ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

COMPARACIÓN DE LOS MÉTODOS MST Y B&B EN LA RESOLUCIÓN DEL TSP EN UNA WSN SIMULADA

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

JONATHAN EDUARDO TITO ONTANEDA
jonathan.e.tito.o@gmail.com

DIRECTOR: ING. MARCO ESTEBAN YACELGA PINTO, MSc.
marco.yacelgap@epn.edu.ec

CODIRECTOR: ING. CARLOS ROBERTO EGAS ACOSTA, MSc.
cegas@ieee.org

Quito, Diciembre 2016

DECLARACIÓN

Yo, Jonathan Eduardo Tito Ontaneda, 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.

Jonathan Eduardo Tito Ontaneda

CERTIFICACIÓN

Certificamos que el presente trabajo fue desarrollado por Jonathan Eduardo Tito Ontaneda, bajo nuestra supervisión.

Ing. Marco Yacelga, MSc.
DIRECTOR DEL PROYECTO

Ing. Carlos Egas, MSc.
CODIRECTOR DEL PROYECTO

AGRADECIMIENTO

A mis padres por su apoyo y sobre todo por el sacrificio para que a mis hermanos y a mí no nos falte nada, e inclusive podamos tener el privilegio de gozar de una educación que ni ellos, ni sus padres, ni los padres de sus padres pudieron disfrutar. Gracias por darme la oportunidad de estudiar en la mejor universidad del Ecuador y por tener la paciencia y el coraje para que la familia a la que pertenecemos madure con el tiempo y no se derrumbe después de tantas vicisitudes que hemos atravesado a lo largo de este camino que es la vida.

A mi hijo por su sonrisa, por el tiempo robado que he tenido que dedicar a la universidad antes que a la verdadera prioridad de mi vida, que es él. Espero que al leer esto muchos años después de mi partida recuerdes que tu viejo te ama, recuerda que posees la herencia más valiosa que me dejaron mis padres: la educación (no la desperdicies).

A mis compañeras y compañeros de aulas con quienes juntos recorrimos el largo y tortuoso, pero a la vez gratificante camino de la universidad hasta este punto, juntos, sin egoísmo, sin pasar por encima de nadie y sin perder la esencia de quienes somos. Gracias por su ayuda inclusive en el ámbito personal, siempre estuvieron allí y quiero que sepan que conmigo pueden contar incondicionalmente.

Para finalizar, agradezco a la Escuela Politécnica Nacional porque hasta cuando fallé aprendí, a mis profesores a lo largo de la vida universitaria, y de manera especial al Ing. Marco Yacelga MSc. y al Ing. Carlos Egas MSc. por su apoyo y guía a lo largo del presente trabajo.

Jonathan.

DEDICATORIA

A Saúl, mi hijo.

A Pablo Tito y Mariana Ontaneda, mis padres.

A mi sangre. De la que vengo, la que di y la que vendrá.

Jonathan.

CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ESPACIOS DE CÓDIGO

RESUMEN

La parte primordial de este trabajo es la resolución del problema del agente viajero (TSP — *Traveling Salesman Problem*), el cual consiste en que un agente viajero encuentre el camino que minimice lo más posible el costo del viaje (o la distancia) para visitar todas las ciudades de su periplo y retornar al punto de partida. En la simulación, el agente viajero sería un paquete a transmitirse desde un nodo fuente que pase por todos los nodos de la red inalámbrica de sensores (WSN — *Wireless Sensor Network*) para finalmente regresar al nodo inicial, utilizando para todo su recorrido el camino cuyo costo sea el mínimo posible. En el primer capítulo se realizará una breve introducción teórica en la que se analizarán la arquitectura, características y manejo de comunicaciones de las WSN. También se analizarán las definiciones fundamentales y los problemas principales de procesamiento de la teoría de grafos. Este último análisis servirá para explicar el TSP a detalle.

En el segundo capítulo se expondrá la teoría de cómo resolver el TSP en una WSN, usando los métodos del árbol de expansión mínima (MST — *Minimum Spanning Tree*) y ramificación y poda[1] (B&B — *Branch and Bound*). En la exposición del método por MST se compararán los siguientes algoritmos para la obtención del MST: el algoritmo de Prim, el algoritmo de Borůvka y el algoritmo de Kruskal con el fin de determinar el algoritmo que menos tiempo tarde en construir el MST en una WSN.

En el tercer capítulo, se definirán en Castalia los escenarios A y B, los modelos de nodos TelosB, Imote2 y Zolertia, además de los métodos de resolución del TSP por MST y B&B. Se implementará la simulación realizando diez repeticiones para cada combinación de escenario, modelo de nodo y método de resolución del TSP.

En el capítulo final, se establecerán las conclusiones y recomendaciones del presente proyecto, además de aplicaciones futuras.

---

[1]Se utiliza cotas para *podar* (eliminar) las ramas del árbol de soluciones que no encaminen la búsqueda realizada a la solución del problema.

