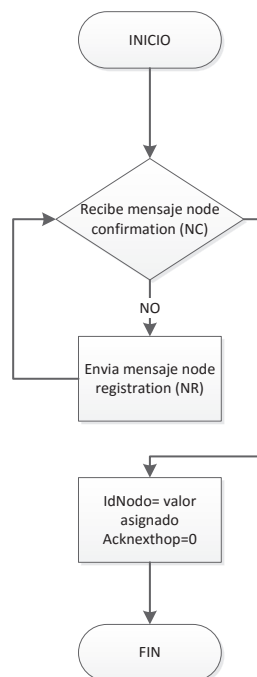


Figura 2.3 Diagrama de Flujo Primera Fase



### 2.3.2 SEGUNDA FASE

Cuando el nodo sensor necesita enviar un dato hacia el destino o recibe un dato para retransmitir, ejecuta el algoritmo en el cual comienza con el cálculo del puerto destino del siguiente salto asociado a la red lógica (*Portnexthop*) y luego el cálculo de la dirección destino del siguiente salto asociado a la red lógica (*Destnexthop*).

Debido a que los nodos sensores no pueden recibir datos de diferentes nodos al mismo tiempo, es necesario hacer el cálculo del puerto de destino. Para el cálculo del puerto destino del posible salto asociado a la red lógica *Portnexthop*, primero se verifica el valor de la confirmación (*Acknexthop*), ya que dependiendo de este valor enviará los datos al puerto 1024 o al puerto 1025 según corresponda. Si *Acknexthop* es igual a 1 entonces envía al puerto 1025, pero si *Acknexthop* es igual a 0 envía al puerto 1024.

Para el cálculo de la dirección destino del siguiente salto asociado a la red lógica *Destnexthop*, el algoritmo primero verifica si el nodo sensor tiene la dirección corta (*IdNodo*) cuyo valor sea 1 ó 2, las cuales corresponden a los nodos sensores que se conectarán directamente con el nodo *Gateway*, por lo tanto el algoritmo devolverá la dirección destino (*Destnexthop*) con el valor de 0, la cual es la dirección corta del nodo *Gateway*; en caso de que no sean esos valores, el algoritmo resta el valor de dos a la dirección corta (*IdNodo*) del nodo sensor y suma el valor de la confirmación (*Acknexthop*), finalmente se le asigna el valor de 1 a la confirmación (*Acknexthop*), el manejo de este valor se explicará más adelante.

Al momento de la creación del paquete se coloca la dirección destino (*Destnexthop*), puerto destino (*Portnexthop*) calculado y el dato, para finalmente enviar el paquete.

Finalmente, para saber si existe una correcta comunicación con el nodo sensor al que envía su dato, éste deberá confirmar que el dato fue recibido con éxito; al recibir la confirmación se asignará a *Acknexthop* el valor de 0, con lo que estará funcionando correctamente la red lógica, en caso contrario (de no recibir la confirmación *Acknexthop*) tendrá el valor de 1, el cual fue asignado al momento

de realizar el cálculo de *Destnexthop*, lo que significa que la red lógica no funciona correctamente; entonces el siguiente dato a ser enviado cambiará la dirección destino (*Destnexthop*) y el puerto destino (*Portnexthop*).

Lo que se pretende con el algoritmo propuesto, es que no exista pérdida de la comunicación de los nodos restantes si algún nodo de cualquiera de las subredes falla, además que los tiempos de recuperación de la red sean mínimos en comparación a los tiempos de recuperación que tiene el protocolo de enrutamiento de IPv6 aplicado a topologías lineales.

En la Figura 2.4 se muestra el diagrama de flujo del funcionamiento de la división de red lógica.

En el diagrama de flujo se observa que la variable principal para el funcionamiento del algoritmo es *Acknexthop*, esta variable permite el aumento de la confiabilidad de la red de sensores inalámbricos 6LowPAN, ya que, con ella se calcula la dirección del siguiente salto (*Destnexthop*), y el puerto del siguiente salto (*Portnexthop*), y evitará la caída de una de las dos redes lógicas.

## **2.4 PARÁMETROS DE COMPROBACIÓN**

Los parámetros para la comprobación dependen de los datos llevados en el *payload* o carga útil del paquete, para lo cual se tendrán los siguientes procedimientos para la determinación del funcionamiento.

### **2.4.1 FUNCIONAMIENTO ÓPTIMO DE LAS 2 REDES LÓGICAS**

Permite comprobar que están funcionando correctamente las dos redes lógicas con topología lineal. Al momento de ejecutar el algoritmo en los nodos sensores, estos envían un paquete, que en su *payload* constará la dirección corta del nodo sensor.

Este *payload* irá cambiando dependiendo de la ruta por la cual sea retransmitido el paquete hasta llegar al nodo de borde (*Gateway*). Este cambio permite que, si el paquete tiene que ser retransmitido, en el *payload* se agregue la dirección corta del nodo que hace la retransmisión, por lo que al llegar al nodo de borde (*Gateway*), se encontrará en el *payload* todo el camino que siguió el paquete.

## 2.4.2 FUNCIONAMIENTO DEL MÉTODO DE RECUPERACIÓN DE LA RED LÓGICA CON FALLA

Permite comprobar que aumentó la confiabilidad de la red de sensores inalámbricos 6LowPAN, para lo cual se desconectará o provocará una falla en un nodo que tiene que hacer retransmisión, lo que implica que un nodo sensor perteneciente a una de las redes lógicas no tendrá retransmisión e inmediatamente se conmutará hacia la otra red lógica; esto se comprobará en el *payload* del paquete que llegará al nodo de borde (*Gateway*).

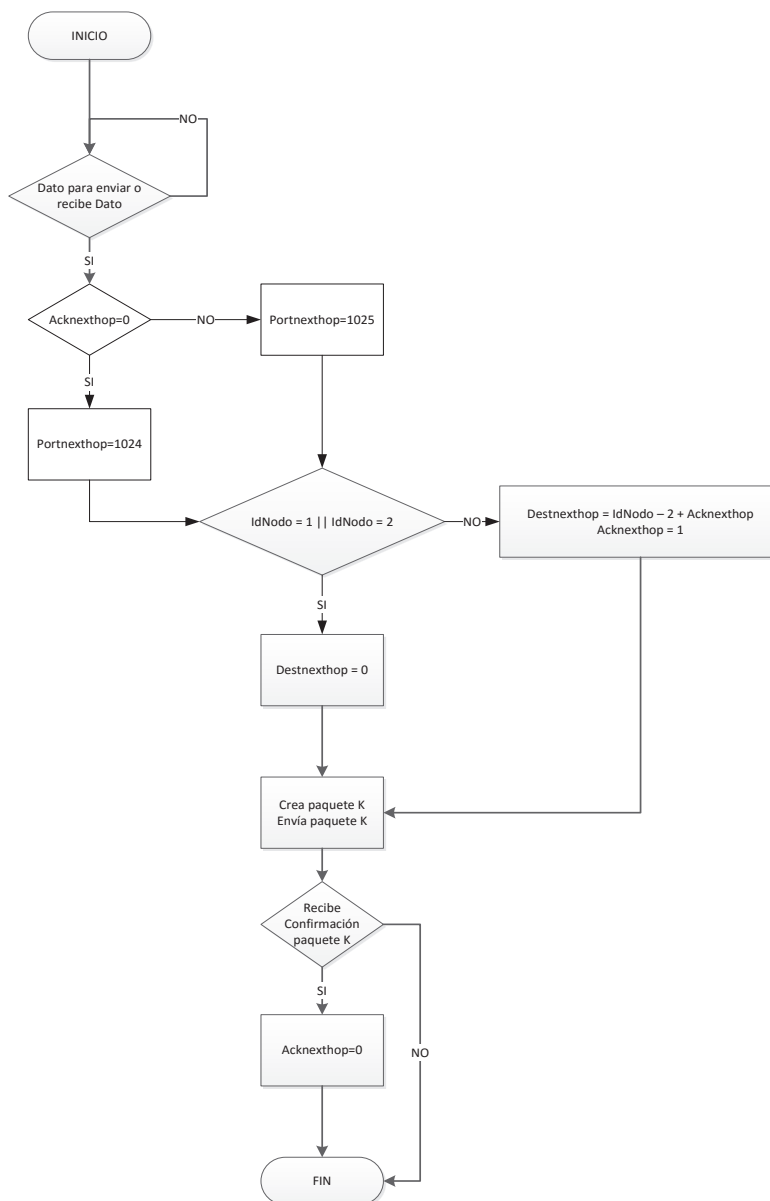


Figura 2.4 Diagrama de Flujo de la División de Red Lógica

## CAPÍTULO III

### 3 IMPLEMENTACIÓN Y PRUEBAS DEL ALGORITMO

#### 3.1 IMPLEMENTACIÓN DEL PROTOTIPO DE RED 6LoWPAN

##### 3.1.1 ANÁLISIS DE REQUERIMIENTOS PARA LA IMPLEMENTACIÓN DEL PROTOTIPO

El prototipo de red 6LoWPAN que se muestra en la Figura 3.1 consta de cuatro nodos sensores y un nodo *Gateway*. La distancia entre cada uno de los nodos deber ser la mitad de la distancia máxima. Este es un requisito necesario para la comprobación del algoritmo planteado.

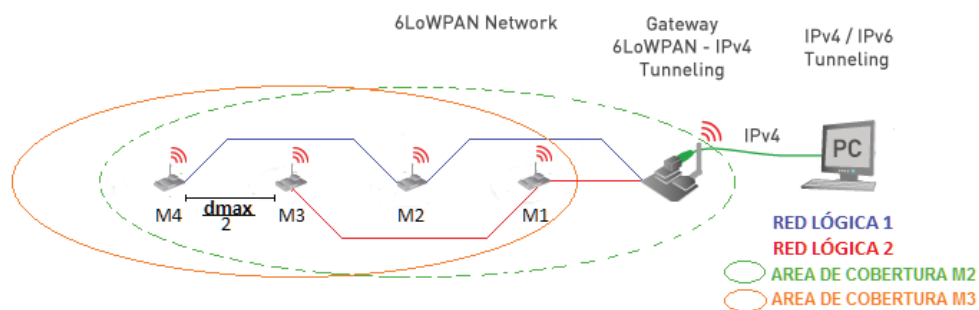


Figura 3.1 Prototipo de red 6LoWPAN

Los requerimientos para la implementación del prototipo son:

- Kit de desarrollo *Waspnote Pro V1.2*
- Sistema operativo Linux Ubuntu 12.04LTS de 64 bits
- Software *MonoDevelop*
- Plataforma *Mote Runner (Mote Runner SDK<sup>18</sup> Software Development Kit)*
- Avrdude 5.11.1-1

##### 3.1.1.1 Kit de desarrollo *Waspnote Pro V1.2* [5]

IBM y Libelium han unido esfuerzos para ofrecer una plataforma de desarrollo de IPv6 para redes de sensores y el Internet de las Cosas (IoT). Mediante la

<sup>18</sup> SDK: Kit de Desarrollo de Software, es generalmente un conjunto de herramientas de desarrollo de software que le permiten al programador crear aplicaciones para un sistema específico.

integración de IBM *Mote Runner* SDK en la parte superior de la plataforma de sensores Libelium *Waspote*, se obtiene una herramienta única y poderosa para los desarrolladores e investigadores interesados en conectividad 6LoWPAN con IPv6 para el Internet de las Cosas

### 3.1.1.2 Ubuntu 12.04LTS [6],[7]

Es un sistema operativo basado en GNU/LINUX y que se distribuye como software libre, lo que permite un fácil acceso a este sistema operativo. LTS (*Long Term Support*) significa que se tiene un soporte más prolongado, ya que las versiones que son LTS tienen soporte de al menos 9 meses.

### 3.1.1.3 MonoDevelop [8]

Es un IDE<sup>19</sup> GNOME<sup>20</sup> gratuito diseñado principalmente para C # y otros lenguajes .NET, aunque abierto a cualquier tipo de lenguaje. Sin embargo, *MonoDevelop* espera ser algo más que un IDE: tiene la intención de ser una plataforma extensible sobre la que cualquier tipo de herramienta de desarrollo se pueda construir.

### 3.1.1.4 Mote Runner SDK [5]

IBM *Mote Runner* es un sistema operativo para los nodos de sensores inalámbricos, además es *run-time* (tiempo de ejecución) y el entorno de desarrollo de redes de sensores inalámbricos (WSN).

#### 3.1.1.4.1 Características No Soportadas [5]

Debido a los recursos limitados en plataformas integradas, *Mote Runner* no es capaz de soportar todas las características del lenguaje. Se trata de un estrategia común para definir un subconjunto del lenguaje de Java o C# adecuado para sistemas pequeños.

En la siguiente lista muestra las restricciones y omisiones más importantes:

Restricciones:

---

<sup>19</sup> IDE (*Integrated Development Environment*): es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software.

<sup>20</sup>GNOME: es un entorno de escritorio e infraestructura de desarrollo para sistemas operativos GNU/Linux, Unix y derivados Unix como, BSD o Solaris; compuesto enteramente de software libre.

- Tipos de datos *float* y *double*
- 64bits o más números enteros
- Arreglos multidimensionales
- Reflexión
- Plantillas
- Hilos y primitivas de sincronización
- Arreglos booleanos
- *Boxing*
- API de tiempo de ejecución estándar
- Enumeraciones

Omisiones:

- Clases internas en C# (Se admiten clases internas en Java)
- Delegados *multicast* (solo soporta delegados *unicast*)
- *Strings* (excepto en situaciones especiales) y Arreglos de *Strings*
- Cualquier información de tipo en tiempo de ejecución (por ejemplo *getClass* (), nombres de clases. Etc.), excepto para los operadores del lenguaje como: *instanceof*, *is*, y *as*

Las API de tiempo de ejecución estándar, utilizadas para la compatibilidad con el lenguaje y las bibliotecas estándar de Java y C #, son demasiado extensas para ser contenidas en pequeños sistemas embebidos. Por este motivo, *Mote Runner* define sus propias API del sistema y cualquier soporte Lenguaje *Runtime* se ha reducido al mínimo. Esto significa que no hay reflexión, no hay información de tipo simbólica en tiempo de ejecución, no hay clases para los tipos de valor *boxing*, la clase raíz *Object* viene sin ningún tipo de métodos y campos, y más.

Las matrices multidimensionales no están directamente soportadas. Sin embargo, es posible simularlas mediante *arrays* de objetos de anidación.

Las matrices booleanas no son compatibles dado que almacenar bits en paquetes requiere demasiado código rara vez utilizado. Si los bits no están almacenados en paquetes, las matrices booleanas son básicamente matrices de bytes. Por esta razón, resulta ineficiente y no es conveniente utilizarlos.

#### 3.1.1.4.2 *Mote Runner Compiler* [5]

El Compilador *Mote Runner* (mrc) se utiliza para convertir el código fuente en lenguaje ensamblador que se puede cargar en una plataforma *Mote Runner*. El mrc organiza la invocación de diversas herramientas, dependiendo de la entrada proporcionada y la salida requerida para lograr la transformación pretendida.

#### 3.1.1.4.3 *Mote Runner Shell* [5]

Proporciona una interfaz de línea de comandos para el entorno de programación y la gestión basada en *Javascript* para *Mote Runner*, como se indica en la Figura 3.2.

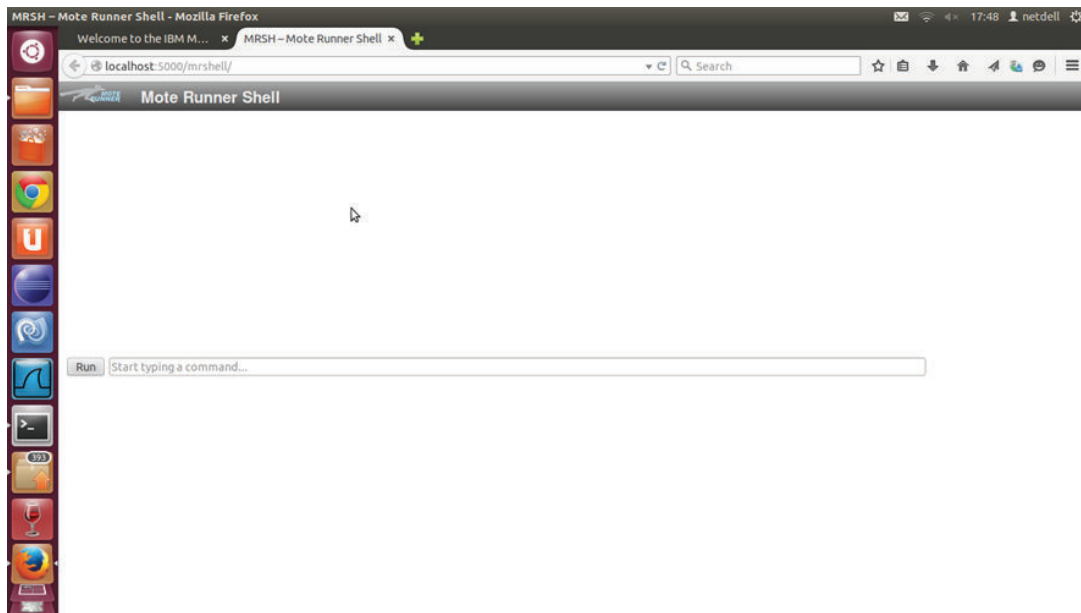


Figura 3.2 Mote Runner Shell

#### 3.1.1.4.4 *MRV6* [5]

*Mote Runner* viene con un protocolo basado en 6LoWPAN, llamado MRv6, que puede ser instalado en los nodos para proporcionar IPv6. MRv6 es una red basada en *multi-hop* TDMA que permite una comunicación fundamentada en IPv6 entre *hosts* en Internet y nodos en la red inalámbrica.

MRv6 no está construido como un componente de *Mote Runner*, no obstante, se encuentra totalmente implementado en C #, empaquetado en *asemblys* y se instala de manera predeterminada con los mecanismos de carga de *Mote Runner*. Además MRv6 es adecuado para aplicaciones de recopilación de datos en donde

los nodos sensores envían periódicamente los datos a un host remoto y rara vez son objeto de actualizaciones o peticiones de *hosts* remotos.

La red MRv6 es administrada en su totalidad por el nodo de borde. Asimismo, en los demás nodos se implementa una funcionalidad de enrutamiento limitada.

El nodo de borde decide sobre las solicitudes de asociación, asigna horarios de comunicación entre nodos inalámbricos, y determina las rutas de la red.

En los nodos, se puede cargar aplicaciones cuyo funcionamiento es independiente del manejo de la red MRv6. Una vez conectado, se puede consultar la red, comunicarse con nodos y recuperar datos de ellos. El esquema de paquetes utilizado en la red forma parte de las especificaciones 6LowPAN que implica una compresión del estándar IPv6.

MRv6 cuenta con un programa de túnel para nodos conectados a un nodo de borde. Éste realiza tareas de conversión de paquetes IPv6 a 6lowPAN y viceversa, y los reenvía a destinos remotos y nodo de borde.

MRv6 soporta un *Shell* y una interfaz de programación para administrar, depurar, consultar y comunicarse con una red inalámbrica, ya sea física o simulada. El túnel sólo es necesario cuando la comunicación se da con hosts externos para lo cual es necesario que esté basado en direccionamiento IPv6.

#### **3.1.1.5 AVRdude 5.11.1-1 [9],[10]**

Es un programa para la descarga de código y datos para microcontroladores Atmel AVR. Permite la realización de la configuración del hardware de los nodos sensores inalámbricos; al ser software libre está disponible en las versiones de Linux.

## **3.2 IMPLEMENTACIÓN Y CONFIGURACIÓN DEL PROTOTIPO DE RED 6LowPAN**

### **3.2.1 INSTALACIÓN DEL FIRMWARE EN CADA NODO WASPMOTE PRO V1.2**

El *firmware* es cargado en cada uno de los nodos, con ayuda del programador AVR y el archivo en formato *.hex*, el cual deberá ser cargado.



Para estos fines, se utiliza el programa de computación AVR Studio en su distribución para el sistema operativo Windows o Linux gracias al software AVRdude.

Para la programación del nodo es necesaria la conexión apropiada del programador de AVR, que en este caso es el AVRISP mkII como se indica en la Figura 3.3 y la Figura 3.4.

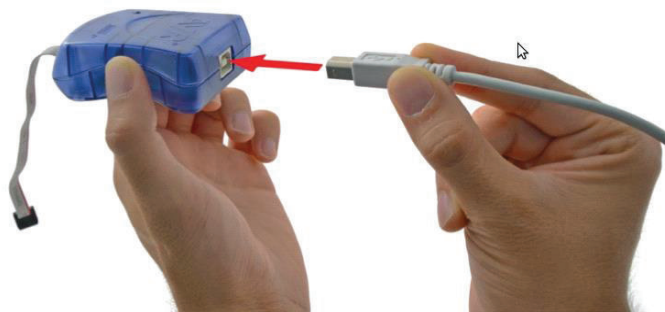


Figura 3.3 Conectando el programador AVR para instalar Firmware  
Fuente: [5]

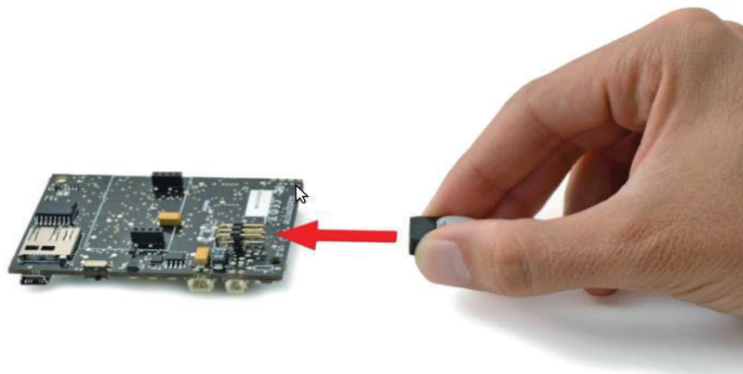


Figura 3.4 Conectando el programador AVR al nodo Waspmote Pro V1.2  
Fuente: [5]

Una vez conectado el nodo al programador AVR, se necesita ejecutar dos comandos. El primer comando, el cual se indica en la Figura 3.5, permite establecer los fusibles para poder transmitir el archivo *hex*, el cual es el *firmware* que deseamos cargar en los nodos sensores.

```
1 root@ubuntu# 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
```

Figura 3.5 Comando para establecer y quemar los fusibles

El segundo comando, el cual se indica en la Figura 3.6, permite instalar el *firmware* en los nodos, para esto el archivo *hex* se encuentra en el siguiente directorio “/moterunner/firmware/waspmote.hex”

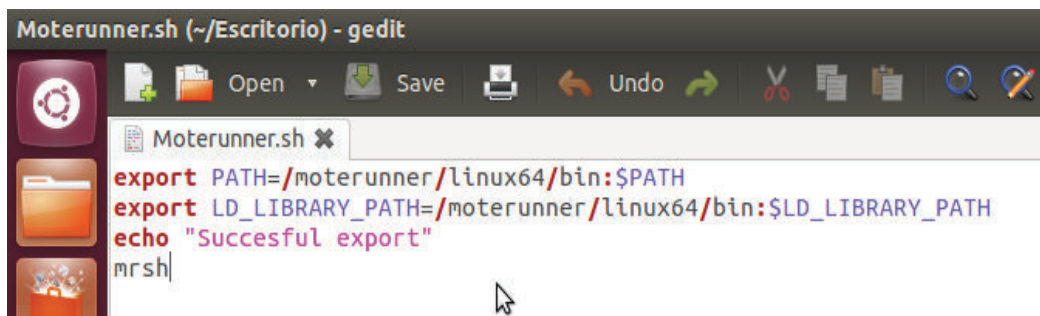
```
1 root@ubuntu# sudo avrdude -c avrispmkII -p m1281 -P usb -b 115200 -U
flash:w:/moterunner/firmware/waspmote.hex
```

Figura 3.6 Comando para instalar el firmware en los nodos

Una observación importante es que el programador AVR deja de ser necesario para cargar las librerías y el programa en los nodos, dado que esto se realiza mediante la conexión USB.

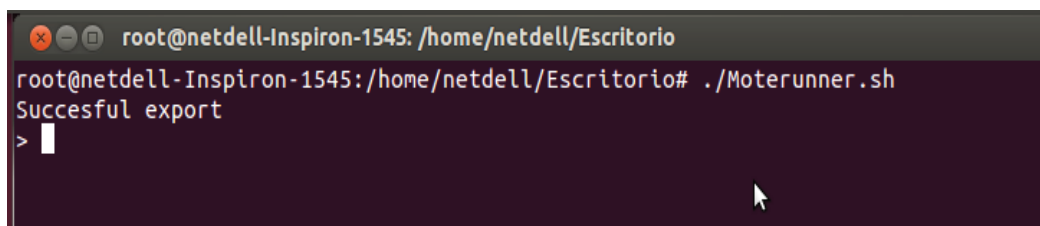
### 3.2.2 SERVIDOR WEB MRSH (*MOTE RUNNER SHELL*)

La ejecución del servidor web se realiza mediante un *script*, como se indica en la Figura 3.7, el mismo permite la exportación de las variables de entorno necesarias para los ejecutables de la aplicación denominada *mrsh*. Se indica en la Figura 3.8 como ejecutar el *script* en el terminal de comandos de Linux.



```
Moterunner.sh (~/Escritorio) - gedit
export PATH=/moterunner/linux64/bin:$PATH
export LD_LIBRARY_PATH=/moterunner/linux64/bin:$LD_LIBRARY_PATH
echo "Successful export"
mrsh
```

Figura 3.7 Script para ejecutar el Servidor Web MRSH



```
root@netdell-Inspiron-1545: /home/netdell/Escritorio
root@netdell-Inspiron-1545: /home/netdell/Escritorio# ./Moterunner.sh
Successful export
>
```

Figura 3.8 Ejecución del script para poner a funcionar el Servidor Web MRSH

Una vez ejecutado el *script*, se inicia Firefox, se ingresa la dirección localhost:5000 y se presenta la página inicial de *Mote Runner*, como se indica en la Figura 3.9. Posteriormente, se selecciona la opción *Shell* que presenta en otra pestaña el *Mote Runner Shell*, como se indica en la Figura 3.2.

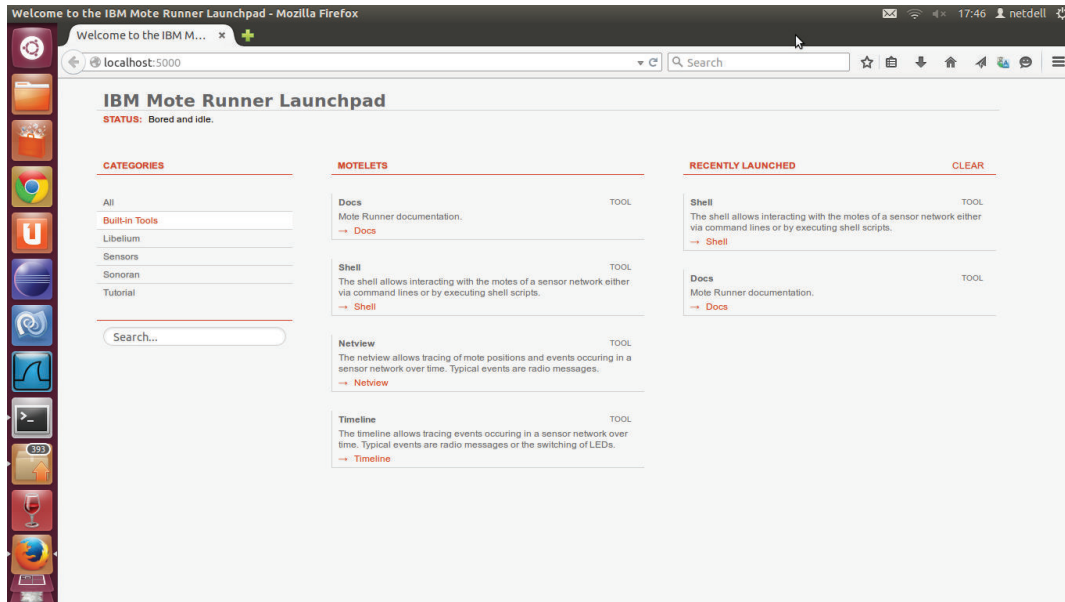


Figura 3.9 Página inicial de Mote Runner

### 3.2.3 INSTALACIÓN DE LA LIBRERÍA MRV6

En la instalación de la librería se debe tomar en cuenta cual nodo se configura como *Gateway*, ya que el mismo contiene una diferente librería de *mrV6* que los demás nodos pertenecientes a la red 6LowPAN.

#### 3.2.3.1 Gateway

La librería que se debe cargar al nodo *Gateway* se denomina *mrV6-edge*. El proceso de conexión con el nodo sensor es mediante el puerto USB, ejecutando los comandos indicados en la Figura 3.10 en el *Mote Runner Shell*.

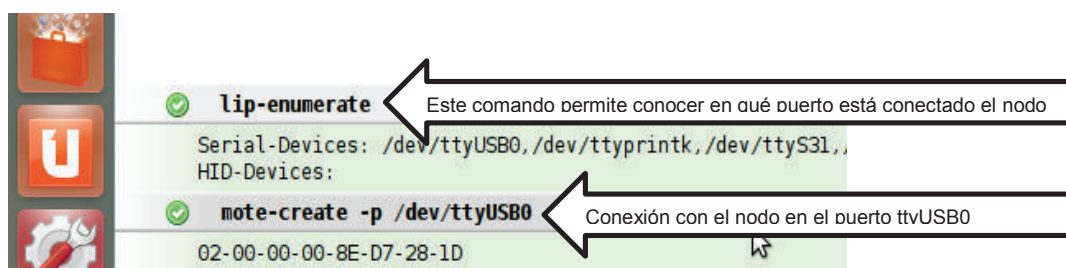


Figura 3.10 Comandos para la conexión USB con el nodo Gateway

Una vez conectado el nodo *Gateway*, se ejecuta el comando *moma-list* con el que se observa que programas están cargados en el nodo sensor. Una vez observados los programas cargados, se cambia de directorio hacia donde se ubica el *mrV6-edge*, como se indica en la Figura 3.11.

```

02-00-00-00-8E-D7-28-1D
✓ moma-list
Mote-Id      Outdated State Assembly  Id Version  Mote-Address
-----
02-00-00-00-8E-D7-28-1D  OK  saguaro-system  0  11.4.34896  lip:/dev/ttyUSB0
02-00-00-00-8E-D7-28-1D  OK  yasmote-system  1  14.0.34899
✓ cd /moterunner/gac
/moterunner/gac

```

Comando que me permite observar los programas cargados

Cambio de directorio donde se encuentra la librería MRV6

Figura 3.11 Comando moma-list y cambio de directorio

Una vez ubicados en el directorio que contiene al archivo correspondiente a la librería *mr6-edge*, se carga con el comando *moma-load*. A continuación, ya cargada la librería, se procede a configurar la dirección IPv4 en el nodo *Gateway* con el comando *moma-ipv4*, estos procedimientos se indican en la Figura 3.12.

```

✓ moma-load mr6-edge-1.0
mr6-edge-1.0.29276(a:02)
✓ moma-ipv4 -i 192.168.1.223 -g 192.168.1.1 -s 255.255.255.0 --udp 9999
New IP configuration will take effect upon
Run Start typing a command...

```

Cargando librería mr6-edge

Configuración de una dirección IPv4 al Gateway

Figura 3.12 Cargando librería mr6-edge y configuración IPv4 nodo Gateway

Entre los parámetros para configuración de la dirección IPv4 se toman: la dirección asignada, la dirección de *Gateway* con su máscara de red y finalmente el puerto UDP por donde se realiza la comunicación. La configuración tendrá efecto luego de realizar el reinicio del nodo.

### 3.2.3.2 Nodo

La librería cargada en los demás nodos se denomina *mr6-lib*. El proceso de conexión con el nodo sensor es mediante el puerto USB, ejecutando los comandos indicados en la Figura 3.13 en el *Mote Runner Shell*.

```

✓ lip-enumerate
Serial-Devices: /dev/ttyUSB0,/dev/ttyprintk,/dev/ttyS31,
HID-Devices:
✓ mote-create -p /dev/ttyUSB0
02-00-00-00-8E-D7-28-1D

```

Este comando permite conocer en qué puerto está conectado el nodo

Conexión con el nodo inalámbrico

Figura 3.13 Comandos para la conexión USB con el nodo

La ubicación de las librerías *mrsv6-lib* y *mrsv6-edge*, es la misma. Una vez ubicados en dicho directorio, se procede a cargar *mrsv6-lib* en el nodo, con el comando que se indica en la Figura 3.14.

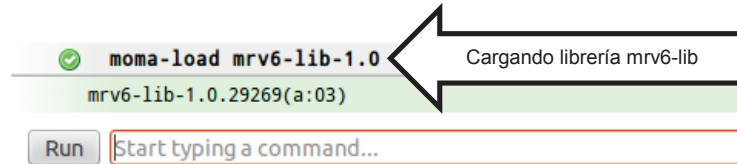


Figura 3.14 Cargando la librería *mrsv6-lib* en el nodo

### 3.2.4 CONFIGURACIÓN RED 6LOWPAN

La configuración de la red 6LowPAN se necesita editar un archivo de denominado *route\_setup\_linux\_ipv6.sh*, y ejecutar una aplicación denominada *tunnel*. Los cuales están ubicados en el directorio que se indica en la Figura 3.15. El archivo y la aplicación permiten la creación del túnel IPv4/IPv6 y la asignación de un prefijo de red IPv6 a los nodos sensores.

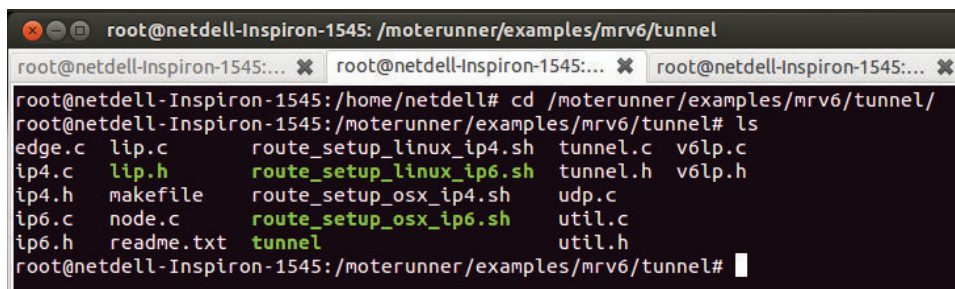


Figura 3.15 Directorio donde se encuentran los archivos del túnel

En el archivo *route\_setup\_linux\_ipv6.sh* se encuentra la información del prefijo de red y la dirección IPv6 del túnel configurado, como se indica en la Figura 3.16.

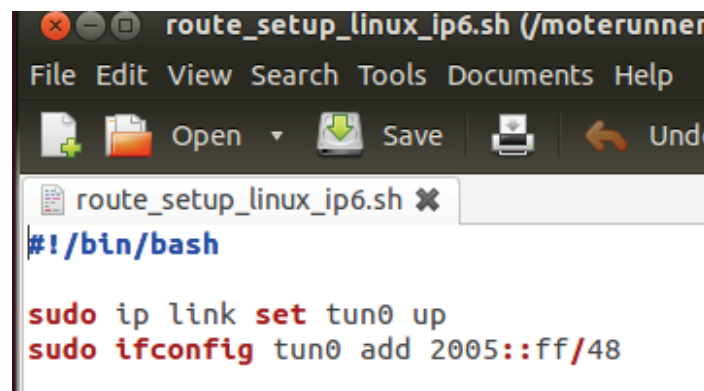


Figura 3.16 Archivo *route\_setup\_linux\_ipv6.sh*

Una vez hecha la configuración del archivo *route\_linux\_ip6.sh*, se procede a ejecutar el *Mote Runner Shell* y la aplicación *tunnel*. Luego se hace una conexión mediante el cable Ethernet con el nodo *Gateway*, para ello se debe configurar la tarjeta Ethernet con la dirección *Gateway* y máscara de red que se configuró en el *Gateway*, como se indica en la Figura 3.17.

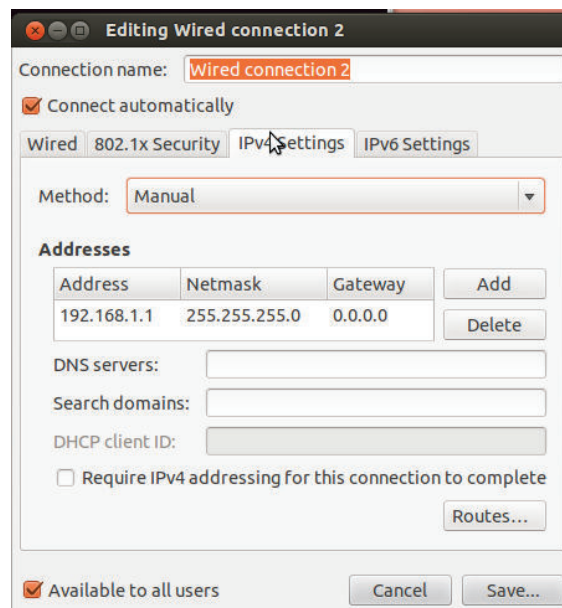


Figura 3.17 Configuración de la tarjeta Ethernet

Posterior a la configuración de la tarjeta Ethernet, en el *Mote Runner Shell* se ejecuta el comando para la conexión con el nodo *Gateway* y la configuración de la red 6LowPAN como se indica en las Figura 3.18 y Figura 3.19.

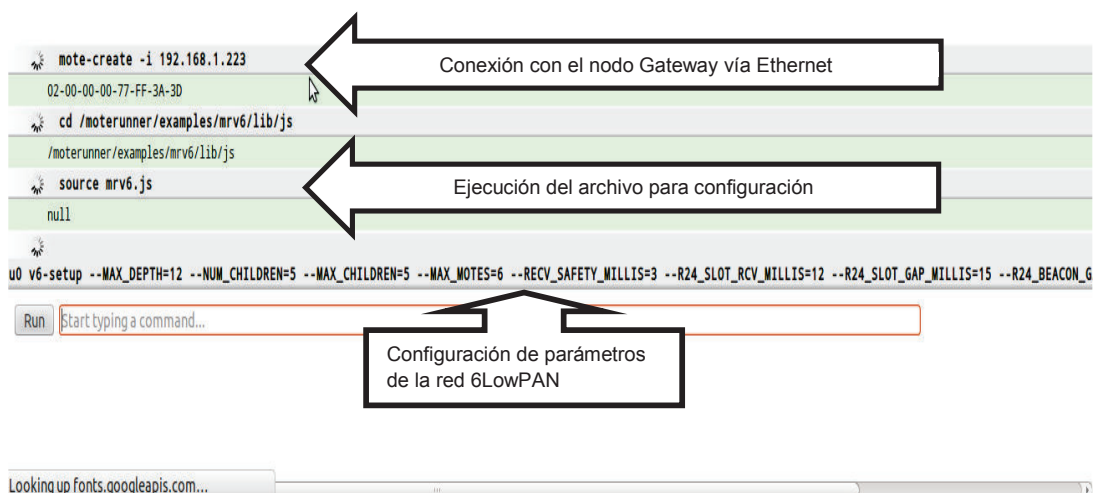


Figura 3.18 Comandos para la configuración de la red 6LowPAN



```

Mote Runner Shell - Mozilla Firefox
Welcome to the IBM M... x MRSH--Mote Runner Shell x
localhost:5000/mrshell/
Mote Runner Shell
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
NODE_BUFFERS_SIZE 512
NODE_BUFFERS_CNT 8
MAX_MOTES 6
MAX_DEPTH 12
MAX_CHILDREN 5
NUM_CHILDREN 5
CHILD_MISSED_BEACON_LIMIT 10
SLOT_TRANSMISSION_TRIES 5
RCV_SAFETY_MILLIS 3
PACKET_BUFFER_MIN_SIZE 100
EDGE_SCAN_EXISTING 0
EDGE_AUTOSTART 0
Starting mrV6 which is not yet active.
Run [start typing a command...]

```

Parámetros que han sido configurados en la red 6LowPAN

Figura 3.19 Parámetros configurados en la red 6LowPAN

Finalmente, se obtiene en la aplicación *tunnel* la dirección del *Gateway* sin asignar el prefijo IPv6 como se indica en la Figura 3.20.

Para la asignación del prefijo se necesita ejecutar el archivo de configuración denominado *route\_setup\_linux\_ip6.sh* y se obtiene el resultado indicado en la Figura 3.21.

```

root@netdell-Inspiron-1545: /moterunner/examples/mrv6/tunnel
root@netdell-Inspi... x root@netdell-Inspi... x root@netdell-Inspi... x root@netdell-Inspi... x
root@netdell-Inspiron-1545:/home/netdell# cd /moterunner/examples/mrv6/tunnel/
root@netdell-Inspiron-1545:/moterunner/examples/mrv6/tunnel# ls
edge.c  lip.c      route_setup_linux_ip4.sh  tunnel.c  v6lp.c
ip4.c  lip.h      route_setup_linux_ip6.sh  tunnel.h  v6lp.h
ip4.h  makefile  route_setup_osx_ip4.sh   udp.c
ip6.c  node.c    route_setup_osx_ip6.sh   util.c
ip6.h  readme.txt tunnel                    util.h
root@netdell-Inspiron-1545:/moterunner/examples/mrv6/tunnel# ./tunnel
Tunnel: opened tunnel device 4, waiting for interface configuration...
New node: 0200000077ff3a3d 0
Nodes Full-Addr          S-Addr      Parent
0000:0000:0000:0000:0200:0000:77ff:3a3d 0           65535
Updated node: 0200000077ff3a3d 0
Nodes Full-Addr          S-Addr      Parent
0000:0000:0000:0000:0200:0000:77ff:3a3d 0           65535

```

Figura 3.20 Dirección del Gateway sin asignar prefijo de red IPv6

```

root@netdell-Inspiron-1545: /moterunner/examples/mrv6/tunnel
root@netdell-Inspiron-1545: /moterunner/examples/mrv6/tunnel# ls
edge.c  lip.c      route_setup_linux_ip4.sh  tunnel.c  v6lp.c
ip4.c   lip.h      route_setup_linux_ip6.sh  tunnel.h  v6lp.h
ip4.h   makefile  route_setup_osx_ip4.sh   udp.c     v6lp.h
ip6.c   node.c     route_setup_osx_ip6.sh   util.c    v6lp.h
ip6.h   readme.txt tunnel              util.h
root@netdell-Inspiron-1545: /moterunner/examples/mrv6/tunnel# ./tunnel
Tunnel: opened tunnel device 4, waiting for interface configuration..
New node: 0200000077ff3a3d 0
Nodes Full-Addr          S-Addr  Parent
0000:0000:0000:0000:0200:0000:77ff:3a3d 0        65535
Updated node: 0200000077ff3a3d 0
Nodes Full-Addr          S-Addr  Parent
0000:0000:0000:0000:0200:0000:77ff:3a3d 0        65535
Tunnel: interface ipaddr: 200500000000000000000000000000ff, xaddr: 00000000000000ff
0ff
Tunnel: network prefix: 2005:0000:0000:0000
Tunnel IP interface configured.

Nodes Full-Addr          S-Addr  Parent
2005:0000:0000:0000:0200:0000:77ff:3a3d 0        65535
icmp6: unhandled ICMPv6 type: 143
icmp6: unhandled ICMPv6 type: 143

```

Figura 3.21 Dirección del Gateway asignado prefijo de red IPv6

### 3.3 IMPLEMENTACIÓN DEL ALGORITMO EN EL PROTOTIPO DE RED 6LowPAN

#### 3.3.1 CÓDIGO DEL ALGORITMO

La aplicación para la realización de las pruebas, fue hecha en una clase llamada *Nodo*, la cual se cargará en todos los nodos pertenecientes a la red 6LowPAN excepto en el nodo *Gateway*. Debido a las limitaciones de no poder utilizar múltiples hilos se ocupa tres sockets los cuales van a permitir el manejo de las comunicaciones con los otros nodos sensores. Para cada uno de los tres sockets se implementa una clase *SenseSocket*.

```

/// <summary>
/// Esta clase maneja el socket en el puerto 1023
/// </summary>
public class SenseSocket : UDPSocket
{
    /// <summary>Constructor </summary>
    public SenseSocket ()
    {
        // Se vincula el socket con el puerto 1023
        this.bind (Nodo.LOCAL_PORT);
    }

    /// <summary>Recibe paquete </summary>
    /// <description>Permite el manejo del paquete recibido por el puerto
    indicado</description>
    public override int onPacket (Packet packet)
    {

```



```

        uint len = packet.payloadLen;
        // creamos un buffer para la informacion recibida
        byte[] buf = (byte[])Util.alloca ((byte)len, Util.BYTE_ARRAY);
        // copiamos la informacion recibida en el buffer
        Util.copyData (packet.payloadBuf, packet.payloadOff, buf, 0,
len);

        // Cambiamos el valor de la confirmacion a 0
        Nodo.ackNextHop = 0;
        return 0;
    }
}

```

**Código 3.1 Socket 1 para el envío de paquetes y recepción de confirmación**

Como se indica en el Código 3.1 la clase *SenseSocket* permite la comunicación por el puerto 1023, este valor está definido en la variable *LOCAL\_PORT* de la clase *Nodo*. Este socket permite el manejo de las confirmaciones de que el paquete enviado se recibió exitosamente, para lo cual el método *onPacket* se ejecuta cuando llega un paquete a dicho puerto, y realiza la asignación del valor de cero a la variable *ackNextHop*, esta es importante en la ejecución del algoritmo para la división de red.

```

/// <summary>
/// Esta clase maneja el socket en el puerto 1024
/// </summary>
public class SenseSocket2 : UDPsocket
{
    /// <summary>Constructor </summary>
    public SenseSocket2 ()
    {
        // Se vincula el socket con el puerto 1024
        this.bind (1024);
    }

    /// <summary>Recibe paquete </summary>
    /// <description>Permite el manejo del paquete recibido por el puerto
indicado</description>
    public override int onPacket (Packet packet)
    {
        uint len = packet.payloadLen;
        // creamos un buffer para la informacion recibida
        byte[] buf = (byte[])Util.alloca ((byte)len, Util.BYTE_ARRAY);
        // copiamos la información recibida en el buffer
        Util.copyData (packet.payloadBuf, packet.payloadOff, buf, 0,
len);

        if (Nodo.enruta1 == true) {
            // Copia los datos del buffer de llegada en el arreglo de
los datos

            Util.copyData (buf, 0, Nodo.datos, 0, len);
            Nodo.tipoDato = 1;
            Nodo.enruta1 = false;
        }
        try {

```

```

        //intercambiamos los datos del paquete para la respuesta
de confirmación
        packet.swap (len);
    } catch {
        return 0;
    }
    Util.copyData (buf, 0, packet.payloadBuf, packet.payloadOff,
len);

    //enviamos el paquete de contestación
    this.send (packet);
    return 0;
}
}

```

**Código 3.2 Socket 2 para recepción de datos**

En el Código 3.2 se indica la clase *SenseSocket2* la cual está vinculada al puerto 1024, por lo que este socket será el que reciba los datos de su respectiva red lógica. El método *onPacket* de esta clase es diferente al anterior, ya que, al recibir el paquete su información es copiada en la variable *datos* de la clase *nodo* y se cambia el valor de *tipoDato*. Esta variable, se explicará más adelante; finalmente ya copiada la información recibida se devuelve el paquete como contestación de confirmación de paquete recibido. La variable *enruta1* permite evitar el cambio de información mientras es procesada para su envío al siguiente nodo de la red lógica o al *Gateway* de la red 6LowPAN.

```

/// <summary>
/// Esta clase maneja el socket en el puerto 1025
/// </summary>
public class SenseSocket3 : UDPsocket
{
    /// <summary>Constructor </summary>
    public SenseSocket3 ()
    {
        // Se vincula el socket con el puerto 1025
        this.bind (1025);
    }

    /// <summary>Recibe paquete </summary>
    /// <description>Permite el manejo del paquete recibido por el puerto
indicado</description>
    public override int onPacket (Packet packet)
    {
        uint len = packet.payloadLen;
        // creamos un buffer para la información recibida
        byte[] buf = (byte[])Util.alloca ((byte)len, Util.BYTE_ARRAY);
        // copiamos la información recibida en el buffer
        Util.copyData (packet.payloadBuf, packet.payloadOff, buf, 0,
len);

        if (Nodo.enruta2 == true) {
            // Copia los datos del buffer de llegada en el arreglo de
los datosEnrutados
            Util.copyData (buf, 0, Nodo.datosEnrutados, 0, len);
            // Cambiamos el tipo de dato a 2

```

```

        Nodo.tipoDato = 2;
        Nodo.enruta2 = false;
        Nodo.enruta1 = false;
    }
    return 0;
}
}
}

```

**Código 3.3 Socket 3 para recepción de datos**

Se puede observar en el Código 3.3 la clase *SenseSocket3*, la cual maneja el puerto 1025, este puerto permite la comunicación entre nodos de diferente red lógica, este puerto se utiliza cuando se pierde comunicación con el siguiente nodo de la red lógica y necesita enviar información. En el método *onPacket* se realiza el copiado de la información recibida en la variable *datosEnrutados* de la clase *nodo*, se cambia el valor de la variable *tipoDato*, esta variable se explicará más adelante. Las variables *enruta1* y *enruta2* permiten evitar el cambio de información mientras es procesada para su envío al siguiente nodo de la red lógica o al *Gateway* de la red 6LowPAN.

En la división de la red se utilizan dos funciones: la primera realiza el cálculo del puerto destino y la segunda efectúa el cálculo de la dirección destino del siguiente salto. Las mismas que serán explicadas a continuación.

```

/// <summary>
    /// Comprobar el puerto.
    /// </summary>
    ///<description> Esta funcion hace el calculo del puerto del siguiente
salto </description>
    static void comprobarPuerto ()
    {
        // Dependiendo del valor del ackNextHop asignara un valor al
puerto de destino
        if (ackNextHop == 0) {
            portNextHop = 1024;
        } else {
            portNextHop = 1025;
        }
    }
}

```

**Código 3.4 Método para selección del puerto del siguiente salto**

Como se puede observar en el Código 3.4 el método *comprobarPuerto* realiza el cálculo del puerto dependiendo de la confirmación de los datos enviados, esto se realiza mediante la variable *ackNextHop* y forma parte del algoritmo para la división de redes lógicas.

```

/// <summary>
    /// Division de red.
    /// </summary>

```

```

    ///<description> Esta funcion hace la division de red, con el calculo
de la direccion del siguiente salto </description>
    static void divisionRed ()
    {
        // Verifica si la direccion del nodo es 1 o 2
        if (IdNodo == 1 || IdNodo == 2) {
            // Asignamos el valor de 0 a la direccion de destino
            destNextHop = 0;
        } else {
            // Calculamos la direccion de destino
            destNextHop = IdNodo - 2 + ackNextHop;
            // Asignamos el valor de 1 a la confirmacion
            ackNextHop = 1;
        }
    }
}

```

**Código 3.5 Método para la división de red**

En el Código 3.5 se encuentra el método *divisionRed* el cual realiza el cálculo de la dirección del siguiente salto, lo que se puede observar es que para los dos primeros nodos de la red no se necesitará de la confirmación de recepción del paquete, ya que, estos enviarán directamente el paquete al *Gateway* de la red 6LowPAN.

```

/// <summary>Evento MAC /// </summary>
/// <description>Permite el manejo de la lectura o generacion de
datos</description>
static int onAdcData (uint flags, byte[] data, uint len, uint info,
long time)
{
    if ((flags & Device.FLAG_FAILED) != 0) {
        // Si la lectura del sensor falla, los datos son
eliminados
        return -1;
    }
    // Creamos un paquete vacio
    Packet packet = Mac.getPacket ();
    if (inicio == true) {
        // Obtenemos la direccion corta del nodo sensor
        IdNodo = Mac.getMyShortAddr ();
        // Obtenemos el numero maximo de nodos
        maxMotas = Mac.getMaxMotes ();
        inicio = false;
        // Creamos los arreglos para el manejo de los datos
        datos = new byte[maxMotas];
        datosEnrutados = new byte[maxMotas];
    }
    // Llamamos a la funcion para verificar a que puerto enviar el
paquete
    comprobarPuerto ();
    // Llamamos a la funcion para la division de redes logicas
    divisionRed ();
    // Creamos un buffer auxiliar
    byte [] aux1 = new byte[maxMotas];
    // El switch maneja el tipo de dato que se desea enviar
    // tipo 1 para reenviar datos en la red logica a la que
pertenece
    // tipo 2 para reenviar datos de la otra red logica
}

```

```

switch (tipoDato) {
case 0:
// Colocamos en la primer posicioin del buffer auxiliar
la direccion del sensor
aux1 [0] = (byte)IdNodo;
// Definimos la direccion destino del paquete con la
direccion destino calculada
Address.fromSaddr (packet.dstaddr, 0, destNextHop);
if (enruta2 == false)
enruta2 = true;
break;
case 1:
// Llamamos a la funcion agregar
Agregar ();
// Copiamos los valores de la variable datos al buffer
auxiliar
Util.copyData (datos, 0, aux1, 0, (uint)datos.Length);
// Definimos la direccion destino del paquete con la
direccion destino calculada
Address.fromSaddr (packet.dstaddr, 0, destNextHop);
tipoDato = 0;
if (enruta2 == false)
enruta2 = true;
enruta1 = true;
break;
case 2:
// Llamamos a la funcion agregarExtras
AgregarExtras ();
// Copiamos los datos de la variable enrutados al buffer
auxiliar
Util.copyData (datosEnrutados, 0, aux1, 0,
(uint)datosEnrutados.Length);
// Definimos la direccion destino del paquete con la
direccion destino calculada
Address.fromSaddr (packet.dstaddr, 0, destNextHop);
tipoDato = 0;
enruta1 = true;
break;
}
// Define los valores de los puertos origen y destino en el
paquete
packet.srcport = LOCAL_PORT;
packet.dstport = portNextHop;
try {
// Creamos el paquete con los datos calculados
packet.create (portNextHop, LOCAL_PORT, (uint)aux1.Length
/*frst two ADC channels 0 & 1*/);
} catch {
// XXX
return -1;
}
// Identificamos el paquete
packetId = packet.getId ();
// Instanciamos los datos a enviar en el paquete
byte[] pybuf = packet.payloadBuf;
uint pyoff = packet.payloadOff;
// Copiamos nuestros datos en el paquete
Util.copyData (aux1, 0, pybuf, pyoff, (uint)aux1.Length /*frst
two ADC channels 0 & 1*/);
// Enviamos el paquete
socket.send (packet);

```

```

// Encendemos los LEDs del nodo sensor para verificacion del
envio del paquete
    onLeds (1);
    onLeds (2);
    return 0;
}

```

### Código 3.6 Método para envío de datos periódicos

```

En el /// <summary>Evento MAC /// </summary>

/// <description>Permite el manejo de la lectura o generacion de
datos</description>
static int onAdcData (uint flags, byte[] data, uint len, uint info,
long time)
{
    if ((flags & Device.FLAG_FAILED) != 0) {
        // Si la lectura del sensor falla, los datos son
        eliminados
        return -1;
    }
    // Creamos un paquete vacio
    Packet packet = Mac.getPacket ();
    if (inicio == true) {
        // Obtenemos la direccion corta del nodo sensor
        IdNodo = Mac.getMyShortAddr ();
        // Obtenemos el numero maximo de nodos
        maxMotas = Mac.getMaxMotes ();
        inicio = false;
        // Creamos los arreglos para el manejo de los datos
        datos = new byte[maxMotas];
        datosEnrutados = new byte[maxMotas];
    }
    // LLamamos a la funcion para verificar a que puerto enviar el
    paquete
    comprobarPuerto ();
    // LLamamos a la funcion para la division de redes logicas
    divisionRed ();
    // Creamos un buffer auxiliar
    byte [] aux1 = new byte[maxMotas];
    // El switch maneja el tipo de dato que se desea enviar
    // tipo 1 para reenviar datos en la red logica a la que
    pertenece
    // tipo 2 para reenviar datos de la otra red logica
    switch (tipoDato) {
    case 0:
        // Colocamos en la primer posicioin del buffer auxiliar
        la direccion del sensor
        aux1 [0] = (byte)IdNodo;
        // Definimos la direccion destino del paquete con la
        direccion destino calculada
        Address.fromSaddr (packet.dstaddr, 0, destNextHop);
        if (enruta2 == false)
            enruta2 = true;
        break;
    case 1:
        // LLamamos a la funcion agregar
        Agregar ();
        // Copiamos los valores de la variable datos al buffer
        auxiliar
        Util.copyData (datos, 0, aux1, 0, (uint)datos.Length);
    }
}

```

```

// Definimos la direccion destino del paquete con la
direccion destino calculada
Address.fromSaddr (packet.dstaddr, 0, destNextHop);
tipoDato = 0;
if (enruta2 == false)
    enruta2 = true;
enruta1 = true;
break;
case 2:
// Llamamos a la funcion agregarExtras
AgregarExtras ();
// Copiamos los datos de la variable enrutados al buffer
auxiliar
Util.copyData (datosEnrutados, 0, aux1, 0,
(uint)datosEnrutados.Length);
// Definimos la direccion destino del paquete con la
direccion destino calculada
Address.fromSaddr (packet.dstaddr, 0, destNextHop);
tipoDato = 0;
enruta1 = true;
break;
}
// Define los valores de los puertos origen y destino en el
paquete
packet.srcport = LOCAL_PORT;
packet.dstport = portNextHop;
try {
// Creamos el paquete con los datos calculados
packet.create (portNextHop, LOCAL_PORT, (uint)aux1.Length
/*frst two ADC channels 0 & 1*/);
} catch {
// XXX
return -1;
}
// Identificamos el paquete
packetId = packet.getId ();
// Instanciamos los datos a enviar en el paquete
byte[] pybuf = packet.payloadBuf;
uint pyoff = packet.payloadOff;
// Copiamos nuestros datos en el paquete
Util.copyData (aux1, 0, pybuf, pyoff, (uint)aux1.Length /*frst
two ADC channels 0 & 1*/);
// Enviamos el paquete
socket.send (packet);
// Encendemos los LEDs del nodo sensor para verificacion del
envio del paquete
onLeds (1);
onLeds (2);
return 0;
}

```

Código 3.6 se especifica cómo se realiza el envío de datos, ya que este se ejecutará periódicamente. En la primera parte del método, en el segundo condicional, se obtienen los datos iniciales los cuales se utilizan para el cálculo de la dirección del siguiente salto y el puerto del siguiente salto, también se inicializan los dos arreglos para los datos a enrutar dependiendo si es de la red lógica o es de la otra red lógica. En la siguiente parte del código tenemos los

métodos utilizados para la selección del puerto y la dirección destino del siguiente salto. Luego de ejecutar estos métodos se tiene la variable *tipoDato*, la cual permite tres tipos de datos que vamos a colocar en el paquete y son los siguientes:

Dato tipo 0: El dato tipo 0 tiene como objetivo enviar la dirección asignada al nodo sensor del siguiente salto dentro de la red 6LowPAN. Este paquete es enviado después de que un nodo sensor es encendido y agregado a la red, también se utiliza este paquete cuando no se recibió datos de otro nodo.