PRESENTACIÓN

Las WSN se han vuelto ubicuas y son usadas en un amplio rango de dominios de aplicación [1]. La importancia de las WSN se ve reflejada en su gran número de aplicaciones dada la gran cantidad de parámetros medibles por los sensores, acorde a [2] se pueden mencionar los siguientes proyectos: Wisevine, Great Duck Island, Ecosense, Vital Sign. El TSP es quizás el problema de optimización combinatoria más estudiado a lo largo de la historia, dando lugar a multitud de variantes y extensiones con infinidad de aplicaciones prácticas en la vida real. En 1972 Richard M. Karp demuestra que el TSP es del tipo NP-duro[2] [3]. Uno de los problemas del milenio, por los que el Instituto Clay ofrece un millón de dólares por su resolución, se denomina P vs NP [4], siendo este antecedente un aliciente para el estudio del TSP.

Los parámetros medidos por los nodos de la WSN deben ser recolectados y transportados dependiendo de la aplicación en cuestión. Según [5] es posible reducir el consumo de potencia de los nodos sensores y, por lo tanto, extender el tiempo de vida de la WSN, reduciendo el número y tamaño de los datos comunicados. Establecer caminos de mínimo costo para el intercambio de datos en una WSN, reduce el número y tamaño de los datos comunicados. Uno de estos esquemas para encontrar un camino de mínimo costo se lo define con la resolución del TSP. La resolución del TSP podría aplicarse en la difusión eficiente de información en una red de datos. Su utilidad en las redes de datos se extiende a aplicaciones que hagan uso de la difusión, uno de los ejemplos mencionados en [6] es *Direct Diffusion*[3] (DD). En el presente trabajo se estudian dos métodos de resolución del TSP, simulándolos en una WSN, para determinar cuál método obtiene el mejor throughput con el menor consumo de energía.

---

[2] Tipo especial de problemas computacionales que no se pueden resolver en un tiempo razonable (polinomial).
[3] Protocolo de enrutamiento centrado en los datos (*data-centric*) en el cual se difunde mensajes considerados importantes para la WSN y solo los nodos interesados responden. Es así que se van creando caminos entre el nodo fuente y los nodos interesados en la información difundida [6].

1 CAPÍTULO 1: MARCO TEÓRICO

1.1 INTRODUCCIÓN A LAS REDES INALÁMBRICAS DE SENSORES

1.1.1 DEFINICIÓN DE RED INALÁMBRICA DE SENSORES

Una red inalámbrica de sensores (*Wireless Sensor Network* – WSN) según [7] es un sistema autónomo auto organizado de nodos sensores, distribuidos sobre una determinada área, para sensar datos de sus alrededores y agruparlos hacia un nodo fuente para su futuro procesamiento, tal como se observa en la Figura 1-1.