Dato tipo 1: Es cuando el nodo recibe datos y estos tienen que ser reenviados al siguiente nodo dentro de la red lógica a la que pertenece hasta llegar al nodo Gateway. A la información recibida se le agrega la dirección asignada al nodo sensor, luego es copiada en el paquete creado, el cual será enviado.

Dato tipo 2: Es cuando el nodo recibe datos y estos provienen de un nodo sensor que no pertenece a la misma red lógica. Estos datos deben ser reenviados, para evitar la pérdida de información por daño o falla de un nodo sensor. A la información recibida se le agrega la dirección asignada al nodo sensor, luego es copiada en el paquete creado, para ser enviado.

### **3.4 PRUEBAS Y RESULTADOS DEL ALGORITMO**

Para la verificación del funcionamiento del algoritmo en la red de sensores inalámbricos se utiliza el *Shell* de comandos de *Mote Runner*, donde se observan las características del paquete enviado al nodo de borde (*Gateway*), además se pueden apreciar entre otras cosas: las direcciones MAC de origen y destino, los puertos origen y destino, *payload* y finalmente el tiempo de arribo del paquete.

Esta herramienta permite visualizar el *payload* del paquete 6LowPAN enviado, en el *payload* se encuentran las direcciones de los nodos de la ruta por la cual se envía el paquete. El conocimiento de la ruta por la cual el paquete fue enviado permite verificar el funcionamiento correcto de la solución propuesta, en ambiente normal y con la simulación de una falla.

Para el prototipo de pruebas se utilizan sensores inalámbricos funcionando con 6LowPAN. Para cumplir con las condiciones de funcionamiento del algoritmo, los



sensores deben ser encendidos en orden, comenzando desde el nodo cercano al *Gateway* y sucesivamente los siguientes nodos, porque la asignación de las direcciones a los nodos sensores (*IdNodo*) es automática dependiendo del orden de encendido.

De las mediciones realizadas en campo abierto, el alcance máximo de los nodos de la red 6LoWPAN fue de 20m, por lo que la separación de los nodos deberá ser de 10m, para garantizar el correcto funcionamiento. Estas condiciones son las que existirán en ambientes reales ya que, por lo general, las infraestructuras lineales donde se encontrarán ubicados los nodos sensores se las pueden considerar libres de interferencias. El ambiente en el que se desarrollaron las pruebas es en un lugar abierto, el número de nodos sensores utilizados es de 4 y el nodo *Gateway*. Los nodos sensores envían información hacia el *Gateway*, donde el dato enviado por cada sensor es: la dirección (*IdNodo*) del nodo sensor, y dependiendo de la ruta por la cual se envía el paquete se irá incorporando la dirección (*IdNodo*) de los nodos sensores por los que haya pasado el paquete, como se indica en la Figura 3.22.

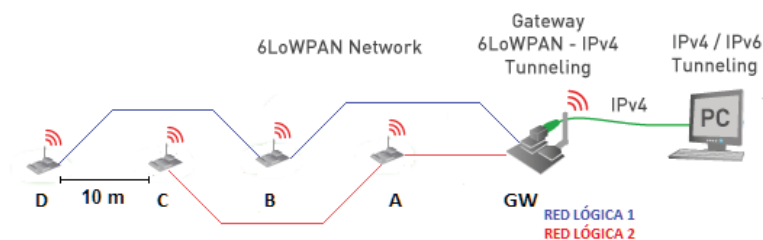


Figura 3.22 Prototipo de red 6LoWPAN

En la Figura 3.23 se indica cómo se conforma la red 6LoWPAN con dos nodos sensores, los cuales se irán encendiendo en orden, primero el nodo A y luego el nodo B. Estos nodos después de estar agregados ejecutan el algoritmo y envían un paquete al *Gateway* con su dirección corta en el *payload*.

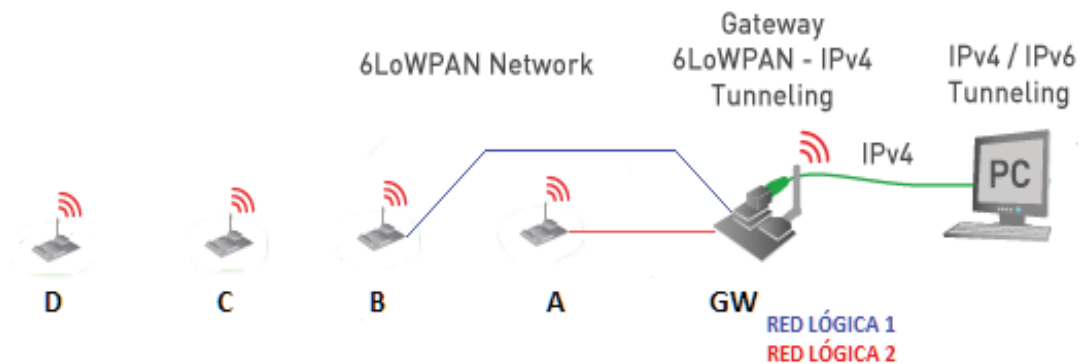


Figura 3.23 Topología lineal con 2 sensores inalámbricos agregados a la red 6LowPAN

La Figura 3.24 indica los paquetes que cada nodo envía, se puede observar la dirección MAC de cada nodo sensor y en el *payload* se aprecia las direcciones cortas.

```

Mote Runner Shell
time: 10:28:04.536'000
dstport: 4c
mote: 02-00-00-00-93-41-0B-13
srcport: 77
data: 00: 87

log-show
10:28:18.074'000 MRV6:INFO
Packet from mote to mote: 02-00-00-00-00-00-20 -> 02-00-00-00-93-41-0B-13
UDP: source:0|0|1|0:2000000000000002 dest:0|0|1|0:130B419300000002 srcport:1023 dstport:1024 payload:010000000000
10:28:18.074'000 SONORAN:INFO
Media event not handled by any socket: category: mote
evname: media
id: 163
time: 10:28:18.074'000
dstport: 400
mote: 02-00-00-00-00-00-20
srcport: 3ff
data: 00: 01 00 00 00 00 00 .....
10:28:18.287'000 SONORAN:INFO
Media event not handled by any socket: category: mote
evname: media
id: 167
time: 10:28:18.286'000
dstport: 4c
mote: 02-00-00-00-93-41-0B-13
srcport: 77
data: 00: 87

10:28:20.584'000 MRV6:INFO
Packet from mote to mote: 02-00-00-00-5D-F6-87-F7 -> 02-00-00-00-93-41-0B-13
UDP: source:0|0|1|0:F787F65D00000002 dest:0|0|1|0:130B419300000002 srcport:1023 dstport:1024 payload:020000000000
10:28:20.585'000 SONORAN:INFO
Media event not handled by any socket: category: mote
evname: media
id: 16c
time: 10:28:20.582'000
dstport: 400
mote: 02-00-00-00-5D-F6-87-F7
srcport: 3ff
data: 00: 02 00 00 00 00 00 .....

log-show

```

Figura 3.24 Visualización de los 2 nodos sensores agregados a la red 6LowPAN

En la Figura 3.25 se indica cómo se conforma la red 6LowPAN con 4 nodos sensores, para ello se encenderán los nodos restantes C y D en el orden respectivo. Estos nodos después de estar agregados a la red, ejecutan el

algoritmo y envían un paquete, el cual va cambiando dependiendo de la ruta por la cual se envía el paquete.

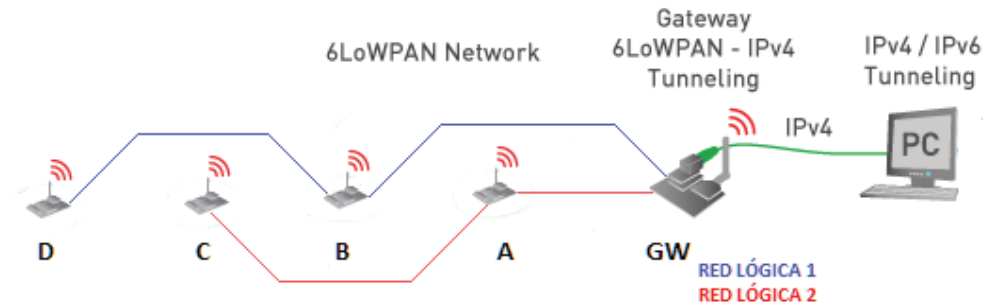


Figura 3.25 Topología lineal con 4 sensores inalámbricos agregados a la red 6LowPAN

La Figura 3.26 indica los paquetes que envían los nodos A y B, los mismos que van a permitir enviar las rutas de cada red lógica con topología lineal hacia el *Gateway*. Se puede observar el *payload* en donde se encuentran las 2 direcciones cortas de cada red lógica, por lo que concluimos que están operando correctamente las dos redes lógicas.

```

Mote Runner Shell
data: 00: 87
10:28:48.194'000 MRv6:INFO
Packet from mote to mote: 02-00-00-00-00-00-20 -> 02-00-00-00-93-41-0B-13
UDP: source:0|0|1|0:2000000000000002 dest:0|0|1|0:130B419300000002 srcport:1023 dstport:1024 payload:030100000000
10:28:48.195'000 SONORAN:INFO
Media event not handled by any socket: category: mote
evname: media
id: 187
time: 10:28:48.194'000
dstport: 400
mote: 02-00-00-00-00-00-20
srcport: 3ff
data: 00: 03 01 00 00 00 00 .....
10:28:48.286'000 SONORAN:INFO
Media event not handled by any socket: category: mote
evname: media
id: 18b
time: 10:28:48.285'000
dstport: 4c
mote: 02-00-00-00-93-41-0B-13
srcport: 77
data: 00: 87

log-show
10:28:49.424'000 MRv6:INFO
Packet from mote to mote: 02-00-00-00-5D-F6-87-F7 -> 02-00-00-00-93-41-0B-13
UDP: source:0|0|1|0:F787F65D00000002 dest:0|0|1|0:130B419300000002 srcport:1023 dstport:1024 payload:040200000000
10:28:49.426'000 SONORAN:INFO
Media event not handled by any socket: category: mote
evname: media
id: 190
time: 10:28:49.424'000
dstport: 400
mote: 02-00-00-00-5D-F6-87-F7
srcport: 3ff
data: 00: 04 02 00 00 00 00 .....
10:28:49.536'000 SONORAN:INFO
Media event not handled by any socket: category: mote
evname: media

```

Figura 3.26 Visualización de los 4 nodos sensores agregados a la red 6LowPAN

En la Figura 3.27 se observa el último paquete enviado por el nodo sensor A hacia el nodo *Gateway* antes de que falle el dicho nodo.

```

Mote Runner Shell
data: 00: 87
log-show
10:29:19.535'000 MRv6:INFO
Packet from mote to mote: 02-00-00-00-00-00-20 -> 02-00-00-00-93-41-0B-13
UDP: source:0|0|1|0:2000000000000002 dest:0|0|1|0:130B419300000002 srcport:1023 dstport:1024 payload:030100000000
10:29:19.537'000 SONORAN:INFO
Media event not handled by any socket: category: mote
evname: media
id: 1ab
time: 10:29:19.534'000
dstport: 400
mote: 02-00-00-00-00-00-20
srcport: 3ff
data: 00: 03 01 00 00 00 00 .....
10:29:19.539'000 MRv6:INFO
Packet from mote to mote: 02-00-00-00-5D-F6-87-F7 -> 02-00-00-00-93-41-0B-13
UDP: source:0|0|1|0:F787F65D00000002 dest:0|0|1|0:130B419300000002 srcport:1023 dstport:1024 payload:040200000000
10:29:19.540'000 SONORAN:INFO
Media event not handled by any socket: category: mote
evname: media
id: 1b1
time: 10:29:19.539'000
dstport: 400
mote: 02-00-00-00-5D-F6-87-F7
srcport: 3ff
data: 00: 04 02 00 00 00 00 .....
10:29:19.785'000 SONORAN:INFO
Media event not handled by any socket: category: mote
evname: media
id: 1b4
time: 10:29:19.784'000
dstport: 4c
mote: 02-00-00-00-93-41-0B-13
srcport: 77
data: 00: 87
10:29:20.035'000 SONORAN:INFO

```

Datos del nodo A

Datos del nodo B

Figura 3.27 Último funcionamiento de 4 nodos sensores en la WSN 6LowPAN

En la Figura 3.28 se puede apreciar que la red 6LowPAN se modificó, en vista de que el nodo sensor A dejó de funcionar, esto hace que el nodo sensor C envíe sus paquetes al nodo sensor B. Con lo descrito anteriormente se puede concluir que la red 6LowPAN se recuperó ante el fallo del nodo sensor antes mencionado.

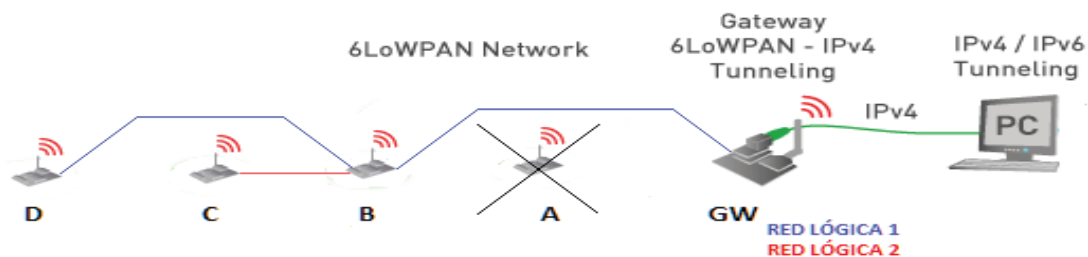


Figura 3.28 Topología lineal con falla de 1 sensor inalámbrico en la red 6LowPAN

En la Figura 3.29 se pueden observar los paquetes que el nodo sensor B está enviando hacia al Gateway. Los datos tanto del nodo sensor C y del nodo sensor D se observan en el *payload* con lo que se concluye que el algoritmo funciona correctamente.

```

data: 00: 12 20 00 00 00 00 00 00 .....
08: 02
10:29:47.106'000 MRv6:INFO
Packet from mote to mote: 02-00-00-00-5D-F6-87-F7 -> 02-00-00-00-93-41-0B-13
UDP: source:0|0|1|0:F787F65D00000002 dest:0|0|1|0:130B419300000002 srcport:1023 dstport:1024 payload:040200000000
10:29:47.107'000 SONORAN:INFO
Media event not handled by any socket: category: mote
evname: media
id: 1e2
time: 10:29:47.105'000
dstport: 400
mote: 02-00-00-00-5D-F6-87-F7
srcport: 3ff
data: 00: 04 02 00 00 00 00 .....
10:29:47.284'000 SONORAN:INFO
Media event not handled by any socket: category: mote
evname: media
id: 1e6
time: 10:29:47.284'000
dstport: 4c
mote: 02-00-00-00-93-41-0B-13
srcport: 77
data: 00: 87
10:30:00.900'000 MRv6:INFO
Packet from mote to mote: 02-00-00-00-5D-F6-87-F7 -> 02-00-00-00-93-41-0B-13
UDP: source:0|0|1|0:F787F65D00000002 dest:0|0|1|0:130B419300000002 srcport:1023 dstport:1024 payload:030200000000
10:30:00.901'000 SONORAN:INFO

```

Figura 3.29 WSN 6LowPAN recuperada después del fallo

Algo importante de recalcar de las siete figuras antes descritas, es que la red 6LowPAN tiene un tiempo de recuperación con el algoritmo, para esto se realizó un cuadro comparativo entre la red con algoritmo y sin algoritmo como se indica en la Tabla 3.1. Para la comprensión de los resultados obtenidos debemos tener en cuenta algunos aspectos: primero ocurre una reasignación en el momento de falla de un nodo en cualquiera de los dos casos, segundo en el algoritmo se necesita de un intento de mensaje para que en el segundo intento se recupere la red, también hay que agregar que existe mayor procesamiento en la red con algoritmo que sin algoritmo, esto es debido al parámetro utilizado para la comprobación del algoritmo. Otras consideraciones es que en la red sin algoritmo tenemos solo una red lógica en cambio en la red con el algoritmo tenemos dos redes lógicas, además la ruta que siguen los paquetes en la red sin algoritmo pueden cargar su ruta por un solo nodo en cambio en la red con algoritmo no hay sobrecarga en un nodo excepto cuando falla un nodo. Debido a los motivos antes mencionados el tiempo de recuperación de la red 6LowPAN sin algoritmo es menor al tiempo de la red con algoritmo.

Tiempo de Recuperación	ALGORITMO	SIN ALGORITMO
1	63 segundos	2 segundos
2	39 segundos	27 segundos
3	44 segundos	7 segundos
<b>Tiempo Promedio</b>	<b>48.67 segundos</b>	<b>12 segundos</b>

Tabla 3.1 Cuadro Comparativo de Tiempo de Recuperación de la Red con y sin Algoritmo

## CAPÍTULO IV

### 4 CONCLUSIONES Y RECOMENDACIONES

#### 4.1 CONCLUSIONES

- El tiempo de recuperación promedio de 48.67 segundos de la red es aceptable, ya que se utilizan en aplicaciones como por ejemplo tuberías de agua; en las que soportan el retardo de los paquetes.
- La utilización de 6LowPAN permitió una mayor facilidad en la división de la red, ya que, estaríamos haciendo la división a nivel lógico.
- El consumo de energía por nodo disminuye, ya que, los datos no deben ser procesados por todos los nodos intermedios entre el origen del dato y su destino, solo deben ser procesados por los nodos intermedios que pertenecen a su red lógica con topología lineal.
- La separación de los nodos será menor para el funcionamiento del algoritmo, ya que la separación máxima entre nodos es de 20m, pero para su correcto funcionamiento la separación entre nodos es de 10m, lo cual es la mitad de la separación máxima, por lo que implicará que se necesite una menor potencia para la transmisión y por ende un menor consumo de energía, pero aumentando el número de nodos.
- Para la implementación del algoritmo se deberá tomar en cuenta el área física, ya que, algunos factores externos, como por ejemplo la interferencia de otras tecnologías que trabajen en la banda de 2.4GHz, influirían en el alcance máximo de los sensores inalámbricos.
- El tiempo de recuperación de la red 6LowPAN sin algoritmo es menor al tiempo de recuperación de la red 6LowPAN con el algoritmo debido a que los nodos no realizan el procesamiento de datos que se utilizan para la verificación de que la red funciona correctamente o de que se ha recuperado la red.
- A nivel económico se incrementaría el costo por el aumento de nodos sensores, pero esto no influirá en las ventajas de utilizar sensores inalámbricos debido a que al usar otras tecnologías en topologías lineales el valor de su implementación sería mayor.

- La instalación y configuración de los nodos es sencilla, ya que solo se necesita poner a cierta distancia, y encenderlos de forma secuencial debido a la utilización de *mrpv6* la cual configura automáticamente la red 6LowPAN.

## **4.2 RECOMENDACIONES**

- Realizar la medición del consumo de energía, ya que, con el funcionamiento del algoritmo, se debe realizar una medición exacta del consumo de energía por parte de los nodos sensores inalámbricos pertenecientes a la red 6LowPAN.
- Realizar una modificación de la librería *mrpv6* utilizada, para que no ocurra la reasignación de otra dirección corta a los nodos sensores pertenecientes a la red; por los problemas de asociación que se puedan presentar.
- Para la realización de aplicaciones utilizando el algoritmo se recomienda tener en cuenta que los nodos utilizados en las pruebas son de evaluación.

## REFERENCIAS BIBLIOGRÁFICAS

- [1] T. Cooklev, *Wireless Communication Standards: A Study of IEEE 802.11, 802.15, 802.16*. IEEE Standards Association, 2004.
- [2] V. Garg, *Wireless Communications & Networking*. Morgan Kaufmann, 2010.
- [3] Z. Shelby and C. Bormann, *6LoWPAN: the wireless embedded internet*. John Wiley & Sons, 2009.
- [4] D. Culler and S. Chakrabarti, "6LoWPAN: Incorporating IEEE 802.15. 4 into the IP architecture," 2009.
- [5] Libelium Comunicaciones Distribuidas S.L., "Waspote Mote Runner Technical Guide wasp mote," *Tech. Guid.*, p. 85, 2015.
- [6] Dimitri John Ledkov, "LTS - Ubuntu Wiki." [Online]. Available: <https://wiki.ubuntu.com/LTS>. [Accessed: 09-Dec-2015].
- [7] Canonical Ltd, "1.1. What is Ubuntu?" [Online]. Available: <https://help.ubuntu.com/12.04/installation-guide/en.amd64/ch01s01.html>. [Accessed: 09-Dec-2015].
- [8] Anónimo, "FAQ | MonoDevelop." [Online]. Available: <http://www.monodevelop.com/help/faq/>. [Accessed: 09-Dec-2015].
- [9] Joerg Wunsch, "avrdude(1) - Linux man page." [Online]. Available: <http://linux.die.net/man/1/avrdude>. [Accessed: 09-Dec-2015].
- [10] Anónimo, "AVRDUDE - AVR Downloader/UploaDEr." [Online]. Available: <http://www.nongnu.org/avrdude/>. [Accessed: 09-Dec-2015].



## **ANEXOS**

ANEXO A: MANUAL DE INSTALACIÓN MOTE RUNNER

ANEXO B: CÓDIGO DE LA APLICACIÓN

## ANEXO A

### MANUAL DE INSTALACIÓN MOTE RUNNER

#### ARCHIVOS NECESARIOS PARA LA INSTALACIÓN:

- UBUNTU 12,04 LTS
- JAVA JRE/SDK :
- SOFTWARE MONO DEVELOP
- FIREFOX
- SOFTWARE MOTERUNNER IBM:

#### INSTALACIÓN UBUNTU 12,04 LTS

Ingresar al *BIOS* para seleccionar el booteo del dvd. Se aconseja que se ingrese en “algo más” en caso de tener el sistema operativo Windows en la computadora para ver las particiones, estas se crearán a gusto y necesidad del usuario, pero se recomienda mínimo para Linux 100GB.

- Si no se asigna un disco fat32 no importa, ya que de esa partición se encargará Windows.
- Una vez finalizada la instalación del sistema operativo Ubuntu 12.04LTS, se debe ingresar a la "Terminal".
- Cambiar al modo de súper usuario y ejecutar los siguientes comandos provistos en Código A-1, previos a la instalación de *Moterunner*

```
1 ubuntu@ubuntu:~$ sudo su
2 [sudo] password for ubuntu:
3 root@ubuntu:~# sudo apt-get update
4 root@ubuntu:~# sudo apt-get upgrade
5 root@ubuntu:~# sudo apt-get dist-upgrade
6 root@ubuntu:~# sudo apt-get install aptitude
7 root@ubuntu:~# sudo aptitude full-upgrade
```

**Código A-1 Instalaciones previas a instalar Moterunner**

## INSTALACIÓN DE COMPILADORES DE JAVA JRE/SDK

Para la instalación de estos elementos se puede realizar de dos maneras:

- a) Mediante el link oficial de descarga de Oracle que se presenta a continuación:

En la siguiente dirección se descarga el JDK:

- <http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html>.

En la siguiente dirección se descarga el JRE:

- <http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html>.

- b) La segunda opción es ejecutar los comandos como se muestra en el Código A-2

```

1 root@ubuntu:~# sudo apt-get update
2 root@ubuntu:~# sudo apt-get upgrade
3 root@ubuntu:~# sudo apt-get install aptitude
4 root@ubuntu:~# sudo aptitude full-upgrade
5 root@ubuntu:~# sudo apt-get install icedtea-7-plugin openjdk-7-
6 jre
root@ubuntu:~# sudo apt-get install openjdk-7-jdk

```

**Código A-2 Actualizaciones previas e instalación de Java**

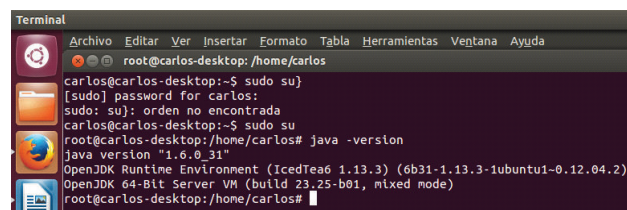
Para verificar que se han instalado satisfactoriamente los paquetes se ejecuta en la terminal los siguientes comandos:

```

1 root@ubuntu:~# java -version

```

**Código A-3 Comprobación de la instalación de Java**



```

Terminal
Archivo Editar Ver Insertar Formato Tabla Herramientas Ventana Ayuda
root@carlos-desktop: /home/carlos
carlos@carlos-desktop:~$ sudo su
[sudo] password for carlos:
sudo: su: orden no encontrada
carlos@carlos-desktop:~$ sudo su
root@carlos-desktop: /home/carlos# java -version
java version "1.6.0_31"
OpenJDK Runtime Environment (IcedTea6 1.13.3) (6b31-1.13.3-1ubuntu1-0.12.04.2)
OpenJDK 64-Bit Server VM (build 23.25-b01, mixed mode)
root@carlos-desktop: /home/carlos#

```

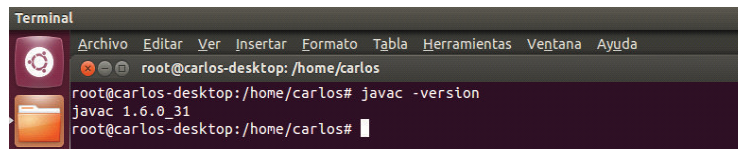
**Figura A-1 Comprobación de la versión de Java**

```

1 root@ubuntu:~# javac -version

```

**Código A-4 Comprobación de Java**



```

Terminal
Archivo Editar Ver Insertar Formato Tabla Herramientas Ventana Ayuda
root@carlos-desktop: /home/carlos
root@carlos-desktop: /home/carlos# javac -version
javac 1.6.0_31
root@carlos-desktop: /home/carlos#

```

Figura A-2 Comprobación de la versión de Java

## SOFTWARE MONO DEVELOP

Para la instalación de este *software*, que permitirá desarrollar aplicaciones en C#, se puede adquirir o descargar desde el Centro de software de Ubuntu, como se indica en la Figura A-3.

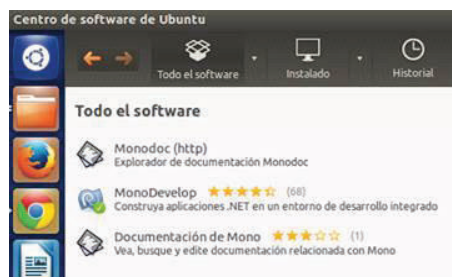


Figura A-3 Instalación del software MonoDevelop

Además es muy importante instalar además estos compiladores de C# para poder compilar código desarrollado para las aplicaciones.

El programa que se presenta en el Código A-5 es un compilador para C#.

```
1 root@ubuntu:~# sudo apt-get install mono-mcs
```

Código A-5 Instalador del compilador para C#

El programa que se presenta en el Código A-6 es un compilador para archivos binarios.

```
1 root@ubuntu:~# sudo apt-get install chicken-bin
```

Código A-6 Instalador de un compilador de archivos binarios

## SOFTWARE MOTERUNNER

Para su instalación es necesario descargar desde la siguiente dirección, dependiendo la distribución que se dispone de Ubuntu, sea de 32 o 64 bits. Para lo cual en la dirección indicada posteriormente se deberá aceptar las licencias

presentadas para poder acceder a la descarga:

<http://www.zurich.ibm.com/moterunner/downloads.html>

Descargar la versión 13.1 más reciente que solo está disponible para sistemas Linux de 64 bits por lo cual se recomienda obligatoriamente tener el Sistema Operativo de 64 bits, ya que la versión 13.1 maneja el hardware de los *Waspmote* de *Libelium* y como se mencionó anteriormente solo está disponible para la versión de 64 bits.

Una vez que se haya descargado, se procede a realizar lo que se muestra a continuación:

- Se descomprime el archivo descargado.
- Se ejecuta el archivo *setup* que está en la carpeta descomprimida, mediante la terminal se debe dirigir a la ubicación de dicha carpeta. Se encontrará tres archivos donde uno es el setup, para ejecutarlo se debe ejecutar el siguiente comando: **sudo ./setup** dentro de la carpeta que se encuentra el archivo ejecutable.
- Seguir los pasos indicados en el terminal, en los cuales se pedirá aceptar la licencia para la instalación de *Moterunner*.

Al finalizar se creará una carpeta llamada "moterunner" en la carpeta anteriormente descomprimida, la cual se procederá a mover a los archivos raíz del sistema con el siguiente comando:

```
1 root@ubuntu:~# cp -R "dirección completa de nuestra carpeta actual" /
```

Código A-7 Copia de la carpeta Moterunner al archivo raíz del sistema "/"

A continuación se presenta un ejemplo de la copia de la carpeta *Moterunner*.

```
1 root@ubuntu:~#cp -R /home/Escritorio/moterunner-install-temp/moterunner /
```

Código A-8 Ejemplo de copia de la carpeta Moterunner en el archivo raíz "/"

- Exportación de las variables de entorno del sistema mediante los siguientes comandos en la terminal.

Esto permitirá utilizar *Moterunner Shell* “*mrsh*”, el cual inicializará el servidor y el compilador *mrc*. Con la exportación de estos *path* o variables de entorno se podrá ejecutar estos comandos desde cualquier directorio que se encuentre el usuario, como se indica a continuación.

### Linux (32bits)

```
1 root@ubuntu:~#export PATH=/moterunner/linux/bin:$PATH
2 root@ubuntu:~#export LD_LIBRARY_PATH=/moterunner/linux/bin:$LD_LIBRARY_PATH
```

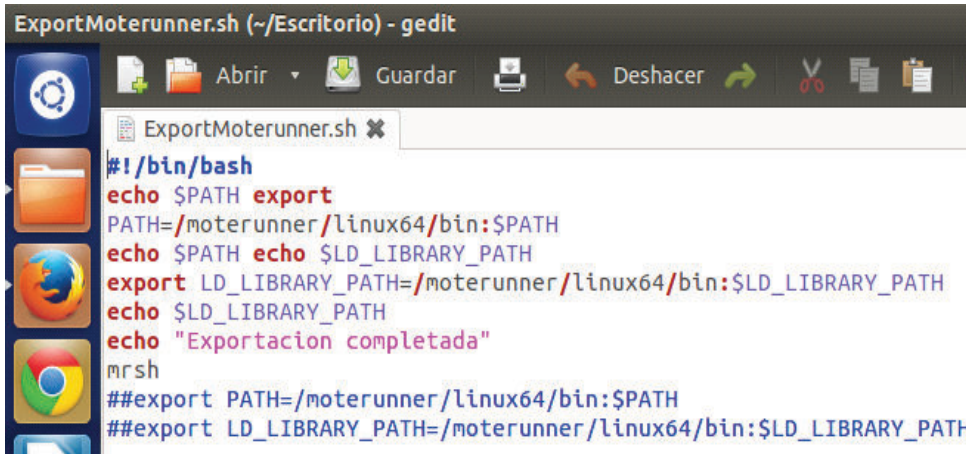
Código A-9 Exportación de variables de entorno para 32 bits

### Linux (64bits)

```
1 root@ubuntu:~#export PATH=/moterunner/linux64/bin:$PATH
2 root@ubuntu:~#export
LD_LIBRARY_PATH=/moterunner/linux64/bin:$LD_LIBRARY_PATH
```

Código A-10 Exportación de variables de entorno para 64 bits

Para la exportación de estas variables de entorno y la ejecución del servidor “*mrsh*” se ha realizado un pequeño Script llamado “*ExportMoterunner.sh*”, es importante la extensión *.sh*, que permitirá ejecutar el mismo y además debe tener todos los permisos para su ejecución, en el script se tiene lo siguiente:



```
ExportMoterunner.sh (~/Escritorio) - gedit
#!/bin/bash
echo $PATH export
PATH=/moterunner/linux64/bin:$PATH
echo $PATH echo $LD_LIBRARY_PATH
export LD_LIBRARY_PATH=/moterunner/linux64/bin:$LD_LIBRARY_PATH
echo $LD_LIBRARY_PATH
echo "Exportacion completada"
mrsh
##export PATH=/moterunner/linux64/bin:$PATH
##export LD_LIBRARY_PATH=/moterunner/linux64/bin:$LD_LIBRARY_PATH
```

Figura A-4 Script para exportación de las variables de entorno y ejecución del servidor Moterunner Shell MRSH

Para su comprobación se ingresa la siguiente dirección web <http://localhost:5000> en el navegador web *Mozilla*.

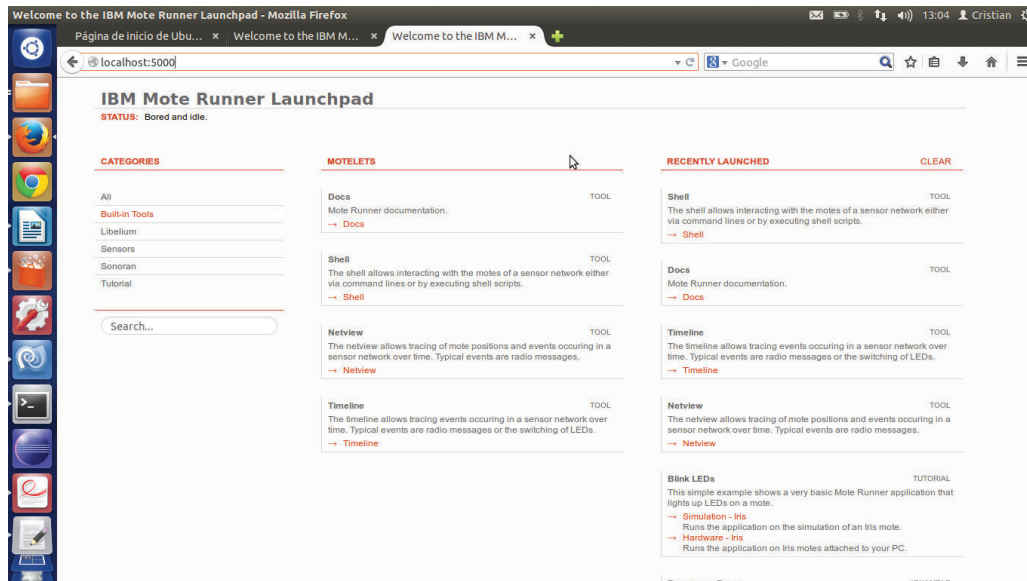


Figura A-5 Servidor MRSH

## AÑADIR LAS LIBRERIAS DEL MOTORUNNER EN MONODEVELOP

Se crea una nueva solución con los siguientes pasos:

- Archivo
- Nuevo
- Solución

En ubicación se selecciona en que directorio de nuestro PC se desea crear la solución, que para estas aplicaciones es una librería como se puede apreciar en la siguiente Figura A-6.

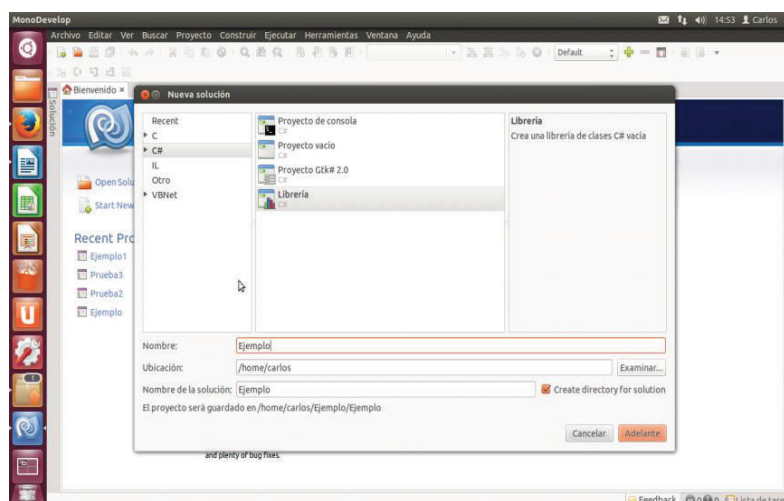


Figura A-6 Creando una solución en MonoDevelop

Se integra el soporte GTK# y la Integración con Unix.

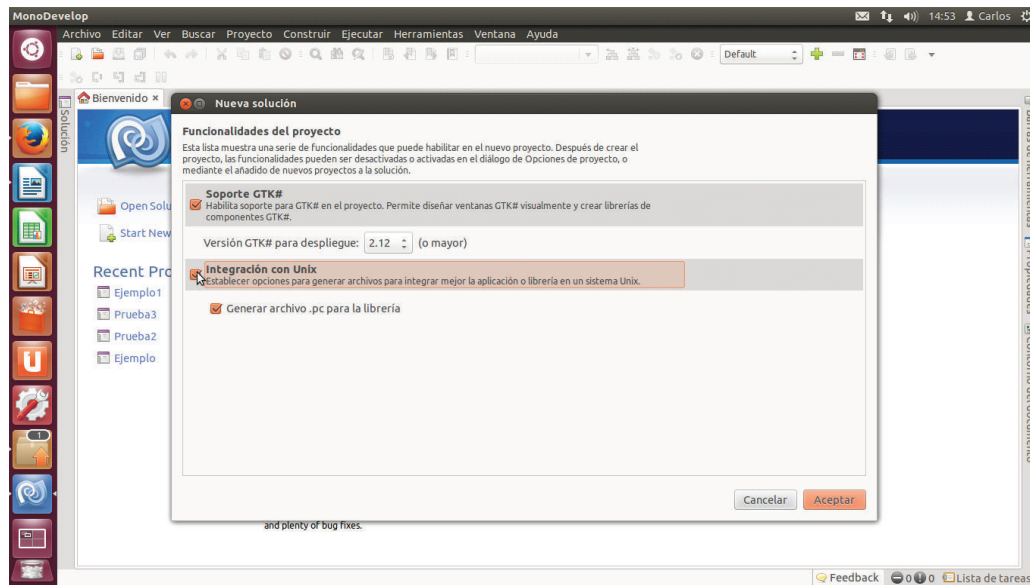


Figura A-7 Integración de GTK# y Unix

Se presenta la librería de la clase que se ha creado.

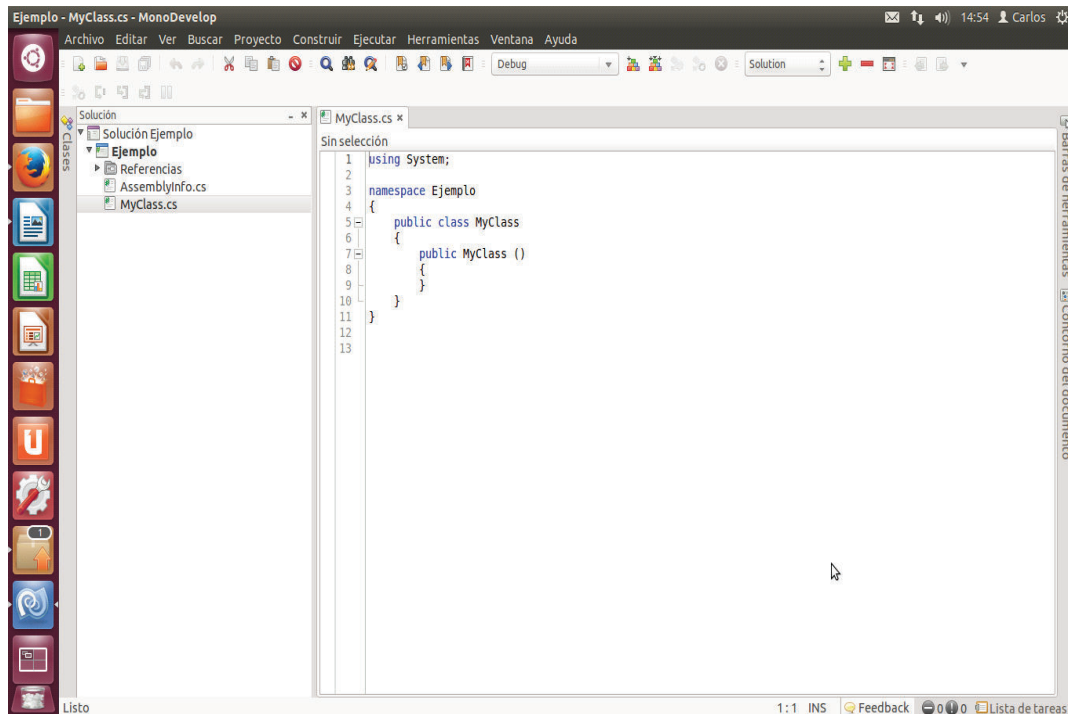


Figura A-8 Creación de la clase

Para agregar las librerías se debe dar click derecho en Referencias.



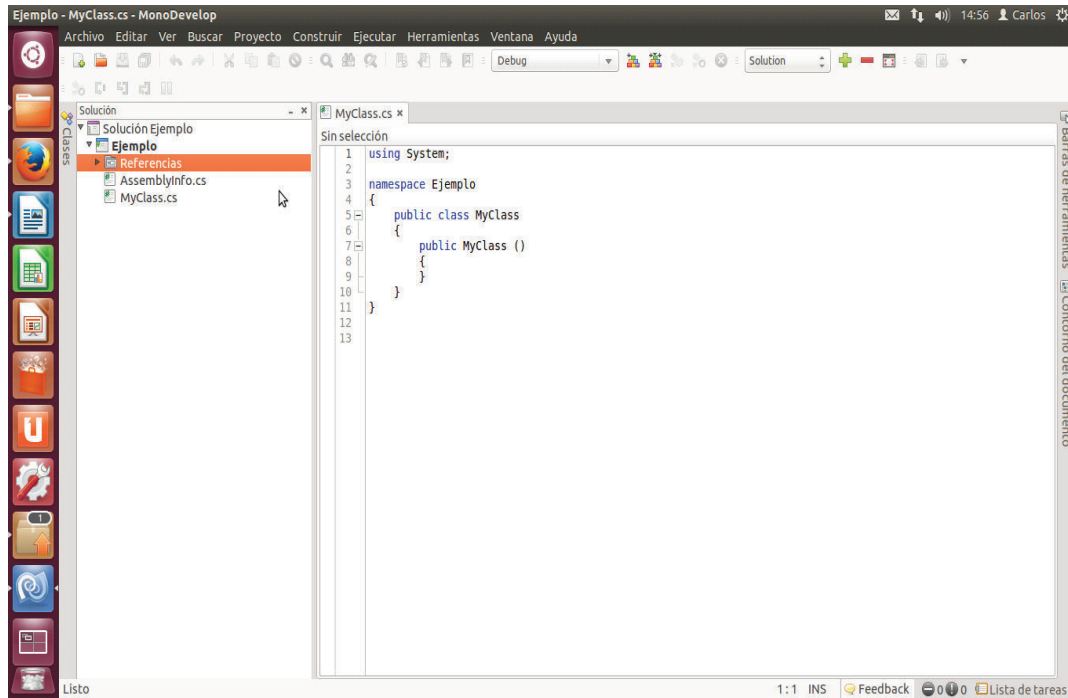


Figura A-9 Añadiendo las referencias de Moterunner

Se selecciona “editar referencias”.

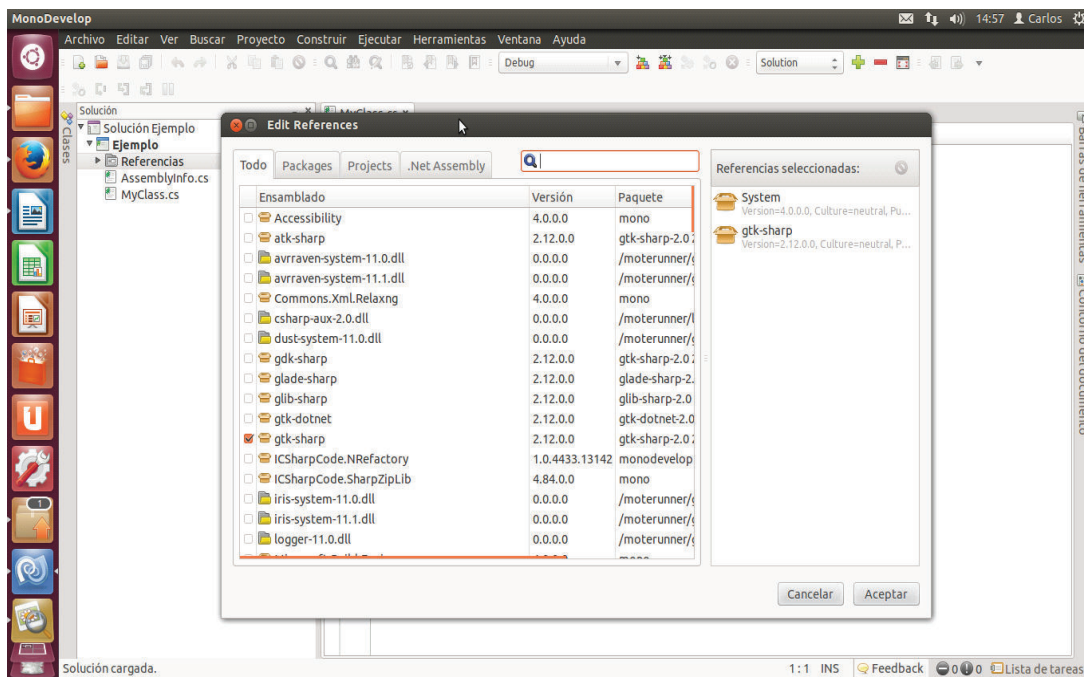


Figura A-10 Añadir referencias

Se selecciona la pestaña “.Net Assembly”.

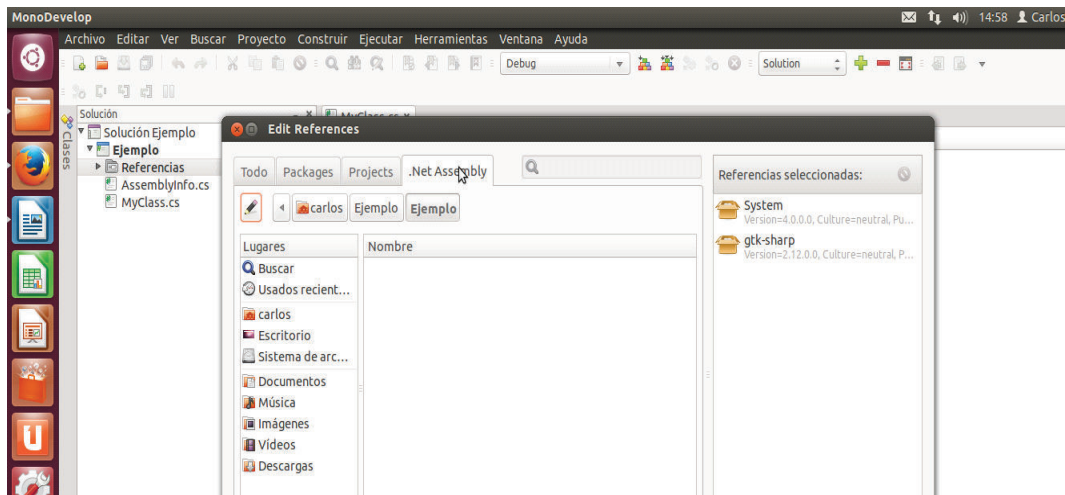


Figura A-11 Añadiendo librerías de Moterunner

Se busca el directorio en donde está instalado el moterunner “/moterunner”.

Se busca la carpeta *gac* y se selecciona todas las librerías cuya extensión son *.dll*.

Es importante indicar que si hay dos librerías con el mismo nombre, seleccionar la librería con el valor más alto, ya que como se puede observar en la Figura A- 12, dichas librerías poseen número de *assemblies*.

Si seleccionamos varios *assemblies* con el mismo nombre van a existir conflictos problemas al momento del desarrollo de la aplicación.

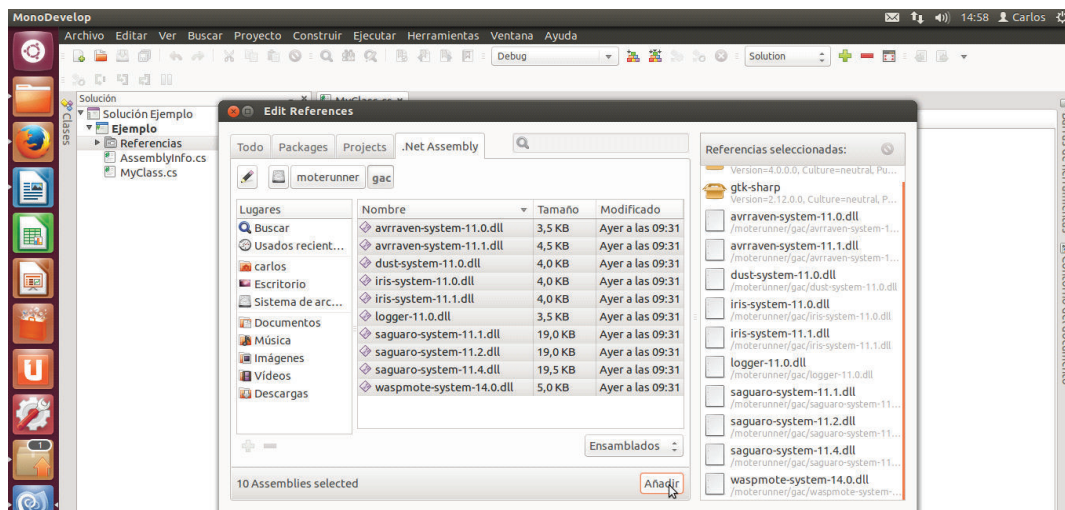


Figura A-12 Insertando librerías de Moterunner de la librería *gac*

Se busca la carpeta *lib* y se añade las librerías cuya extensión es *.dll*.

De la misma forma solo seleccionar las librerías con el número más alto, en el caso de que existan dos con el mismo nombre.

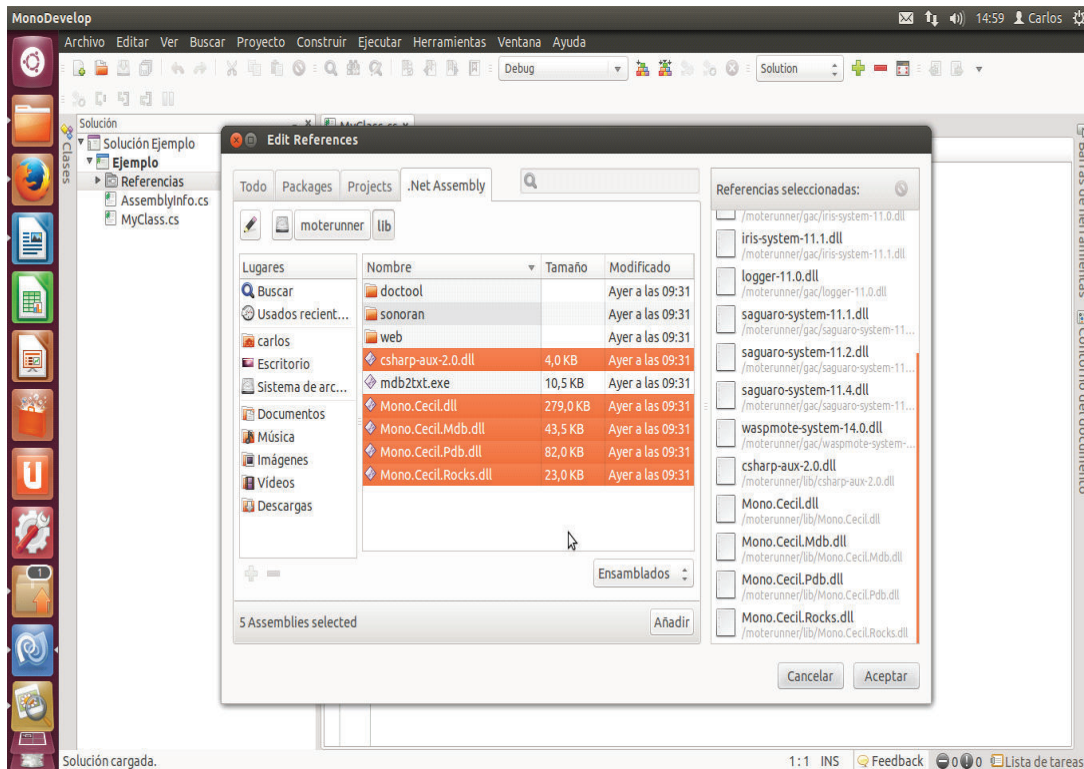


Figura A-13 Insertando librerías de la carpeta *lib*

Para comprobar la importación de las librerías se observará que se puede agregar a la clase creada, pulsando el botón “*Añadir*”.

Ahora para añadir un comando personalizado, que a su vez permite ensamblar la clase con extensión *.cs* directamente desde el programa MonoDevelop, esto se va a realizar con ayuda “*Comandos personalizados*” desde el MonoDevelop, el MRC permite crear los archivos *.sba*, *.sdx* y *.sxp*, los cuales se enviarán al nodo para su ejecución.

Para este propósito es necesario realizar lo que se indica a continuación:

- Se selecciona el proyecto al cual se desea agregar los comandos personalizados.

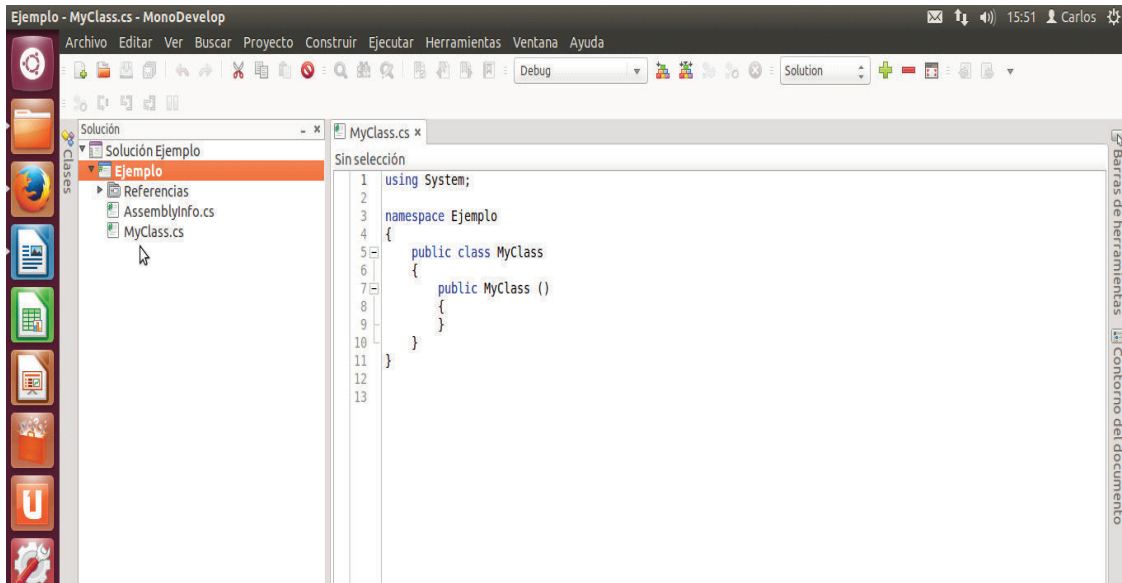


Figura A-14 Añadiendo compiladores dentro de MonoDevelop

- Click derecho, se selecciona “opciones”.

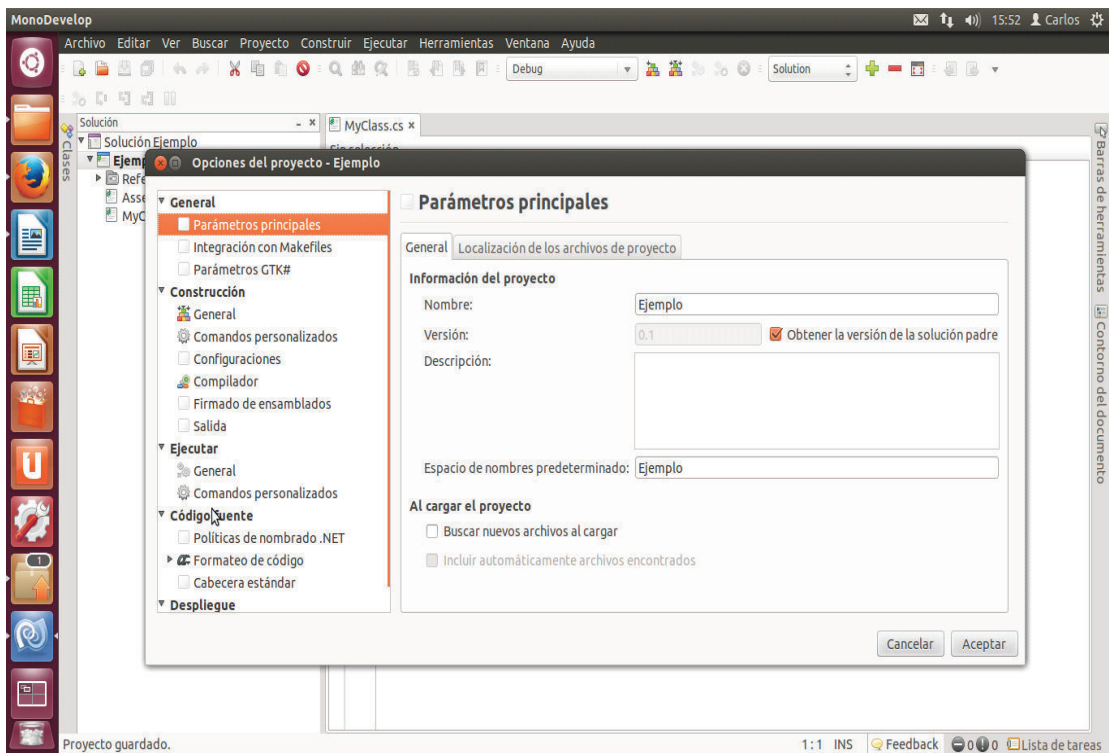


Figura A-15 Opciones del proyecto

- Se selecciona “comandos personalizados”.

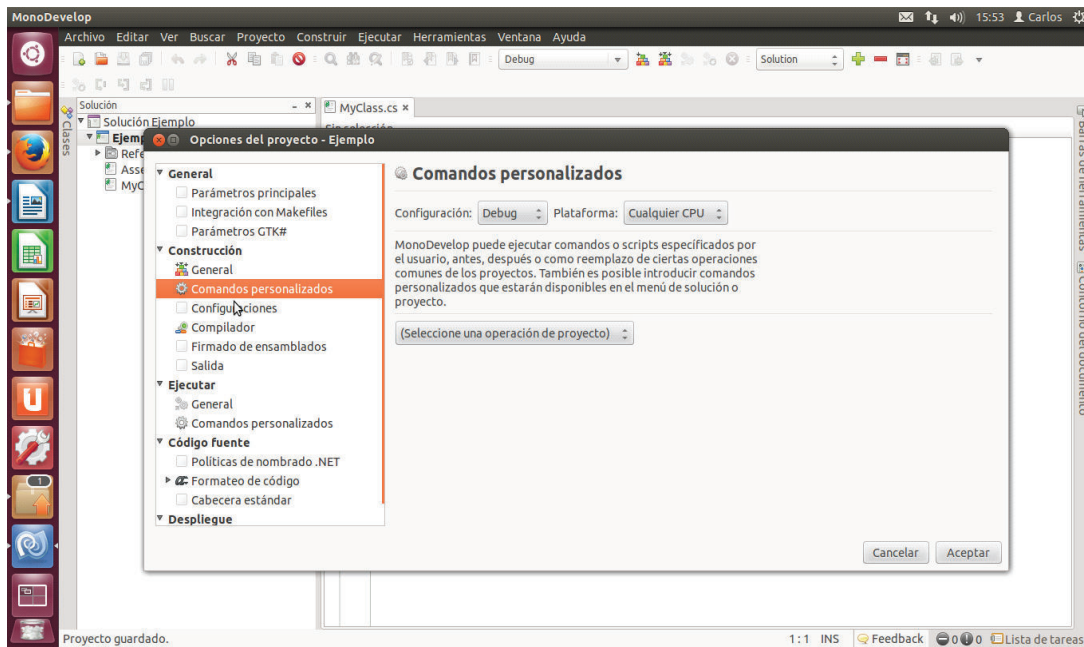


Figura A-16 Comandos personalizados

- En la opción seleccione una operación de proyecto, se selecciona “Después de construir”.

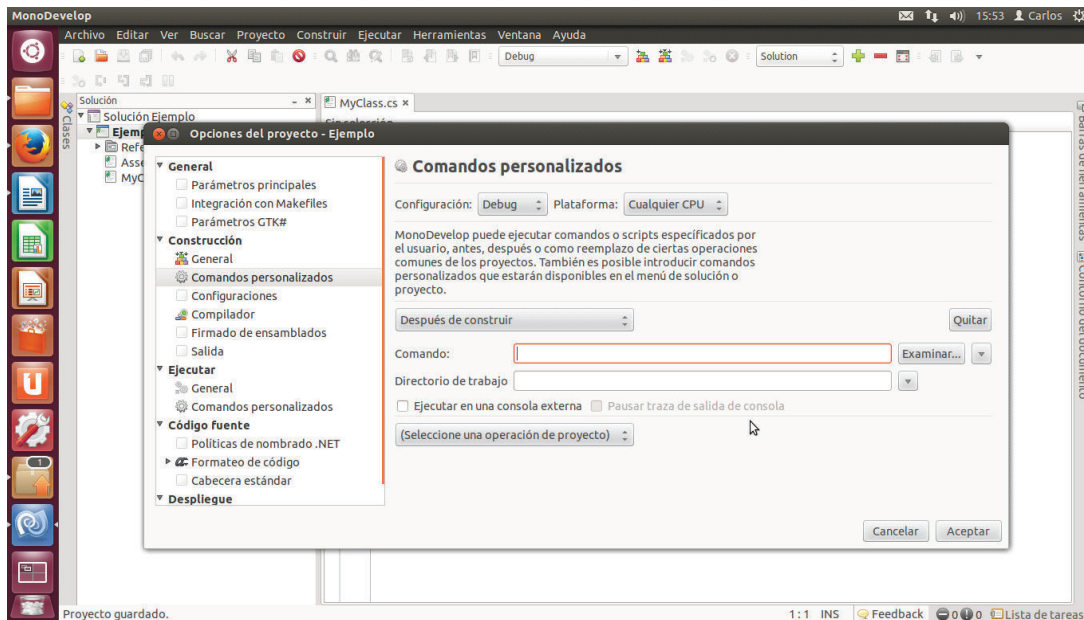


Figura A-17 Añadiendo comandos personalizados

- En comando se selecciona “examinar”.



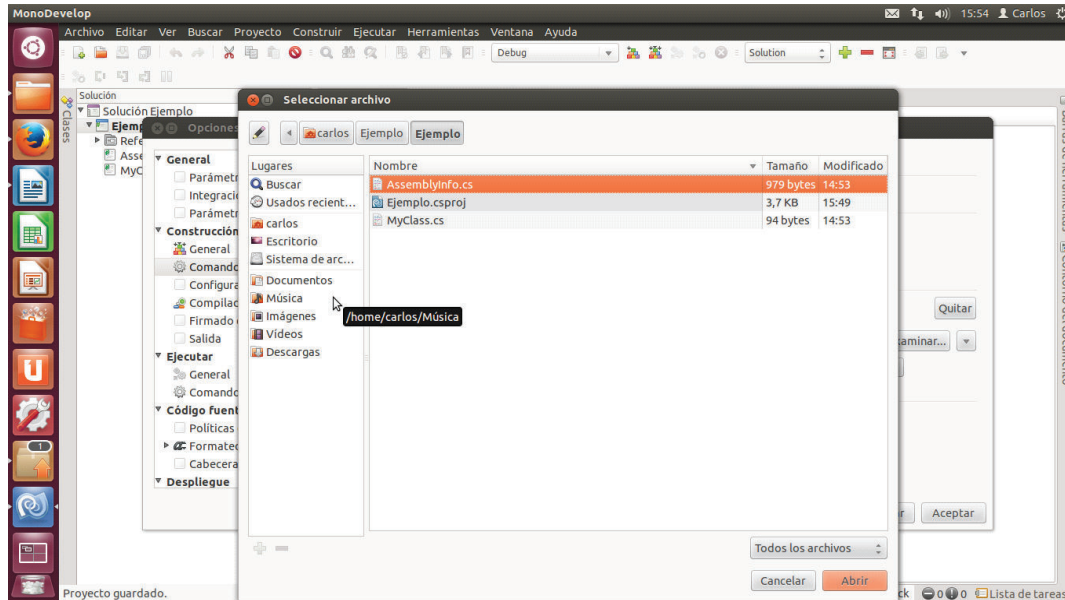


Figura A-18 Compilador MRC

- Se busca el directorio *moterunner* y en este se localiza al compilador *mrc*.
- En este caso la dirección es */moterunner/linux64/bin*.

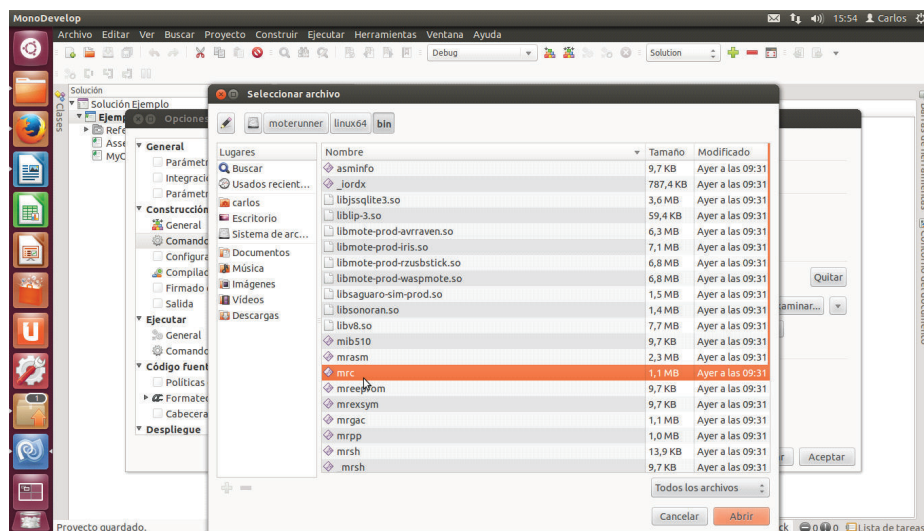


Figura A-19 Compilador MRC

En la caja de edición del comando personalizado, se completa dando un espacio y añadiendo lo siguiente:

```
1 --debug --assembly=Nombre-1.0 --ref=/moterunner/examples/mrv6/src/tmp/mrv6-lib-1.0
   NombreClase.cs
```

Código A-11 Comandos personalizados

**Nombre:** cualquier nombre

**NombreClase.cs:** El nombre de la clase que se compila con la extensión .cs

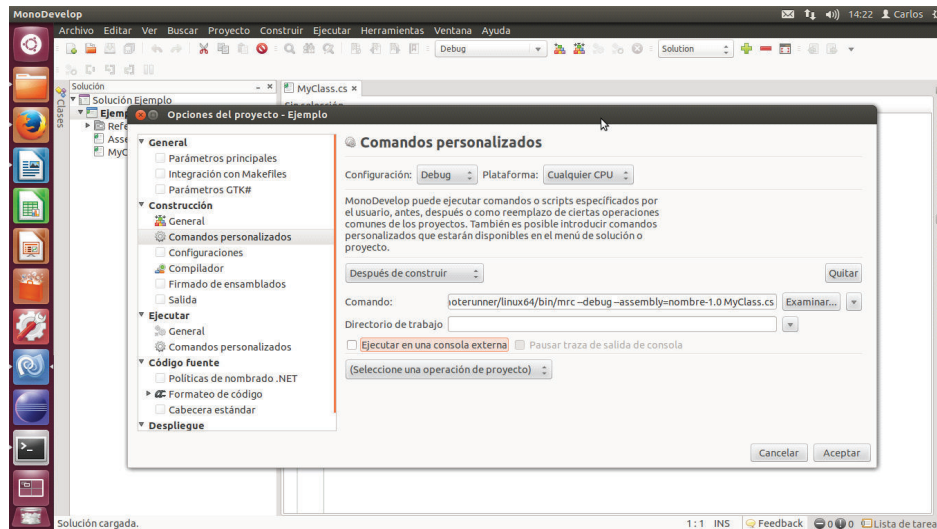


Figura A-20 Comandos personalizados

- En el directorio de Trabajo.
- Click en la pestaña que indica el Directorio de Trabajo y se selecciona Project Directory.

Click en las opciones para ejecutar en consola la depuración, para indicar que se ha compilado correctamente y se presente en consola posibles errores y éxito en la compilación.

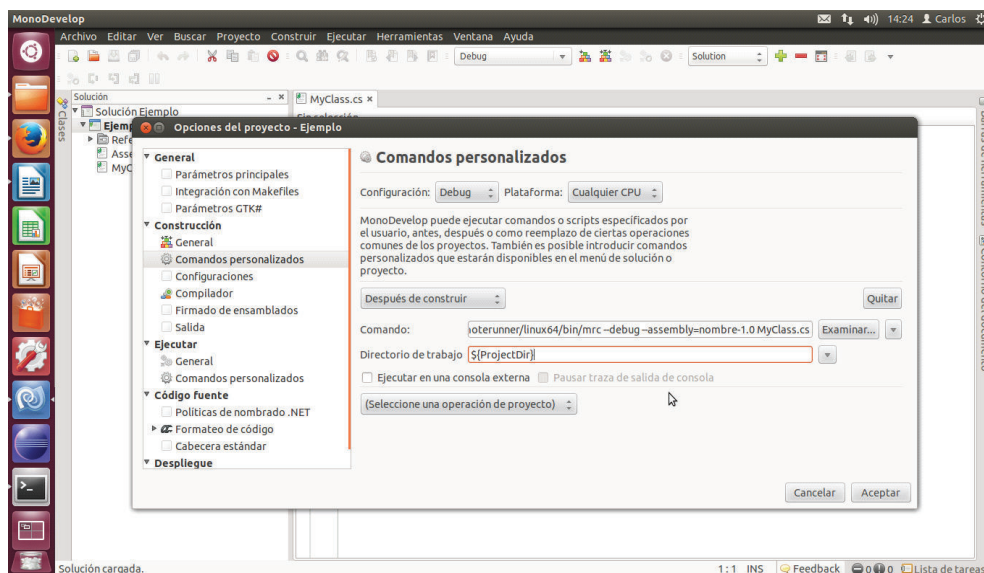


Figura A-21 Directorio actual es donde se va a compilar nuestro código

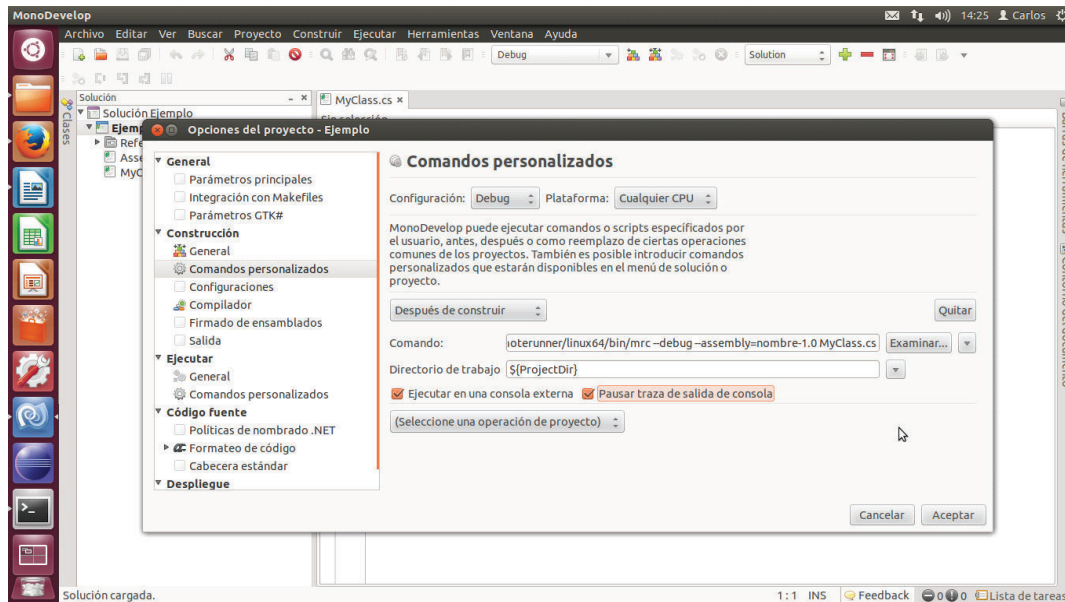


Figura A-22 Ejecución y depuración en la terminal

El proyecto se depura, en una ventana de consola, en la cual se indicará si se ejecutó perfectamente, para su verificación se buscará en la carpeta solución los archivos .sba, .sdx, .spx.

```

1 namespace com.ibm.saguaro.gateway.generic {
2     using com.ibm.saguaro.system;
3     using com.ibm.saguaro.mrv6;
4     using com.ibm.saguaro.util;
5     using com.libelium.rtc;
6     using com.libelium.common;
7     public class ReplySocket : UDPSocket {
8         internal static uint LOCAL_PORT = 1024;
9         internal static ReplySocket socket = new ReplySocket();
10        internal static ADC adc;
11        public ReplySocket() {
12            this.bind(LOCAL_PORT);
13        }
14        adc = new ADC();
15
16        adc.open(0x01,GPIO.NO_PIN,0, 0);
17    }
18    public override int onPacket(Packet packet) {
19        // An UDP packet has reached this mote
20        // Toggle LED 2
21        LED.setState(2, (byte)(LED.getState(2)^1));
22        // We read ADC channel 0
23        //uint adcRead = adc.readChannel(0);
24        // We store the ADC reading on a byte array
25        byte[] buf = new byte[2];
26        //Util.set16be(buf,0,adcRead);
27        // ADC value hexadecimal representation is transformed to ASCII.
28        // Doing so we can see the hexa value directly on Netcat
29        uint temp = (uint)RTC.GetInstance().getTemperature();
30        byte aux = (byte)temp;
31        buf[0] = aux;
32        buf = adc2Ascii(buf);
33        uint len = packet.payloadLen;
34    try {
35        packet.swap(len);
36    } catch {

```



```

37 return 0;
38 }
39         Util.copyData(buf, 0, packet.payloadBuf, packet.payloadOff, 4);
40 this.send(packet);
41 return 0;
42 }
43
44 private byte[] adc2Ascii(byte[] adcValue){
45     byte[] aux = new byte[4];
46     aux[0]=(byte)((adcValue[0] & 0xF0) >> 4);
47     aux[1]=(byte)(adcValue[0] & 0x0F);
48     aux[2]=(byte)((adcValue[1] & 0xF0) >> 4);
49     aux[3]=(byte)(adcValue[1] & 0x0F);
50
51     for(int i=0;i<4;i++){
52         if(aux[i] < 9){
53             aux[i] += 48;
54         }
55         else{
56             aux[i] += 55;
57         }
58     }
59 return aux;
60 }
61 }
62 }
63
64 public override int onPacket(Packet packet) {
65     // An UDP packet has reached this mote
66     // Toggle LED 2
67     LED.setState(2, (byte)(LED.getState(2)^1));
68     // We read ADC channel 0
69     //uint adcRead = adc.readChannel(0);
70     // We store the ADC reading on a byte array
71     byte[] buf = new byte[2];
72     //Util.set16be(buf,0,adcRead);
73     // ADC value hexadecimal representation is transformed to ASCII.
74     // Doing so we can see the hexa value directly on Netcat
75     uint temp = (uint)RTC.getInstance().getTemperature();
76     byte aux = (byte)temp;
77     buf[0] = aux;
78     buf = adc2Ascii(buf);
79     uint len = packet.payloadLen;
80     try {
81         packet.swap(len);
82     } catch {
83         return 0;
84     }
85     Util.copyData(buf, 0, packet.payloadBuf, packet.payloadOff, 4);
86     this.send(packet);
87     return 0;

```

#### Código A-12 Código para leer la temperatura del sensor del nodo

Para el momento de compilar el código implementado en el lenguaje de programación C#, es importante tener en cuenta todas las referencias utilizadas para la compilación, además si se va a desarrollar aplicaciones con los módulos externos (sensores), tener muy en cuenta el agregar las respectivas librerías.

Se puede descargar las librerías de los distintos sensores mediante este link:

<http://www.libelium.com/downloads/moterunner-1.0.zip>

A continuación se presentará un ejemplo al utilizar el sensor de temperatura del mote. Para lo cual se logró con el siguiente código:

Como se puede ver en la Figura A-23, se agregaron las referencias de las librerías con la versión más alta y solo se selecciona una de ellas como es el ejemplo de la librería *saguaro-system*, ya que existen las versiones 11.0, 11.1 y 11.4 por lo cual se deberá seleccionar la versión 11.4.

Además como se va a utilizar el sensor de temperatura se necesita las librerías de dicho sensor las cuales son en nuestro caso es “*wasp-rtc-1.0.dll*” y “*wasp-common-1.0*”

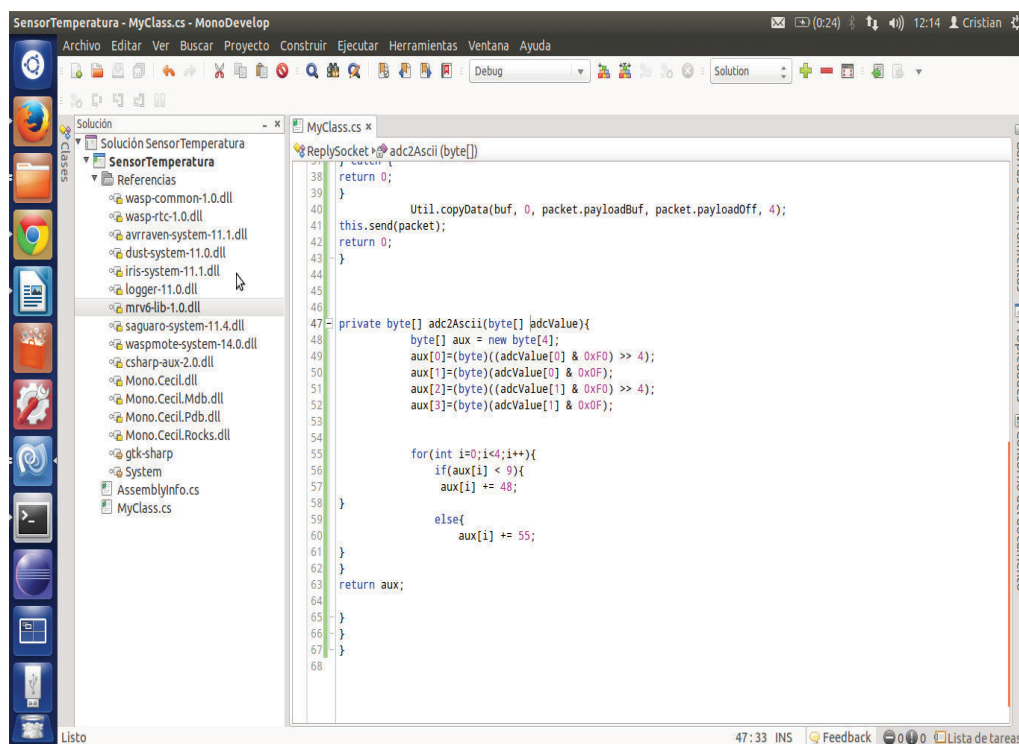


Figura A-23 Librerías *wasp-common* y *wasp-rtc* para lector de temperatura

Para la compilación del código implementado, se realiza mediante consola de la aplicación implementada, por lo cual el comando en la terminal sería el siguiente:

```

1 mrc --debug --assembly=SenseTemp-1.0 --ref=/moterunner/examples/mrv6/src/tmp/mrv6-lib-1.0
--ref=/home/cristian-espinoza/Escritorio/moterunner-1.0/src/com/libelium/rtc/wasp-rtc-1.0
--ref=/home/cristian-espinoza/Escritorio/moterunner-1.0/src/com/libelium/common/wasp-
common-
1.0/home/cristianespinoza/Escritorio/C#_Moterunner/SensorTemperatura/SensorTemperatura/My
Class.cs

```

Código A-13 Compilación de la aplicación del sensor de temperatura

```

root@ubuntu: /home/cristian-espinoza/Escritorio
root@ubuntu: /home/cristian-espinoza/Escritorio# cd /home/cristian-espinoza/Escritorio/
root@ubuntu: /home/cristian-espinoza/Escritorio# export PATH=/usr/local/bin:$PATH
root@ubuntu: /home/cristian-espinoza/Escritorio# export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH
root@ubuntu: /home/cristian-espinoza/Escritorio# mrc
usage: mrc [options] --assembly=NAME:R.N sourcefiles.(cs,java,sl) .. libdir .. (-r:lib | lib.sxp) ..

Compile a set of source files into a loadfile that can be uploaded onto Mote Runner notes.
Instead of a set of cs files a single DLL may be specified as well.
Instead of a set of java files a single JAR archive with compiled class file
may be specified.

mrc [options] [--out={with-sxp|DIR}] libdir .. lib.sxp ..

Create stub files for each listed sxp exchange file. It creates stubs
for Java and CSharp. The generated jar/dll stub files are placed
next to the respective sxp file or in the specified directory DIR.
Default storage policy is 'with-sxp'.

mrc [options] [--out=DIR] --doc lib.sxp [cssfiles..][htmfiles..][jsfiles..]

Create HTML documentation from the sxp file. The output file is
a single HTML file lib.htm stored next to the input file or in the
specified directory. Additional css and javascript files are
added to the generated output. The provided htm files define
documentation for application specific terms.

For more option details try: mrc --help
root@ubuntu: /home/cristian-espinoza/Escritorio# mrc --debug --assembly=SensorTemperatura-1.0 --ref=/moterunner/examples/mrv6/src/tmp/mrv6-lib-1
.0 --ref=/home/cristian-espinoza/Escritorio/moterunner-1.0/src/com/libellun/rtc/wasp-rtc-1.0 --ref=/home/cristian-espinoza/Escritorio/moterunne
r-1.0/src/com/libellun/common/wasp-common-1.0 /home/cristian-espinoza/Escritorio/C#_Moterunner/SensorTemperatura/SensorTemperatura
SensorTemperatura/ SensorTemperatura.sln
root@ubuntu: /home/cristian-espinoza/Escritorio# mrc --debug --assembly=SenseTemp-1.0 --ref=/moterunner/examples/mrv6/src/tmp/mrv6-lib-1
.0 --ref=/home/cristian-espinoza/Escritorio/moterunner-1.0/src/com/libellun/rtc/wasp-rtc-1.0 --ref=/home/cristian-espinoza/Escritorio/moterunne
r-1.0/src/com/libellun/common/wasp-common-1.0 /home/cristian-espinoza/Escritorio/C#_Moterunner/SensorTemperatura/SensorTemperatura/MyClass.cs
mrc: Assembly name too long (max 30 characters): <SensorTemperatura>
root@ubuntu: /home/cristian-espinoza/Escritorio# mrc --debug --assembly=SenseTemp-1.0 --ref=/moterunner/examples/mrv6/src/tmp/mrv6-lib-1.0 --ref
/home/cristian-espinoza/Escritorio/moterunner-1.0/src/com/libellun/rtc/wasp-rtc-1.0 --ref=/home/cristian-espinoza/Escritorio/moterunner-1.0/sr
c/com/libellun/common/wasp-common-1.0 /home/cristian-espinoza/Escritorio/C#_Moterunner/SensorTemperatura/SensorTemperatura/MyClass.cs
root@ubuntu: /home/cristian-espinoza/Escritorio# ls

```

Figura A-24 Compilación MRC en la terminal

A continuación se puede observar que en el escritorio se han creado los 3 archivos compilados.



Figura A-25 Creación de los tres archivos mediante el compilador MRC

## INSTALACIÓN DE FIRMWARE EN CADA NODO

El firmware se carga o instala a cada uno de los nodos, con ayuda del programador AVR. Cabe recalcar que el firmware es algo parecido al BIOS de nuestro PC, esto es solo para activar el hardware en cada nodo, para que éste pueda ser utilizado de la forma que requiera el usuario y puedan ser cargadas las aplicaciones desarrolladas.

Para esto, se puede realizar con el AVR Studio en Windows o mediante Linux gracias al AVRdude.

- Para el caso de Windows es importante realizar una descarga en el link indicado, pero antes de proceder a la descarga se tiene que registrar con un correo electrónico para la creación de un usuario.
- Link de Descarga: <http://www.atmel.com/tools/atmelstudio.aspx>

Para este proyecto se utilizará todas las herramientas provistas por el sistema operativo Ubuntu 12.04 LTS en Linux, para esto se procede a instalar el software “*avrdude*” desde el “Centro de Software de Ubuntu”, como se puede apreciar la búsqueda se realiza por su nombre, es decir *avrdude* y se procede con la descarga.

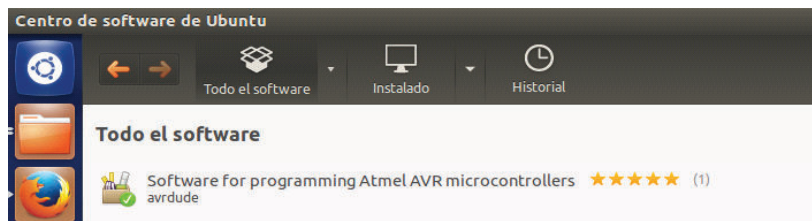


Figura A-26 Centro de Software Ubuntu, instalación de avrdude

## CARGANDO EL FIRMWARE EN EL NODO

- Primero antes de todo se conecta el nodo *Wasp mote* (Hardware) al cable de poder mediante el conector USB al adaptador de 120V o a la PC, este proceso recomienda el fabricante y debe hacerse por 24 horas. Luego de estar cargado el nodo, se enciende el *Wasp mote* moviendo los interruptores de encendido hacia a la izquierda como muestra en la Figura A-27.

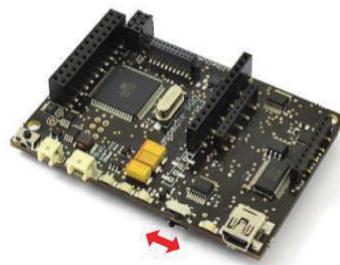


Figura A-27 Encendido del nodo Wasp mote

- El siguiente paso es conectar el programador AVR a la PC mediante el cable como muestra la Figura A-28.

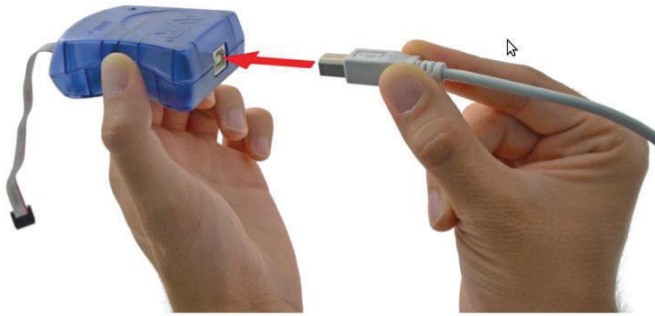


Figura A-28 Insertando el cable USB al quemador AVR

- Ahora el programador puede ser conectado al Waspote mediante el conector ICSP en la parte trasera del mismo.

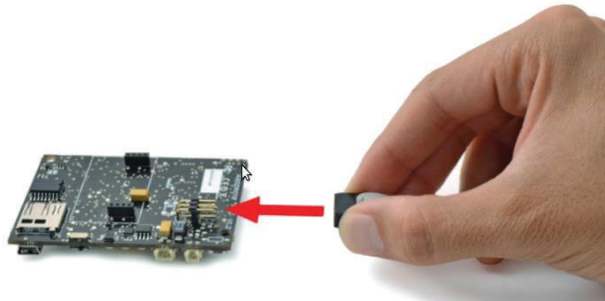


Figura A-29 Conector ICSP conectando al nodo

- Una vez conectado lo anteriormente mencionado, el siguiente paso es cargar el firmware al *Waspote*, como se ha mencionado en nuestro caso se hizo mediante consola en LINUX con los siguientes comandos:

El comando indicado en el Código A-14 permite establecer y quemar los respectivos fusibles que permitirán transmitir el Archivo .hex del firmware al nodo.

```
1 root@ubuntu:~# 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
```

Código A-14 Quemando los fusibles de los nodos

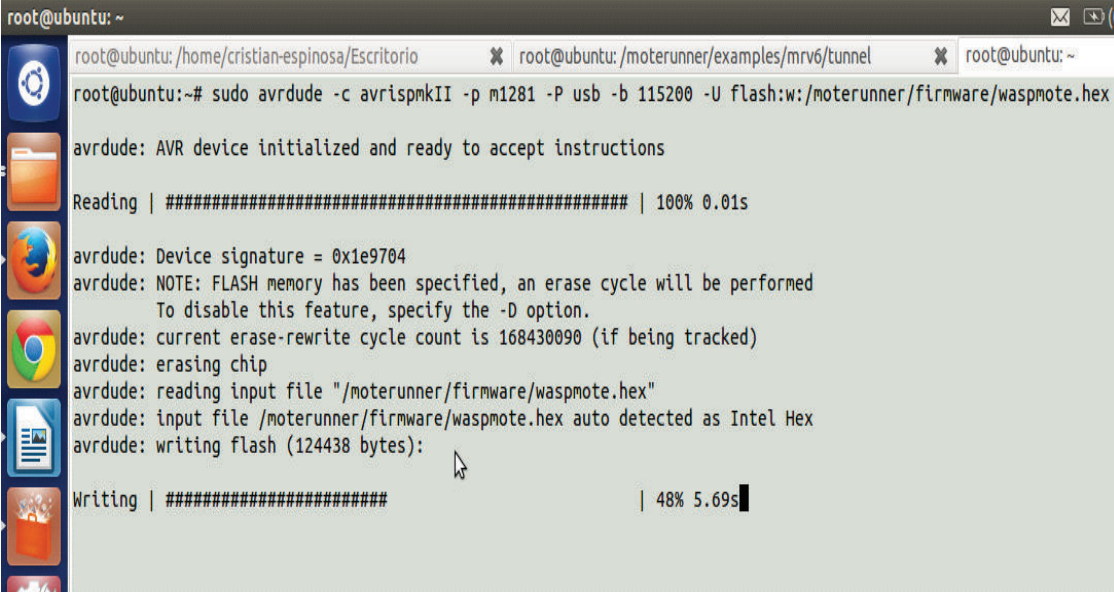
Finalmente se procede a instalar el archivo “.hex” del firmware que permitirá cargar en cada uno de los nodos, las aplicaciones desarrolladas anteriormente, para este propósito hay que trasladarse al directorio donde se encuentra el archivo hex, el mismo se encuentra en la siguiente ubicación “/moterunner/firmware/waspote.hex”. Cómo se indica en el Código A-15 y en las



Figuras A-30, A-31, A-32.

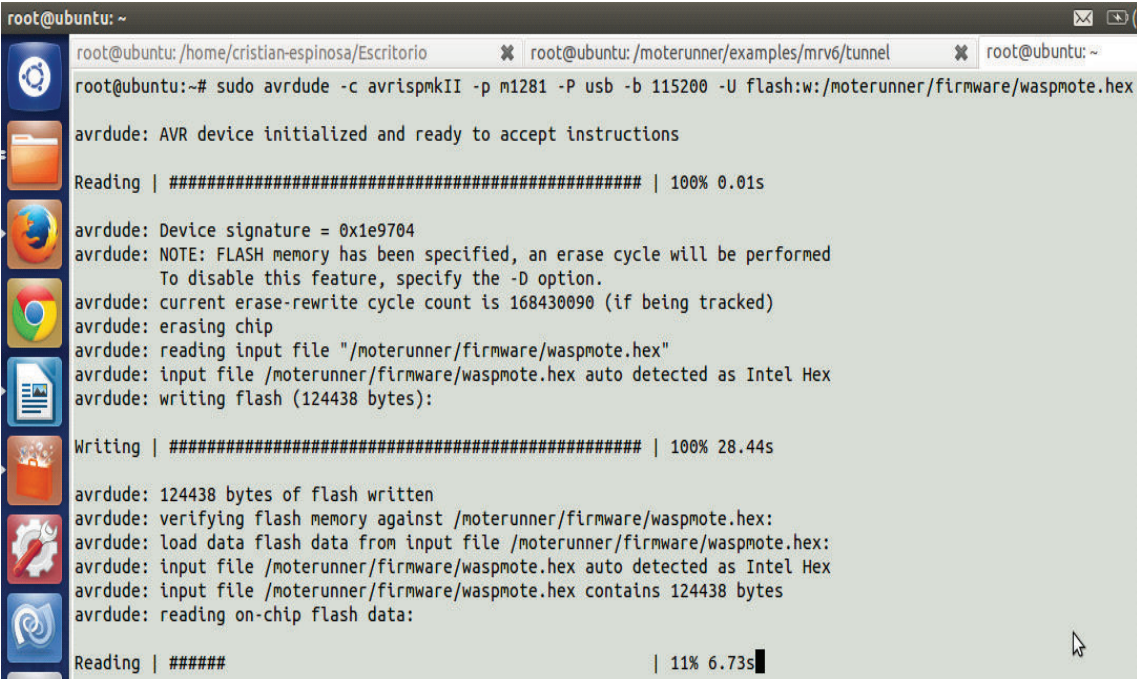
```
1 root@ubuntu:~# sudo avrdude -c avrispmkII -p m1281 -P usb -b 115200 -U
flash:w:/moterunner/firmware/waspote.hex
```

Código A-15 Quemando el firmware Waspote



```
root@ubuntu:~# sudo avrdude -c avrispmkII -p m1281 -P usb -b 115200 -U flash:w:/moterunner/firmware/waspote.hex
avrdude: AVR device initialized and ready to accept instructions
Reading | ##### | 100% 0.01s
avrdude: Device signature = 0x1e9704
avrdude: NOTE: FLASH memory has been specified, an erase cycle will be performed
        To disable this feature, specify the -D option.
avrdude: current erase-rewrite cycle count is 168430090 (if being tracked)
avrdude: erasing chip
avrdude: reading input file "/moterunner/firmware/waspote.hex"
avrdude: input file /moterunner/firmware/waspote.hex auto detected as Intel Hex
avrdude: writing flash (124438 bytes):
Writing | ##### | 48% 5.69s
```

Figura A-30 Ejecutando el primer comando para quemar los fusibles en los nodos



```
root@ubuntu:~# sudo avrdude -c avrispmkII -p m1281 -P usb -b 115200 -U flash:w:/moterunner/firmware/waspote.hex
avrdude: AVR device initialized and ready to accept instructions
Reading | ##### | 100% 0.01s
avrdude: Device signature = 0x1e9704
avrdude: NOTE: FLASH memory has been specified, an erase cycle will be performed
        To disable this feature, specify the -D option.
avrdude: current erase-rewrite cycle count is 168430090 (if being tracked)
avrdude: erasing chip
avrdude: reading input file "/moterunner/firmware/waspote.hex"
avrdude: input file /moterunner/firmware/waspote.hex auto detected as Intel Hex
avrdude: writing flash (124438 bytes):
Writing | ##### | 100% 28.44s
avrdude: 124438 bytes of flash written
avrdude: verifying flash memory against /moterunner/firmware/waspote.hex:
avrdude: load data flash data from input file /moterunner/firmware/waspote.hex:
avrdude: input file /moterunner/firmware/waspote.hex auto detected as Intel Hex
avrdude: input file /moterunner/firmware/waspote.hex contains 124438 bytes
avrdude: reading on-chip flash data:
Reading | ##### | 11% 6.73s
```

Figura A-31 Quemando el firmware Waspote en el nodo

```

root@ubuntu:~# sudo avrdude -c avr1ispnkII -p m1281 -P usb -b 115200 -U flash:w:/moterunner/firmware/waspnote.hex
avrdude: AVR device initialized and ready to accept instructions
Reading | ##### | 100% 0.81s
avrdude: Device signature = 0x1e9704
avrdude: NOTE: FLASH memory has been specified, an erase cycle will be performed
To disable this feature, specify the -D option.
avrdude: current erase-rewrite cycle count is 168430090 (if being tracked)
avrdude: erasing chip
avrdude: reading input file "/moterunner/firmware/waspnote.hex"
avrdude: input file /moterunner/firmware/waspnote.hex auto detected as Intel Hex
avrdude: writing Flash (124438 bytes):
Writing | ##### | 100% 28.44s
avrdude: 124438 bytes of flash written
avrdude: verifying flash memory against /moterunner/firmware/waspnote.hex:
avrdude: load data flash data from input file /moterunner/firmware/waspnote.hex:
avrdude: input file /moterunner/firmware/waspnote.hex auto detected as Intel Hex
avrdude: input file /moterunner/firmware/waspnote.hex contains 124438 bytes
avrdude: reading on-chip flash data:
Reading | ##### | 100% 66.61s
avrdude: verifying ...
avrdude: 124438 bytes of flash verified
avrdude: safenode: Fuses OK
avrdude done. Thank you.

```

Figura A-32 Finalización de la grabación del firmware en el nodo

Cabe recalcar que una vez instalado el firmware en cada *Waspnote*, ya no se necesita el programador AVR, ya que solo mediante el cable USB conectado al *Waspnote* a la PC y con los comandos del *Moterunner Shell*, se puede cargar el archivo *.sba* que se crea al momento de utilizar el compilador “mrc”. Al ejecutar este se nos va a crear tres archivos *.sdx*, *.sba*, *.sxp* el archivo que se envía al *Waspnote* es el *.sba* (*Saguaro Binary Assembly*) mientras que los otros dos son documentación del assembly.

## COMPILANDO APLICACIONES PARA LOS WASPMOTE UTILIZANDO EL SHELL

Esto se puede realizar directamente desde el software MonoDevelop o mediante el shell en la terminal ejecutando el siguiente comando:

```

1 root@ubuntu:~# mrc --debug --assembly=<name>-<major>.<minor> --<Dirección de la
librería a utilizar> <Dirección donde se encuentre nuestra clase.cs>

```

Código A-16 Ejemplo de la utilización del compilador MRC

**Nota:** Para la utilización de este comando se necesita exportar las variables de entorno, ya que solo funciona por cada terminal que se ejecute.

Como ejemplo de su utilización, actualmente la terminal se encuentra en el escritorio:

```

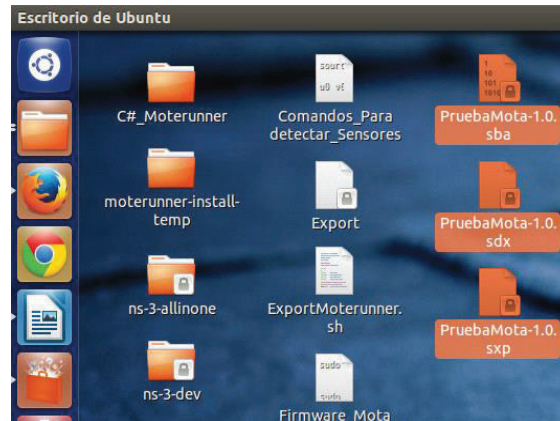
1 root@ubuntu:/home/cristian-espinoza/Escritorio# mrc --debug --assembly=SenseTemp-1.0 --
ref=/moterunner/examples/mrv6/src/tmp/mrv6-lib-1.0 --ref=/home/cristian-
espinoza/Escritorio/moterunner-1.0/src/com/libelium/rtc/wasp-rtc-1.0 --
ref=/home/cristian-espinoza/Escritorio/moterunner-1.0/src/com/libelium/common/wasp-
common-1.0 /home/cristian-

```

```
espinosa/Escritorio/C#_Moterunner/SensorTemperatura/SensorTemperatura/MyClass.cs
```

**Código A-17** Compilación utilizando el compilador MRC

Como se encuentra actualmente en el escritorio se van a crear los 3 archivos antes mencionados en el Escritorio como se puede ver en la imagen.



**Figura A-33** Archivos generados una vez que se ejecutó la compilación MRC

## CARGANDO EL CODIGO DE LAS APLICACIONES A LOS NODOS WASPMOTE

### a) Nodos Simulados

Para esto primeramente es necesario tener el servidor “*mrsh*” activo, es decir ejecutándose, como se va a ver en una página web, este servidor se puede visualizar en el navegador web en la siguiente dirección <http://localhost:5000/>.

En la consola se ejecuta el *script* *ExportMoterunner.sh* realizado, con el propósito de exportar el path y librería para poder ejecutar el servidor “*mrsh*” desde cualquier directorio que se esté ubicado, una forma de visualizar que se ha lanzado en el navegador el servidor, es verificándolo en una página web como se mencionó anteriormente.

```
root@ubuntu: /home/cristian-espinoza/Escritorio
root@ubuntu: /home/cristian-espinoza/Escritorio# ./ExportMoterunner.sh
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games export
/moterunner/linux64/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games echo
/moterunner/linux64/bin:
Exportacion completada
>
```

**Figura A-34** Ejecución del script



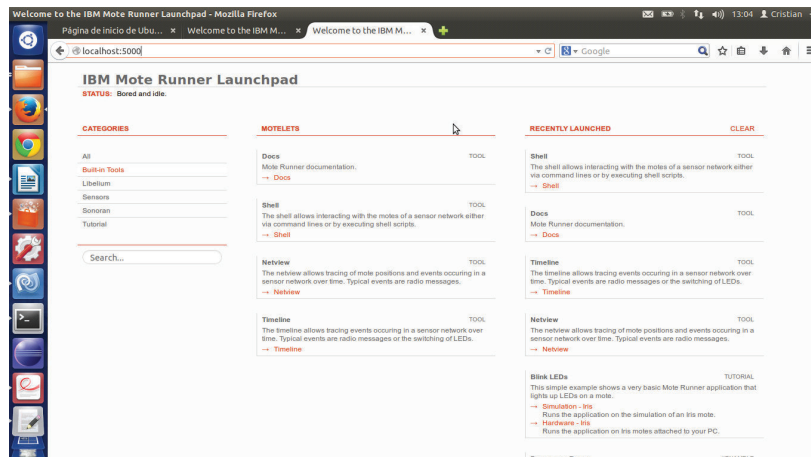


Figura A-35 Servidor MRSH

Algo importante en esta sección es que el servidor Web MRSH, tiene una opción de Shell que es prácticamente la misma que se está ejecutando en la terminal. Por lo tanto se puede ingresar comandos en cualquiera de estos dos. El shell del Servidor Web es mucho más amigable que el de la terminal; por lo cual se recomienda usar el shell del servidor web.

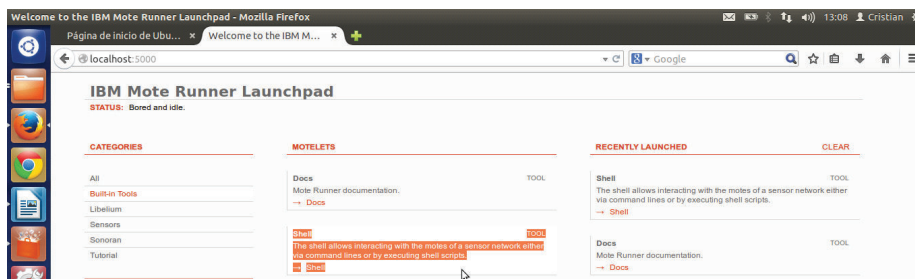


Figura A-36 Shell de MRSH

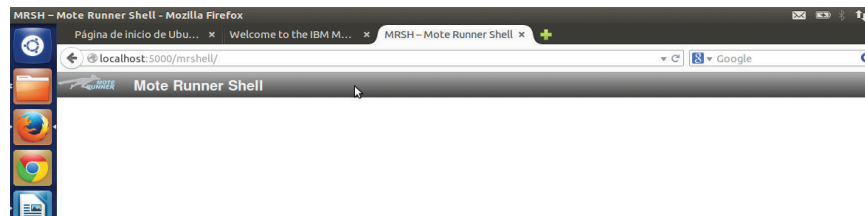


Figura A-37 Shell de MRSH

Ahora una vez que se encuentra dentro del shell de la terminal o del Servidor Web. Para la simulación se debe iniciar el proceso *saguaro* que nos permite especificar en qué puerto en el que se desea trabajar o por defecto está en el puerto 44044.

```
1 > saguaro-start
sag://localhost:44044
```

**Código A-18 Inicio de Saguaro para simular nodos**

Al ingresar el comando para el inicio del saguaro, se presentará que está corriendo en el local-host en el puerto 44044.

Ahora se establece la conexión con el proceso saguaro.

```
1 > saguaro-connect
```

**Código A-19 Conexión con el proceso Saguaro**

Se crea un *mote* con el proceso ya creado con; el `-d` se especifica qué tipo de hardware a simular.

```
1 > mote-create -d
waspmote
```

**Código A-20 Creación de un nodo waspmote**

Para cargar el programa a simular en el nodo se debe utilizar el *Mote Manager* (MOMA) el comando es el `"moma-load"`. Hay que recalcar que en esta sección, es que desde la misma *shell* del *Moterunner* se puede ejecutar comandos para ver en qué directorio se encuentra actualmente el administrador, similar a como estar desde el terminal, como cuando se ejecuta el comando `"pwd"`, también para ver un listado de los archivos se escribe el comando `"ls"` o `"ls -l"`, para un cambio de directorio el comando `"cd"`. Se menciona esto ya que al momento de cargar el programa debe desplazarse hasta el directorio donde se encuentre el programa que se desea cargar en el nodo, es decir donde se encuentre el archivo generado mediante el compilador MRC que es archivo con extensión `.sba`.

### Ejemplo:

En este ejemplo se carga la librería `"mrv6-lib-1.0"` y `"reply-1.0"` que están en diferentes directorios.

```
1 > cd /moterunner/examples/mrv6/src/tmp
2 > moma-load mrv6-lib-1.0
3 > cd /moterunner/examples/mrv6/reply
4 > moma-load reply-1.0
```

**Código A-21 Cargando librerías**

**Moma-load:** permite cargar el mismo archivo a varios waspmote o solo a un

nodo.

Para borrar el programa se utilizará el comando “*moma-delete*”, para ello se debe colocar el nombre del archivo cargado observando con el *moma-list*.

```
1 > moma-delete reply-1.0
```

**Código A-22 Borrar un programa**

Para conocer todos los comandos y su sintaxis colocar.

```
1 >?
```

**Código A-23 Para ver todos los comandos aceptados por la plataforma**

## b) Nodos Físicos

- Se conecta directamente con el cable mini USB desde la PC al nodo.
- Asegurarse que el nodo esté encendido y no haya nodos creados, para esto verifique con el comando *moma-list*, y que no esté funcionando.
- Si se conoce el puerto USB en el cual está el Nodo solo se utiliza el siguiente comando:

```
1 > mote-create -p /dev/ttyUSB0
```

**Código A-24 Conexión del nodo físico con el servidor MRSH**



```
root@ubuntu: /home/cristian-espinoza/Escritorio
> mote-create -p /dev/ttyUSB0
02-00-00-00-65-1F-EE-15
>
```

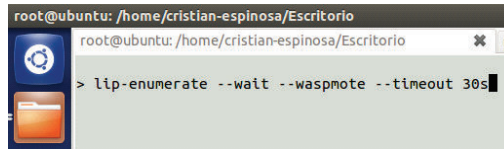
**Figura A-38 Conexión del nodo con el servidor MRSH**

Si no se conoce el puerto entonces se procede a realizar lo siguiente:

- Se desconecta el cable USB de la PC y se ingresa el siguiente comando en el Shell.
- Es importante seguir los pasos antes de ejecutar *lip-enumerate*.

```
1 >lip-enumerate --wait --waspmote --timeout 30s
```

**Código A-25 Comando para verificar en que puerto USB se conectó el nodo**



```

root@ubuntu: /home/cristian-espinoza/Escritorio
> lip-enumerate --wait --waspmote --timeout 30s

```

Figura A-39 LIP enumerate

```

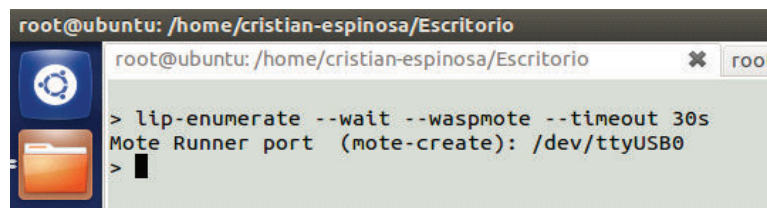
1 > lip-enumerate --wait --waspmote --timeout 30s
2 Mote Runner port (mote-create): /dev/ttyUSB0

```

Código A-26 Verificación del puerto USB

Se procede a conectar el USB a la PC y detectará en que puerto estará conectado.

Como se aprecia se ha detectado que el nodo se encuentra en `/dev/ttyUSB0`



```

root@ubuntu: /home/cristian-espinoza/Escritorio
> lip-enumerate --wait --waspmote --timeout 30s
Mote Runner port (mote-create): /dev/ttyUSB0
>

```

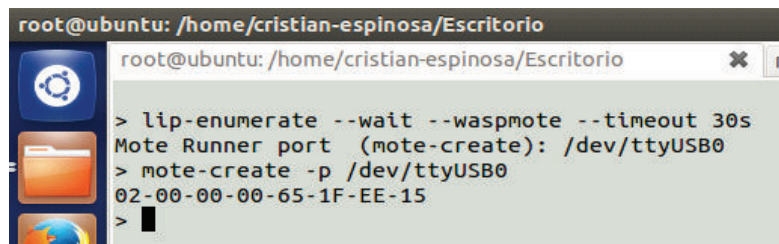
Figura A-40 Conexión del Nodo

Una vez localizado el puerto lo siguiente a realizar es conectar el nodo con el siguiente comando:

```

1 > mote-create -p /dev/ttyUSB0
2

```



```

root@ubuntu: /home/cristian-espinoza/Escritorio
> lip-enumerate --wait --waspmote --timeout 30s
Mote Runner port (mote-create): /dev/ttyUSB0
> mote-create -p /dev/ttyUSB0
02-00-00-00-65-1F-EE-15
>

```

Figura A-41 Conexión con el Nodo

Ahora para cargar el archivo se debe mover al directorio en el cual se encuentre la librería y se procede a cargar la librería `mr6-lib` y el ejemplo del `reply` con el `moma-load`.

```

1 > cd /moterunner/examples/mrv6/src/tmp
2 > moma-load mr6-lib-1,0
3 > cd /moterunner/examples/mrv6/reply
4 > moma-load reply-1.0

```

Código A-27 Ejemplo carga de librerías

## DEPURACIÓN DE APLICACIONES UTILIZANDO EL PLUGIN DE ECLIPSE

Se puede instalar el Eclipse (Entorno de desarrollo integrado Eclipse) mediante el Centro de Software de Ubuntu.

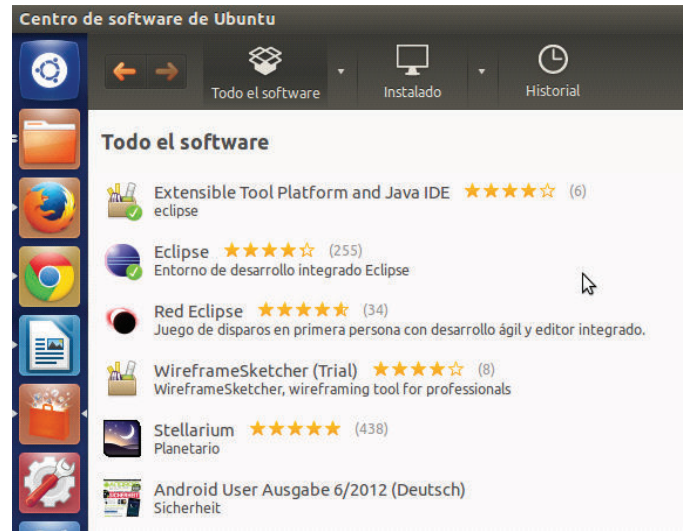


Figura A-42 Centro de Software de Ubuntu, descarga de Eclipse

Una vez instalado se abre el programa instalado, que es el Eclipse.

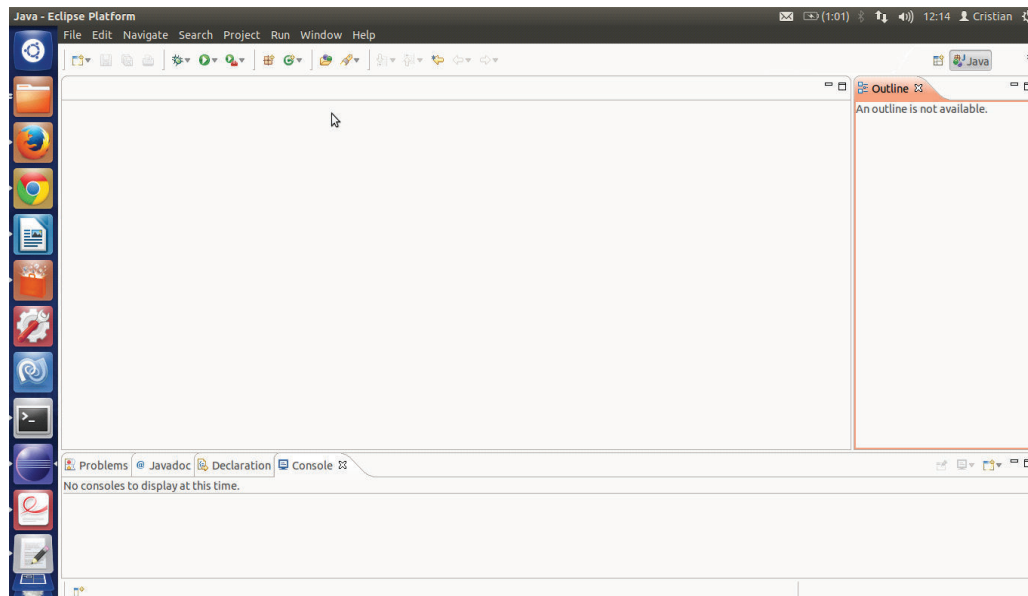


Figura A-43 IDE Eclipse

- Se da un click a la pestaña *Help* e *Install New Software*.

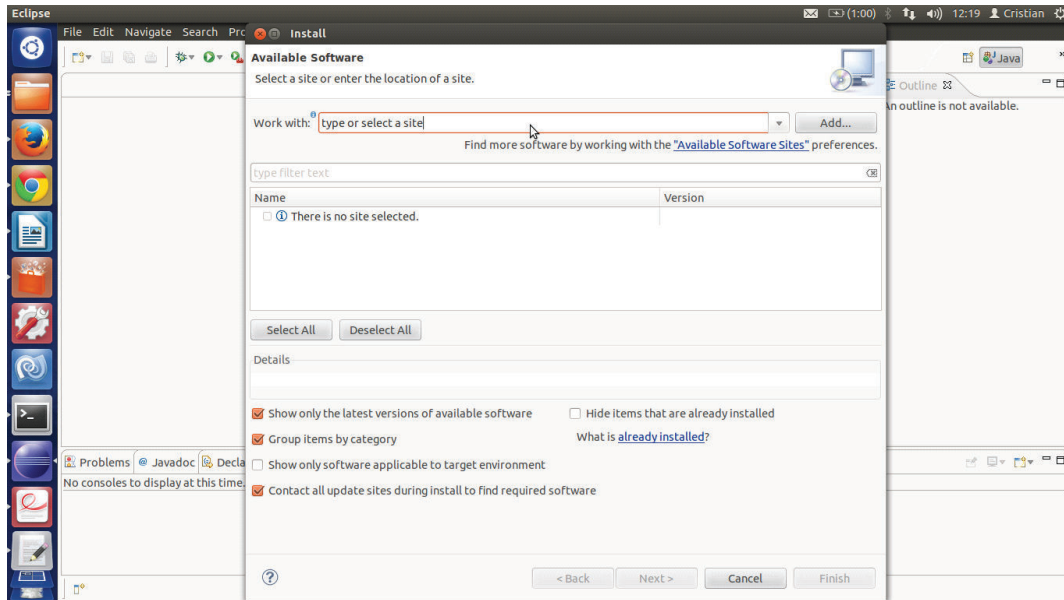


Figura A-44 Instalar plugin-Eclipse

- Se pulsa en la pestaña *Add*.

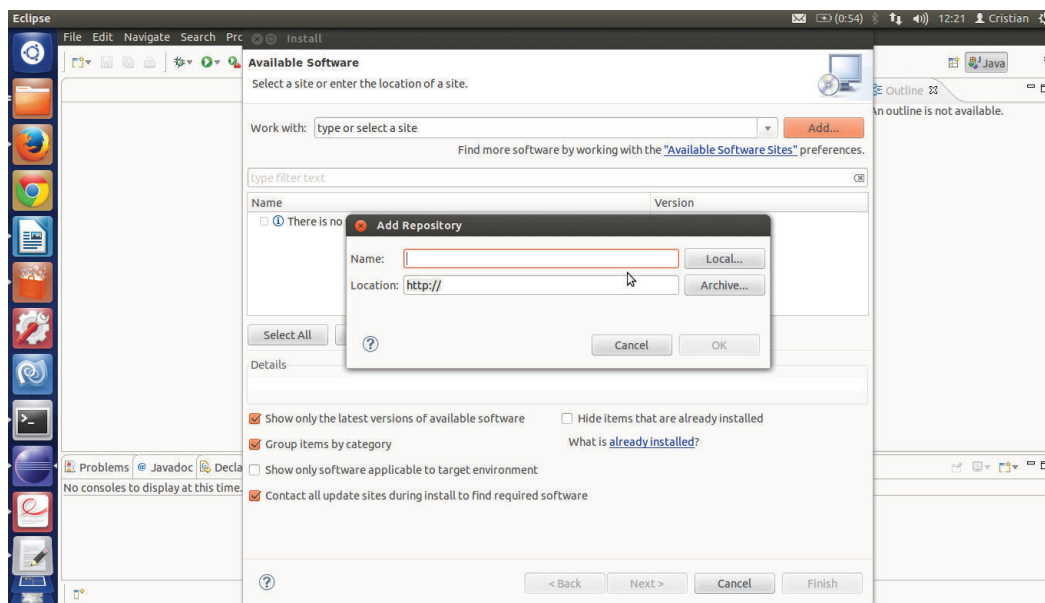


Figura A-45 Añadiendo nuevo software

Se presiona el botón *Archivo* y se busca el Plug-in en la dirección `"/moterunner/IDE/mrdevel-feature.zip"`.

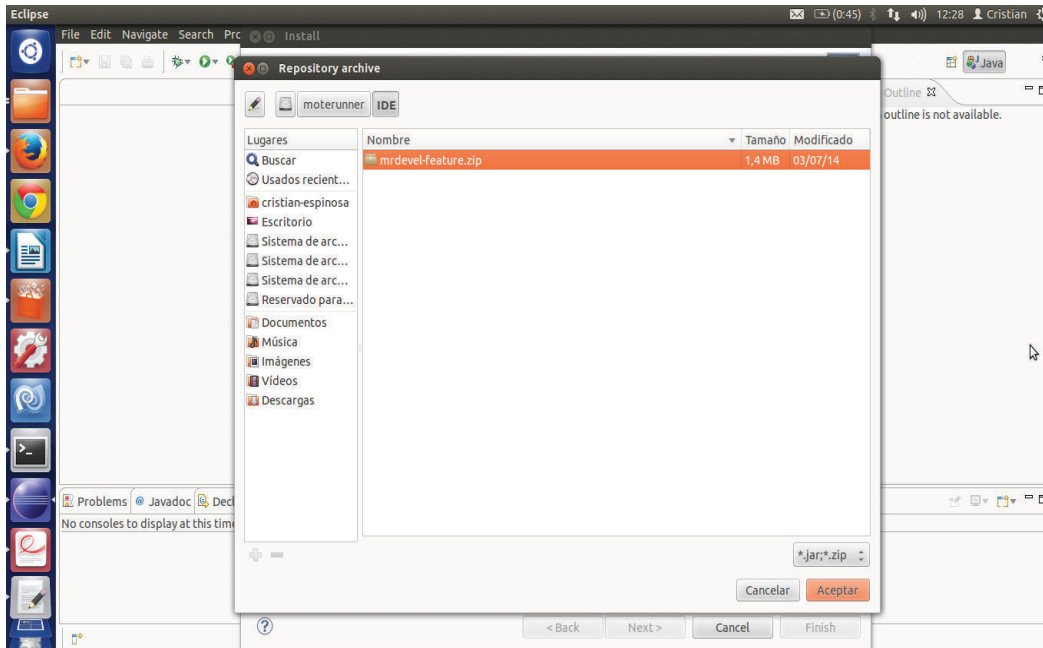


Figura A-46 Añadiendo el plugin

Ahora se pulsa en el botón *Aceptar*, a continuación se selecciona *Mote Runner Development*.

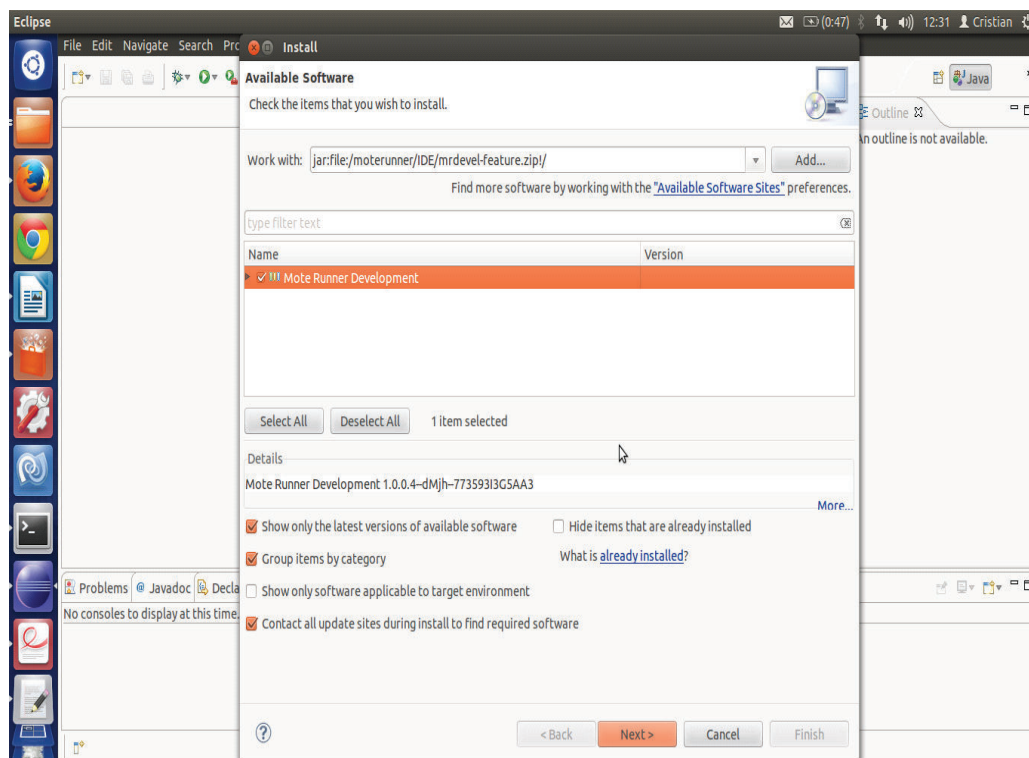


Figura A-47 Eclipse Plugin

Se pulsa el botón *Next*.



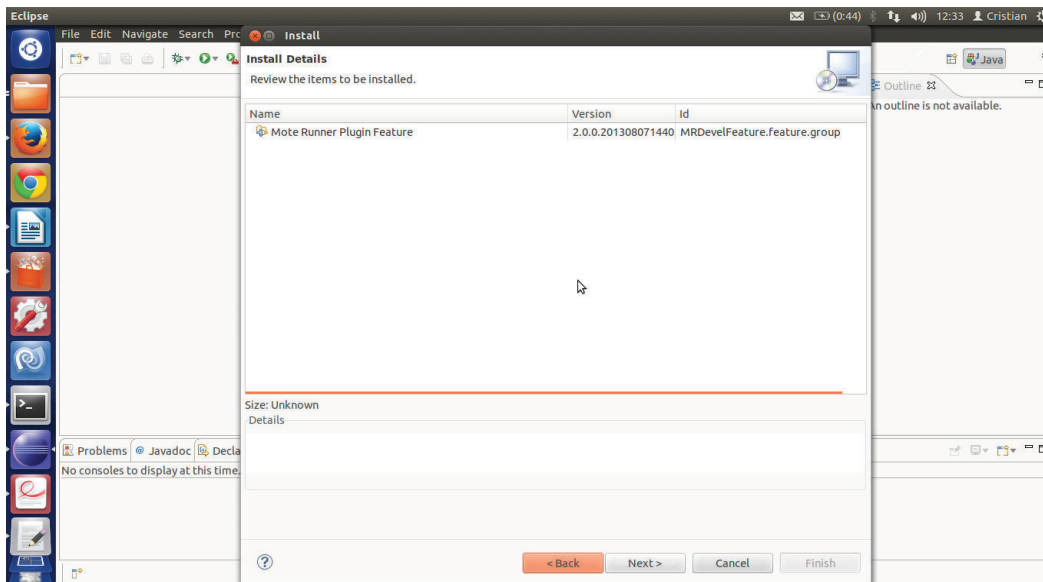


Figura A-48 Instalación Plugin Eclipse

Se pulsa *Next*, a continuación se acepta los términos de la licencia y finalmente se pulsa el botón *Finish*.

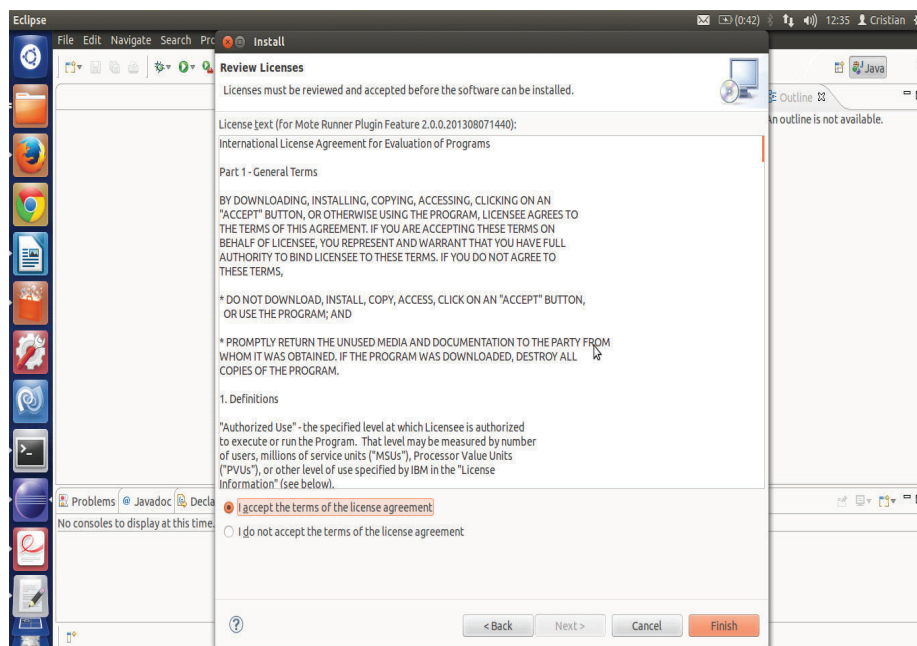


Figura A-49 Licencia de acuerdo del plugin

Aparecerá un mensaje en la pantalla con la indicación de que se va a instalar un software de contenido desconocido; sin embargo este mensaje es ignorado y se pulsa el botón *Ok*.



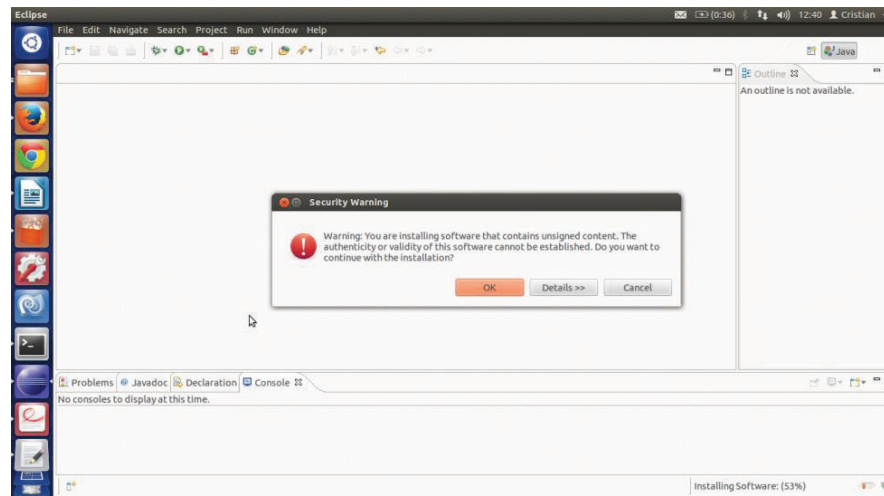


Figura A-50 Finalización de la instalación del plugin

Se finaliza la instalación y la misma solicitará que se debe realizar un reinicio del software Elipse.

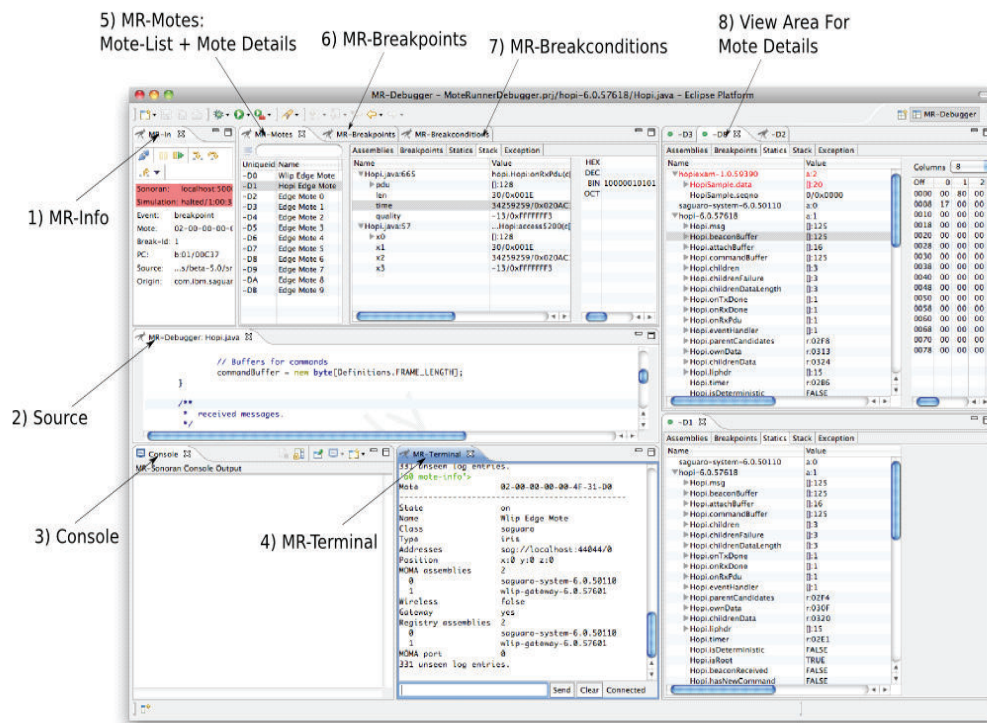


Figura A-51 Explicación del plugin para depuración en Eclipse

## PASOS PARA SIMULAR NODOS INALÁMBRICOS

- Eliminar todas las motas creadas.

- Eliminar el proceso saguaro que se estaba ejecutando.
- Iniciar el proceso saguaro con los siguientes comandos.

```
1 >saguaro-start
2 >saguaro-connect
```

Código A-27 Inicio del proceso Saguaro

- Crear una mota.

```
1 >mote-create -d waspmote
```

Código A-28 Creación del mote

- Con el comando “*cd*” se desplaza hacia el directorio donde están los archivos para cargar al nodo.

## DESARROLLO DE UNA RED 6LOWPAN UTILIZANDO NODOS MOTERUNNER

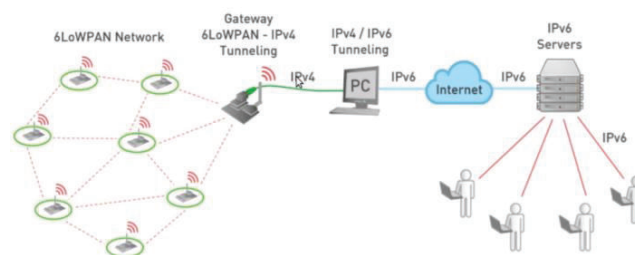


Figura A-52 Red que se pretende implementar

Como se puede apreciar en la imagen, se va a desarrollar una red de sensores inalámbricos los cuales trabajan con 6LowPAN, con direccionamiento IPv6, todos los nodos van a enviar información al Gateway. La conexión desde la PC y el Nodo Gateway se realizarán físicamente y en esta se va a estar ejecutado un túnel, el cual va a encapsular paquetes IPv6 dentro de IPv4. Al extremo derecho solo se trabajará con direccionamiento en IPv6.

En este ejemplo el cliente enviará una petición mediante NetCat versión 6 hacia los nodos los cuales devolverán el valor de temperatura que se podrá visualizar en la consola.

### NODOS FINALES

En los nodos finales se cargarán los siguientes programas:

- El programa o mejor dicho librería “*mrV-6-lib-1.0*” está ubicado en el siguiente directorio “*/moterunner/examples/mrv6/src/tmp*”
- La librería *wasp-common-1.0* ubicada en la siguiente dirección (dirección de descarga librerías Libelium moterunner-1,0 */src/com/libelium/common*)
- La librería *wasp-rtc-1.0* ubicada en la siguiente dirección (dirección de descarga librerías libelium moterunner-1.0 */src/com/libelium/rtc*)
- El programa que se ha desarrollado para medir la temperatura, que en este caso tiene por nombre *SenseTemp-1.0*, el código se muestra a continuación:

```

1 namespace com.ibm.saguaro.gateway.generic {
  using com.ibm.saguaro.system;
  using com.ibm.saguaro.mrv6;
  using com.ibm.saguaro.util;
  using com.libelium.rtc;
  using com.libelium.common;
  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
    uint temp = (uint)RTC.getInstance().getTemperature();
    byte aux = (byte)temp;
    buf[0] = aux;
    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;
  }

  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;
}
}
}

```

Código A-29 Programa para devolver la temperatura

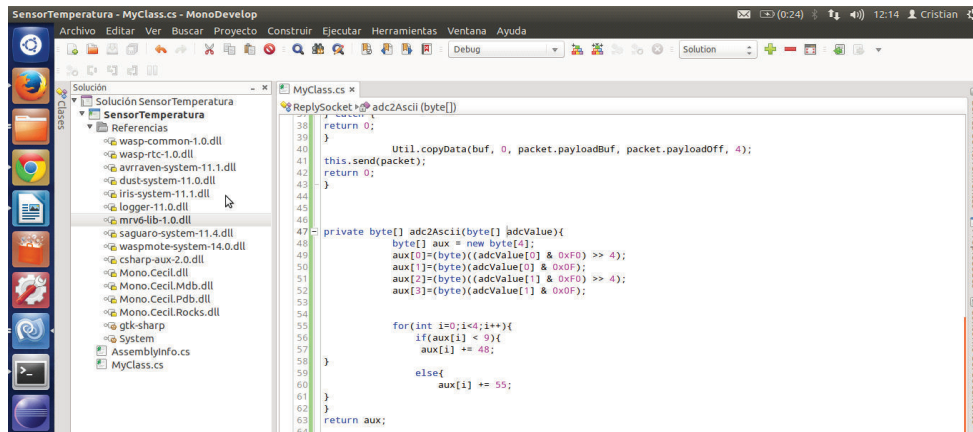


Figura A-53 Programa Sense

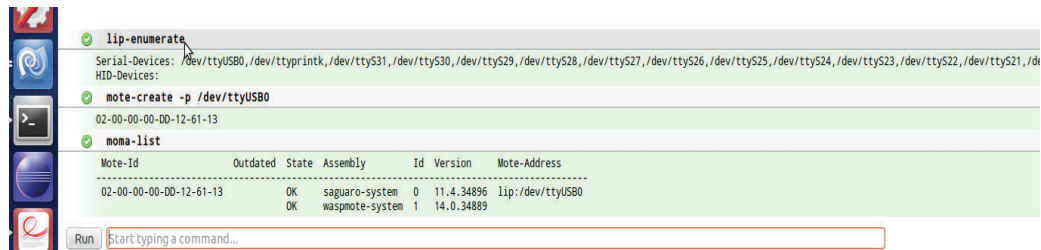


Figura A-54 Conexión con el nodo



Figura A-55 Carga de las librerías mrv6-lib y wasp-common

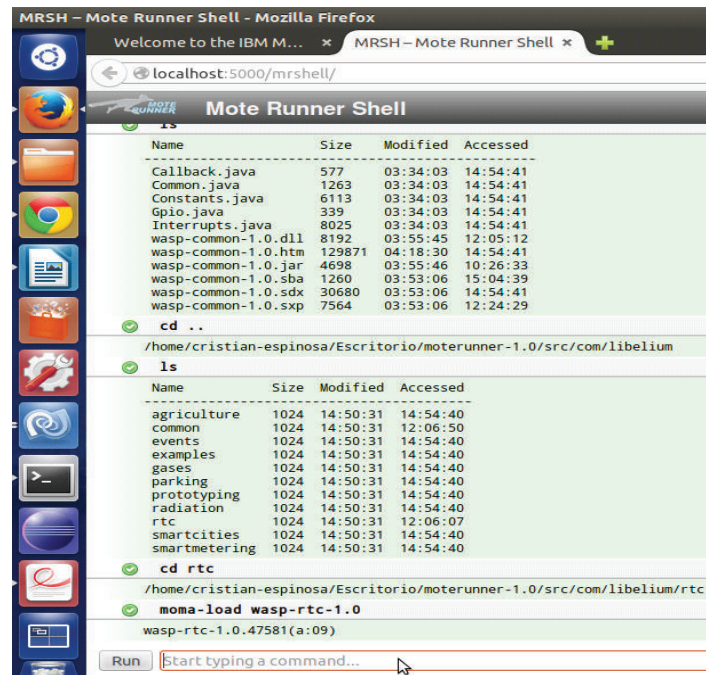


Figura A-56 Carga de la librería wasp-rtc

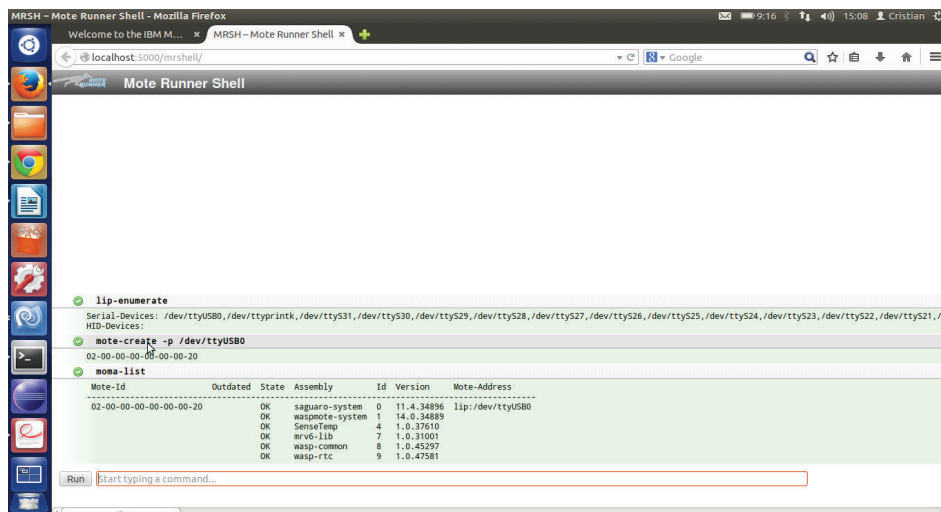


Figura A-57 Carga del programa SenseTemp

## NODO DE BORDE GATEWAY

En el nodo Gateway o de borde se instalará el siguientes programa: “*mrv6-edge-1.0*” que se encuentra en la dirección (*/moterunner/examples/mrv6/src/tmp*).

Después de cargar esta librería, para poder acceder al nodo puesto el módulo LAN, se debe asignar una dirección IPv4 al nodo, en vista de que ahora se va a

realizar la conexión a la PC a través del cable UTP con terminal RJ-45 y por lo tanto, no se utiliza el cable USB a la PC. Este proceso se realiza mediante el comando que se ejecuta en el servidor MRSB.

```
1 moma-ipv4 --ip 192.168.1.223 --subnwMask 255.255.255.0 --gateway 192.168.1.1 --udp 9999
```

Código A-30 Configuración de una dirección IPv4 al nodo Gateway

Como se puede ver en la Figura A-59, se indica que los cambios realizados serán efectuados al momento de reiniciar el nodo. Ahora bien, ya se puede desconectar el cable USB y colocar los módulos de radio frecuencia y de LAN para Gateway.

En este instante se conecta el nodo vía cable UTP a la computadora, para lo cual a nuestra PC, se tiene que asignarle la dirección IPv4 que se utiliza como dirección *Gateway* del nodo. En nuestro caso sería:

- Dirección IP: 192.168.1.1
- Máscara: 255.255.255.0

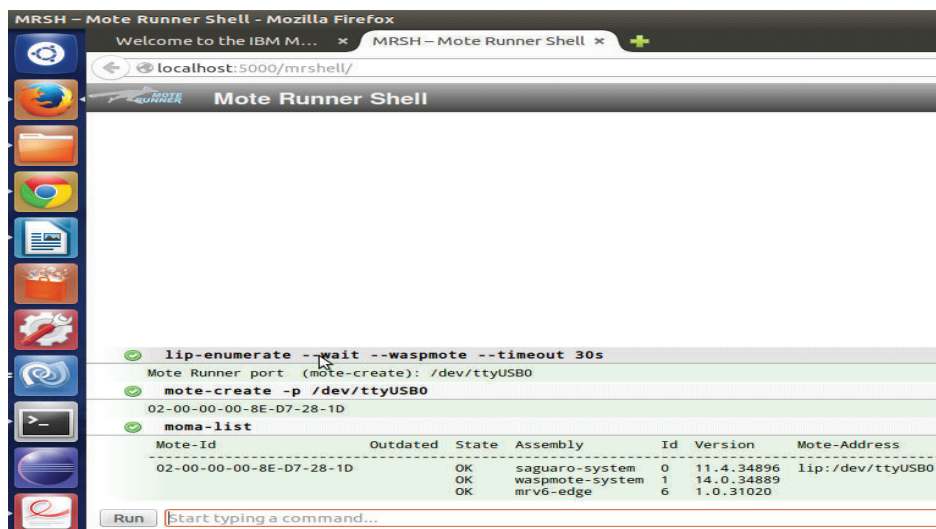


Figura A-58 Cargando la librería mrV6-edge en el nodo Gateway

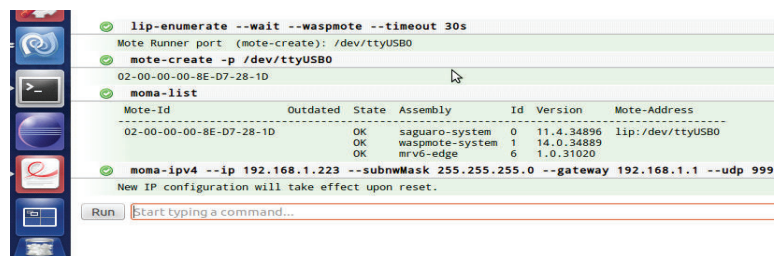


Figura A-59 Configuración de la dirección IPv4 al nodo Gateway



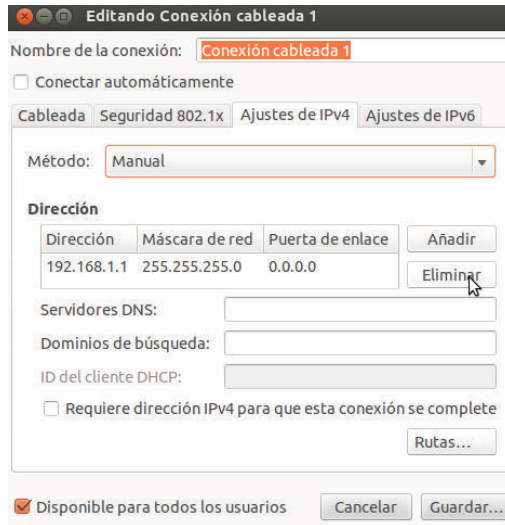


Figura A-60 Configuración de una dirección IPv4 en la PC

Una vez hecho esto, para que se efectúen los cambios realizados se ejecuta en una terminal el siguiente comando:

```
1 root@ubuntu:/home/cristian-espinoza/Escritorio# ifconfig eth0 down
```

Código A-31 Comando para poner shut en la interfaz eth0

Ahora se debe esperar un tiempo hasta visualizar que la interfaz cableada se haya apagado).

```
1 root@ubuntu:/home/cristian-espinoza/Escritorio# ifconfig eth0 up
```

Código A-32 Comando para levantar la interfaz eth0

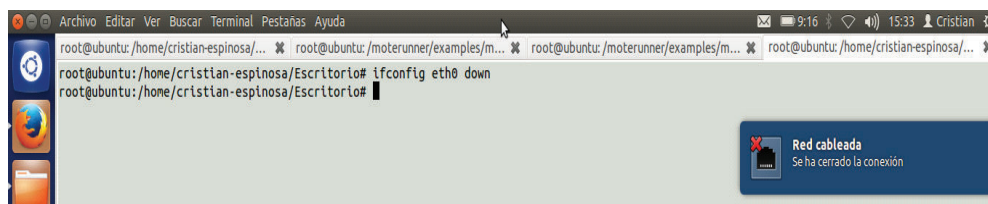


Figura A-61 Red cableada desconectada

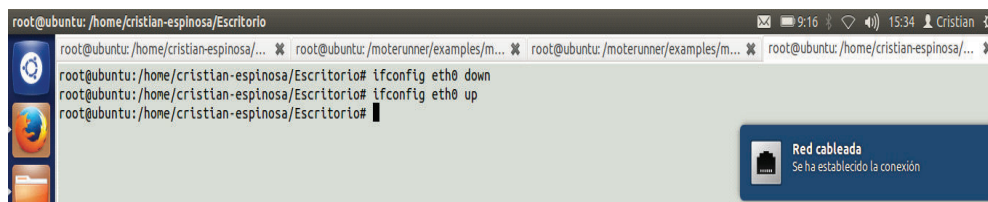


Figura A-62 Red cableada conectada

Ahora para poder visualizar que los cambios se llevaron a cabo, se efectúa en una terminal el siguiente comando:

```
1 root@ubuntu:/home/cristian-espinoza/Escritorio# ifconfig
```

Código A-33 Comando para verificar la configuración de red en las interfaces de la PC

```

root@ubuntu: /home/cristian-espinoza/Escritorio
root@ubuntu: /home/cristian-espinoza/Escritorio# ifconfig eth0 down
root@ubuntu: /home/cristian-espinoza/Escritorio# ifconfig eth0 up
root@ubuntu: /home/cristian-espinoza/Escritorio# ifconfig
eth0      Link encap:Ethernet  direcciónHW 00:8c:fa:3b:99:98
          Dirección inet:192.168.1.1  Difus.:192.168.1.255  Másc:255.255.255.0
          Dirección inet6: fd00::1/8  Alcance:Global
          Dirección inet6: fe80::28c:faff:fe3b:9998/64  Alcance:Enlace
          ACTIVO DIFUSIÓN FUNCIONANDO MULTICAST  MTU:1500  Métrica:1
          Paquetes RX:0 errores:0 perdidos:0 overruns:0 frame:0
          Paquetes TX:0 errores:0 perdidos:0 overruns:0 carrier:0
          colisiones:0 long.colaTX:1000
          Bytes RX:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupción:16

```

Figura A-63 Verificación de la configuración IPv4 en la red cableada

Ahora se verificará que existe conectividad con el nodo Gateway usando el comando ping en una terminal a la dirección IPV4, la cual fue asignada nodo Gateway:

```

1 root@ubuntu: /home/cristian-espinoza/Escritorio# ping
  192.168.1.223

```

Código A-34 Verificación de conectividad con el Nodo Gateway mediante IPv4

Si el ping es exitoso se verá parpadear el led naranja del módulo LAN del nodo Gateway.

```

root@ubuntu: /home/cristian-espinoza/Escritorio# ping 192.168.1.223
PING 192.168.1.223 (192.168.1.223) 56(84) bytes of data.
64 bytes from 192.168.1.223: icmp_req=1 ttl=128 time=0.489 ms
64 bytes from 192.168.1.223: icmp_req=2 ttl=128 time=0.201 ms
64 bytes from 192.168.1.223: icmp_req=3 ttl=128 time=0.206 ms
64 bytes from 192.168.1.223: icmp_req=4 ttl=128 time=0.203 ms

```

Figura A-64 Verificación de conectividad IPv4 con el nodo Gateway

Ahora se procede a conectar el nodo por medio del *Moterunner* para esto se debe tener ejecutando el servidor web y abrir un Shell:

```

1 mote-create -i 192.168.1.223

```

Código A-35 Conexión con el nodo Gateway mediante su dirección IPv4

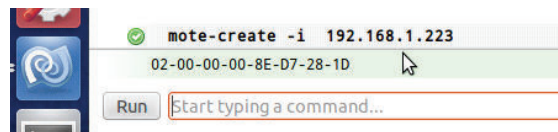


Figura A-65 Conexión al nodo Gateway mediante su dirección IPv4

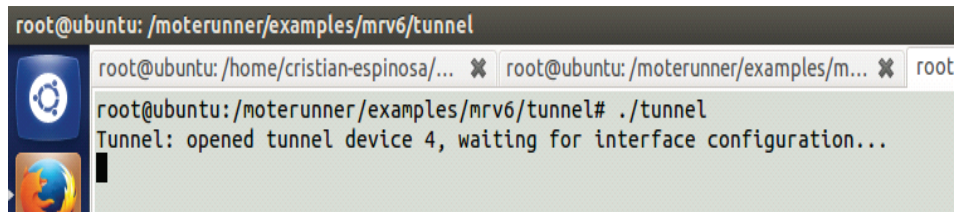
Ahora se configura el túnel para esto se deberán tener abiertas tres pestañas de una terminal, en la primera pestaña debe estar ejecutándose servidor *MRS*H, en la siguiente pestaña se ejecutará la aplicación para la PC del túnel “que se encuentra en la dirección `/moterunner/examples/mrv6/tunnel`” y para poder



ejecutar dicho aplicativo se utiliza el comando que se muestra a continuación:

```
1 root@ubuntu:/moterunner/examples/mrv6/tunnel# ./tunnel
```

**Código A-36 Ejecución del túnel**



**Figura A-66 Inicio del túnel**

Ahora en el shell del moterunner, se debe dirigir a la siguiente dirección:

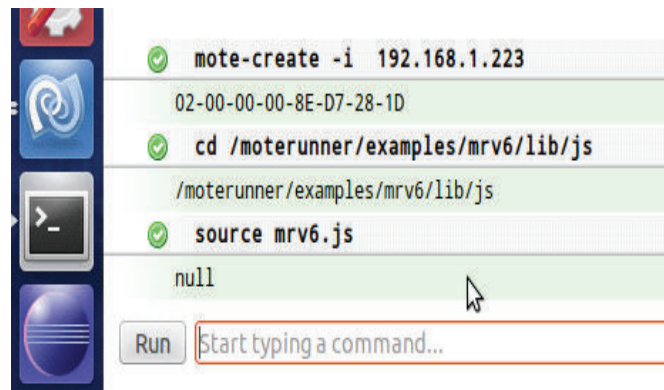
```
1 cd /moterunner/examples/mrv6/lib/js
```

**Código A-37 Directorio donde se encuentra el JavaScript**

Se ejecuta el siguiente comando del Código A-38:

```
1 source mrv6.js
```

**Código A-38 Carga del JavaScript**



**Figura A-67 Inicio del JavaScript para la configuración de la red 6LowPAN**

Ahora se debe dirigir a la siguiente dirección:

*“cd /moterunner/examples/mrv6/src/tmp”*

Se ejecuta el siguiente comando para configurar el túnel y las direcciones de todos los nodos que conforman la red 6LowPAN:

```
1 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=20 -INFO_INTERVAL_CNT=0
```

**Código A-39 Comando que permitirá configurar la red 6LowPAN**

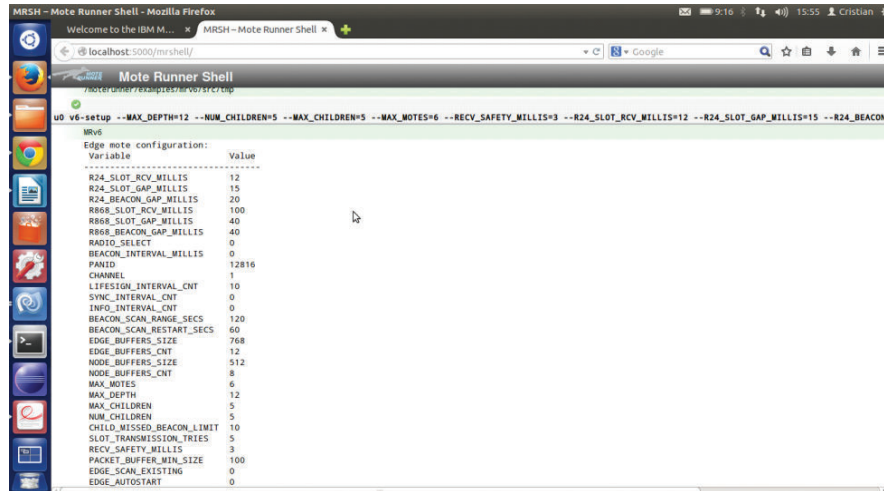


Figura A-68 Red 6LowPAN configurada con algunos parámetros

En la terminal donde está corriendo el tunnel se puede apreciar que todos los nodos han sido añadidos incluso hasta el nodo Gateway.

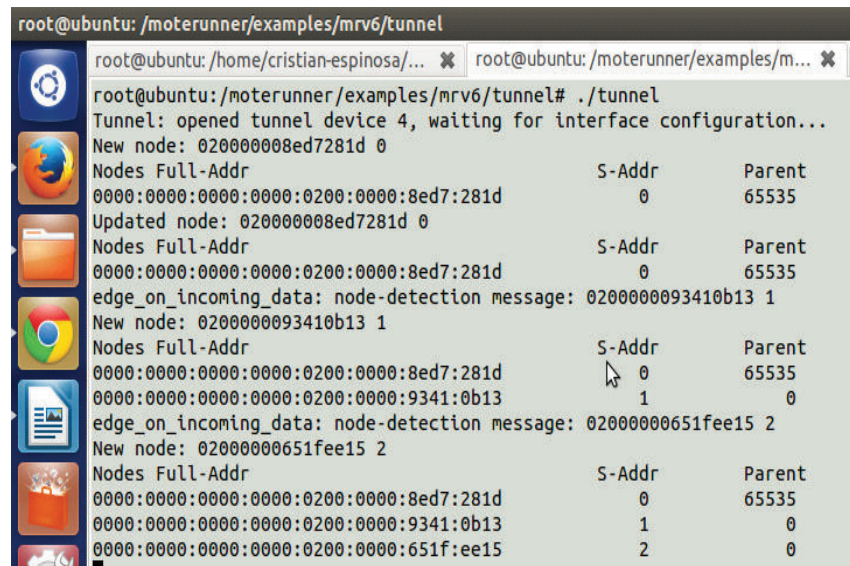


Figura A-69 Nodos añadiéndose a la red 6LowPAN

Para el establecimiento de las rutas y la configuración del tunnel es importante dirigirse a la siguiente dirección:

```
1 root@ubuntu:~# cd
  /moterunner/examples/mrv6/tunnel/
```

Código A-40 Comando para cambiarme de dirección donde se encuentra el túnel

Se ejecuta el siguiente archivo de configuración:

```
1 root@ubuntu: /moterunner/examples/mrv6/tunnel# ./route_setup_linux_ip6.sh
```

```

root@ubuntu: /moterunner/examples/mrv6/tunnel
root@ubuntu: /home/cristian-espinoza/... ✖ root@ubuntu: /moterunner/examples/m... ✖ root@ubuntu: /
root@ubuntu: /moterunner/examples/mrv6/tunnel# cd ~
root@ubuntu: ~# cd /moterunner/examples/mrv6/tunnel/
root@ubuntu: /moterunner/examples/mrv6/tunnel# ./route_setup_linux_ip6.sh
root@ubuntu: /moterunner/examples/mrv6/tunnel#

```

Figura A-70 Configuración del túnel el cuál va a dar el direccionamiento IPv6 a la red 6LowPAN

Con esto se puede observar en la pestaña donde se ejecuta la aplicación tunnel que las direcciones Ipv6 fueron asignadas, tanto al nodo Gateway como a los nodos inalámbricos.

Para poder visualizar en el *Moterunner Shell* las direcciones asignadas se utiliza los comandos:

- network-list
- v6-connect

```

Nodes Full-Addr          S-Addr      Parent
0000:0000:0000:0000:0200:0000:8ed7:281d      0          65535
Updated node: 020000008ed7281d 0
Nodes Full-Addr          S-Addr      Parent
0000:0000:0000:0000:0200:0000:8ed7:281d      0          65535
edge_on_incoming_data: node-detection message: 0200000093410b13 1
New node: 0200000093410b13 1
Nodes Full-Addr          S-Addr      Parent
0000:0000:0000:0000:0200:0000:8ed7:281d      0          65535
0000:0000:0000:0000:0200:0000:9341:0b13      1           0
edge_on_incoming_data: node-detection message: 02000000651fee15 2
New node: 02000000651fee15 2
Nodes Full-Addr          S-Addr      Parent
0000:0000:0000:0000:0200:0000:8ed7:281d      0          65535
0000:0000:0000:0000:0200:0000:9341:0b13      1           0
0000:0000:0000:0000:0200:0000:651f:ee15      2           0
Tunnel: interface ipaddr: 20000000000000000000000000000000ff, xaddr: 00000000000000ff
Tunnel: network prefix: 2000:0000:0000:0000
Tunnel IP interface configured.

Nodes Full-Addr          S-Addr      Parent
2000:0000:0000:0000:0200:0000:8ed7:281d      0          65535
2000:0000:0000:0000:0200:0000:9341:0b13      1           0
2000:0000:0000:0000:0200:0000:651f:ee15      2           0
icmp6: unhandled ICMPv6 type: 143
icmp6: unhandled ICMPv6 type: 143

```

Figura A-71 Direccionamiento IPv6 a la red 6LowPAN

```

network-list
No simulations started.

Mote-Abbrev  Name      State  Connection      Uniqueid      Addresses
-----
u0,w0       192.168.1.22  on     UDP:192.168.1.223:9999  02-00-00-00-8E-D7-28-1D  udp://192.168.1.223:9999
w1          Hw-Mote 0      on     MRv6             02-00-00-00-93-41-0B-13  rf://02-00-00-00-93-41-0B-13  v6://02-00-00-00-93-41-0B-13
w2          Hw-Mote 1      on     MRv6             02-00-00-00-65-1F-EE-15  rf://02-00-00-00-65-1F-EE-15  v6://02-00-00-00-65-1F-EE-15

v6-connect

Mote          Parent      Address  Hops
-----
02-00-00-00-8E-D7-28-1D  Gateway    0        0
02-00-00-00-93-41-0B-13  02-00-00-00-8E-D7-28-1D  1        1
02-00-00-00-65-1F-EE-15  02-00-00-00-8E-D7-28-1D  2        1

```

Figura A-72 Comandos "network-list" y "v6-connect"

Ahora desde la PC se puede hacer ping tanto al nodo Gateway como a los nodos inalámbricos que conforman la red 6LowPAN:

```
1 root@ubuntu:/moterunner/examples/mrv6/tunnel#ping6 2000:0000:0000:0000:0200:0000:8ed7:281d
```

**Código A-41 probando conectividad IPv6 con la red 6LowPAN**

```
1 root@ubuntu:/moterunner/examples/mrv6/tunnel# ping6
2000:0000:0000:0000:0200:0000:651f:ee15
```

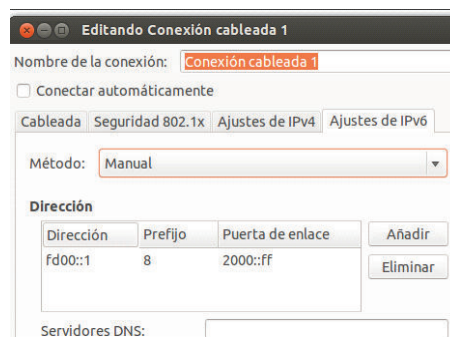
**Código A-42 Comprobando conectividad IPv6 con la red 6LowPAN**

```
root@ubuntu:~# cd /moterunner/examples/mrv6/tunnel/
root@ubuntu:/moterunner/examples/mrv6/tunnel# ./route_setup_linux_ip6.sh
root@ubuntu:/moterunner/examples/mrv6/tunnel# ping6 2000:0000:0000:0000:0200:0000:8ed7:281d
PING 2000:0000:0000:0000:0200:0000:8ed7:281d(2000::200:0:8ed7:281d) 56 data bytes
16 bytes from 2000::200:0:8ed7:281d: icmp_seq=1 ttl=64 (truncated)
16 bytes from 2000::200:0:8ed7:281d: icmp_seq=2 ttl=64 (truncated)
16 bytes from 2000::200:0:8ed7:281d: icmp_seq=3 ttl=64 (truncated)
^C
--- 2000:0000:0000:0000:0200:0000:8ed7:281d ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 9223372036854775.807/0.000/0.000/0.000 ms
root@ubuntu:/moterunner/examples/mrv6/tunnel# ping6 2000:0000:0000:0000:0200:0000:651f:ee15
PING 2000:0000:0000:0000:0200:0000:651f:ee15(2000::200:0:651f:ee15) 56 data bytes
16 bytes from 2000::200:0:651f:ee15: icmp_seq=1 ttl=64 (truncated)
16 bytes from 2000::200:0:651f:ee15: icmp_seq=2 ttl=64 (truncated)
16 bytes from 2000::200:0:651f:ee15: icmp_seq=3 ttl=64 (truncated)
16 bytes from 2000::200:0:651f:ee15: icmp_seq=4 ttl=64 (truncated)
16 bytes from 2000::200:0:651f:ee15: icmp_seq=5 ttl=64 (truncated)
^C
--- 2000:0000:0000:0000:0200:0000:651f:ee15 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3997ms
rtt min/avg/max/mdev = 9223372036854775.807/0.000/0.000/0.000 ms
root@ubuntu:/moterunner/examples/mrv6/tunnel#
```

**Figura A-73 Verificación de conectividad IPv6 con toda la red 6LowPAN**

Cabe recalcar que el archivo `route_setup_linux_ip6.sh` fue modificado para obtener direcciones que puedan ser publicadas en el internet. Por defecto viene la dirección `fc00:db8:5::ff/8`, por lo cual es recomendable modificar esta dirección IPv6, específicamente en este caso se ha puesto con la dirección `2000::ff/8`, esta dirección no puede ser mayor a `/48`.

Posteriormente falta configurar una dirección IPv6 a la máquina donde correrá la aplicación del túnel, así como también en el lado de los computadores donde corre la aplicación diseñada.



**Figura A-74 Configuración de una dirección IPv6 a la PC externa**





Figura A-75 Configuración de una dirección IPv6 a la PC que ejecuta el túnel

En este caso se le asigna la dirección IPv6 *fd00:1/8* a la computadora donde corre el programa del túnel, así como la dirección IPv6 *fd00:2/8* al otro computador final. Ya establecidas las direcciones IPv6 es posible hacer una comprobación de conectividad por medio del comando “ping6”.

Ping desde aplicación hasta el Gateway:

```

root@carlos-desktop: /home/carlos
--- 2000::200:0:8ed7:281d ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4004ms
rtt min/avg/max/mdev = 9223372036854775.807/0.000/0.000/0.000 ms
root@carlos-desktop: /home/carlos# ping6 2000::200:0:8ed7:281d
PING 2000::200:0:8ed7:281d(2000::200:0:8ed7:281d) 56 data bytes
16 bytes from 2000::200:0:8ed7:281d: icmp_seq=1 ttl=63 (truncated)
16 bytes from 2000::200:0:8ed7:281d: icmp_seq=2 ttl=63 (truncated)
16 bytes from 2000::200:0:8ed7:281d: icmp_seq=3 ttl=63 (truncated)
16 bytes from 2000::200:0:8ed7:281d: icmp_seq=4 ttl=63 (truncated)
16 bytes from 2000::200:0:8ed7:281d: icmp_seq=5 ttl=63 (truncated)
^C
--- 2000::200:0:8ed7:281d ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4001ms
rtt min/avg/max/mdev = 9223372036854775.807/0.000/0.000/0.000 ms
root@carlos-desktop: /home/carlos#
    
```

Figura A-76 Conectividad extremo a extremo

Ping desde aplicación hacia los nodos:

```

root@carlos-desktop: /home/carlos
--- 2000::200:0:8ed7:281d ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4001ms
rtt min/avg/max/mdev = 9223372036854775.807/0.000/0.000/0.000 ms
root@carlos-desktop: /home/carlos# ping6 2000::200:0:9341:0b13
PING 2000::200:0:9341:0b13(2000::200:0:9341:b13) 56 data bytes
16 bytes from 2000::200:0:9341:b13: icmp_seq=1 ttl=63 (truncated)
16 bytes from 2000::200:0:9341:b13: icmp_seq=2 ttl=63 (truncated)
16 bytes from 2000::200:0:9341:b13: icmp_seq=3 ttl=63 (truncated)
16 bytes from 2000::200:0:9341:b13: icmp_seq=4 ttl=63 (truncated)
16 bytes from 2000::200:0:9341:b13: icmp_seq=5 ttl=63 (truncated)
^C
--- 2000::200:0:9341:0b13 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4003ms
rtt min/avg/max/mdev = 9223372036854775.807/0.000/0.000/0.000 ms
root@carlos-desktop: /home/carlos#
    
```

Figura A-77 Verificación de conectividad extremo a extremo

```

root@carlos-desktop: /home/carlos
5 packets transmitted, 5 received, 0% packet loss, time 4003ms
rtt min/avg/max/mdev = 9223372036854775.807/0.000/0.000/0.000 ms
root@carlos-desktop: /home/carlos# ping6 2000::200:0:651f:ee15
PING 2000::200:0:651f:ee15(2000::200:0:651f:ee15) 56 data bytes
16 bytes from 2000::200:0:651f:ee15: icmp_seq=1 ttl=63 (truncated)
16 bytes from 2000::200:0:651f:ee15: icmp_seq=2 ttl=63 (truncated)
16 bytes from 2000::200:0:651f:ee15: icmp_seq=3 ttl=63 (truncated)
16 bytes from 2000::200:0:651f:ee15: icmp_seq=4 ttl=63 (truncated)
16 bytes from 2000::200:0:651f:ee15: icmp_seq=5 ttl=63 (truncated)
16 bytes from 2000::200:0:651f:ee15: icmp_seq=6 ttl=63 (truncated)
^C
--- 2000::200:0:651f:ee15 ping statistics ---
5 packets transmitted, 6 received, 0% packet loss, time 5000ms
rtt min/avg/max/mdev = 9223372036854775.807/0.000/0.000/0.000 ms
root@carlos-desktop: /home/carlos#
    
```

Figura A-78 Verificación de conectividad extremo a extremo

En este punto se tiene ya la conectividad tanto del lado de los nodos que conforman la red 6LowPAN hasta la PC donde se ejecuta el túnel, como de esta misma PC hasta a la otra.

Sin embargo, todavía no se tiene una total comunicación total de extremo a extremo, para lo cual es necesario establecer en la misma instancia donde corre el túnel, una función de enrutamiento para poder especificar el túnel como interfaz de entrada/salida. Para ello se debe instalar la aplicación *radvd*.

```
1 root@ubuntu:/moterunner/examples/mrv6/tunnel# sudo apt-get install radvd
```

**Código A-43 Aplicación que permitirá enrutar los datos de forma correcta**

El archivo de configuración deberá ser creado con el comando *gedit /etc/radvd.conf* y deberá tener el siguiente contenido o parecido:

```
1 interface ath0
2
3 {
4
5     AdvSendAdvert on;
6
7     prefix fd00::/8
8
9     {
10
11         AdvOnLink on;
12
13     };
14 };
15
16 };
17
```

**Código A-44 Archivo de configuración radvd**

En este archivo se puede configurar la dirección de red correspondiente a la configurada entre estas dos máquinas.

Con la siguiente instrucción se puede iniciar el demonio que corresponde al *radvd*. Esto es muy importante ya que una vez hecho demonio ya se ejecutará por sí solo.

```
1 root@ubuntu:/moterunner/examples/mrv6/tunnel# sudo /etc/init.d/radvd start
```

**Código A-45 Comando para ejecutar como demonio la aplicación radvd**

## **ANEXO B**

### **CÓDIGO DE LA APLICACIÓN**

El código de la aplicación se encuentra en un CD adjunto.