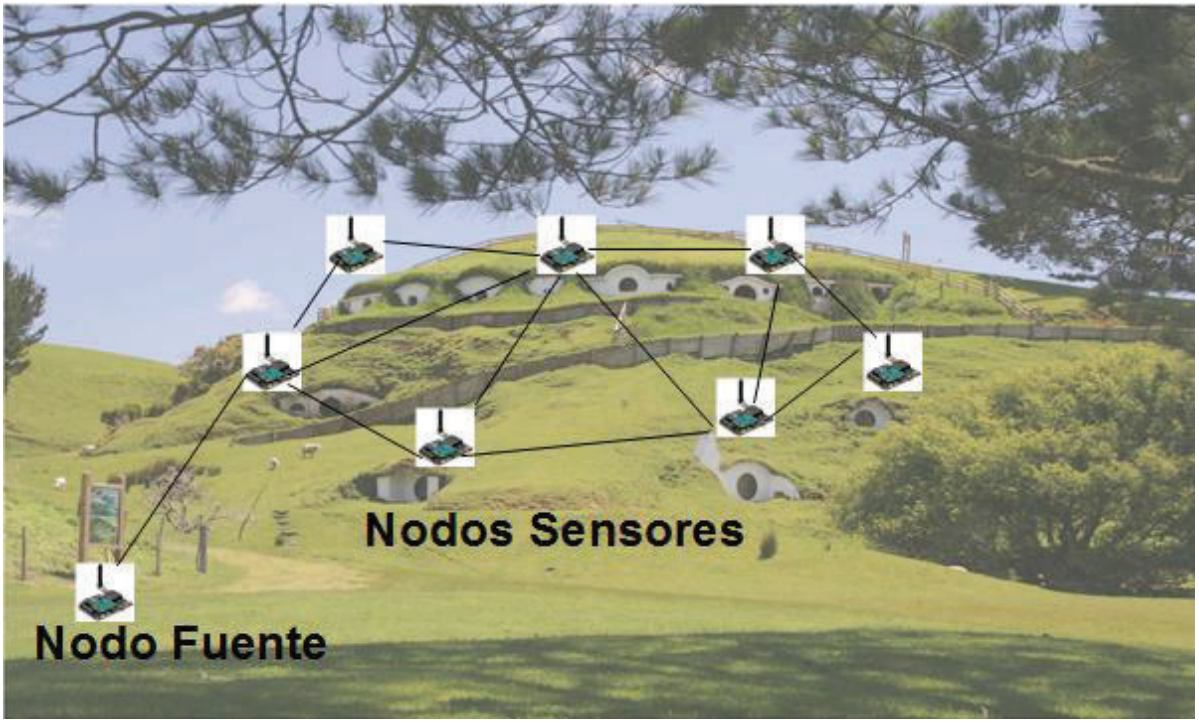


Figura 1-1 Esquema general de una WSN

Para ampliar la definición anterior, podemos citar a [8] que define una WSN como una colección de pequeños dispositivos de alimentación autónoma (comúnmente llamados nodos sensores), dotados con capacidades de sensado, comunicación y procesamiento.

En la Figura 1-1 se observa una WSN conformada por nodos sensores de la marca Libelium, para poder visualizar con mayor detalle la apariencia física de uno de estos nodos se añade la Figura 1-2.

Una vez desplegados los nodos sensores pueden capturar datos de alguna magnitud física tal como la temperatura, presión atmosférica o alguna concentración de contaminantes. Las lecturas de los sensores son usualmente reportadas a un servidor central, también llamado nodo fuente, donde estas lecturas serán posteriormente procesadas de acuerdo a los requerimientos de la aplicación.

Para reportar sus lecturas a uno o más colectores de datos, los nodos sensores se comunican a través de su radio transceptor y constituyen colaborativamente una red ad-hoc, posiblemente una red de retransmisión multisalto.

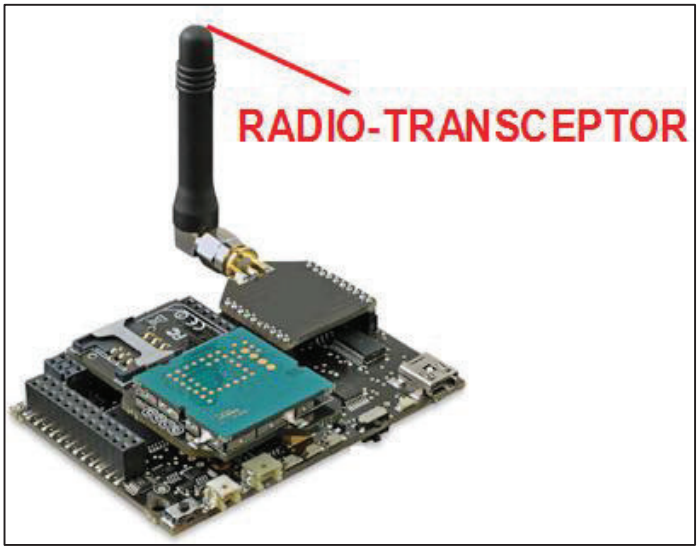


Figura 1-2 Ejemplo de nodo sensor marca Libelium, con su radio-transceptor [9]

Según [10] una WSN se define como una clase especial de las redes inalámbricas ad-hoc (redes independientes de infraestructura) que nos permiten estructurar, observar y responder a fenómenos en un ambiente natural, en nuestra infraestructura física y lógica. Esta última definición, es más clara si se realiza un estudio de la arquitectura de las WSN, como se presenta a continuación.

1.1.2 CARACTERÍSTICAS DE LAS REDES INALÁMBRICAS DE SENSORES

A continuación, se citan algunas características de las WSN tomadas de [11]:

Topología: La topología siempre es cambiante, debido a que los nodos sensores pueden dejar de funcionar en cualquier momento, y estos tienen que adaptarse además para poder comunicar nuevos datos adquiridos.

Ad-hoc: Es decir, una red inalámbrica de sensores no tiene necesidad alguna de infraestructura para poder operar.

Medio de Transmisión: En una red inalámbrica de sensores, los nodos están conectados inalámbricamente mediante radio, utilizando para ello transceptores como el CC2420 que opera en la banda de 2.4 GHz bajo el estándar IEEE 802.15.4.

Por ello, se debe tener en cuenta fenómenos propios del canal inalámbrico como la atenuación, desvanecimientos rápidos, desvanecimientos lentos e interferencias que pueden producir errores en la transmisión

Consumo energético: Los nodos sensores al tratarse de un dispositivo de escaso tamaño, poseen una fuente de energía bastante limitada, dependiendo de las baterías utilizadas.

Debido a esto el tiempo de vida de un nodo viene condicionado por la batería y el ciclo útil de trabajo de cada nodo, ya que en algunos casos la recarga de ellas depende de los módulos de alimentación adicionales, utilizando paneles solares para pilas recargables.

Tolerancia a errores: Una red inalámbrica de sensores tiene que ser capaz de seguir funcionando, en el caso de que un nodo sensor o varios no funcionen adecuadamente o dejen de funcionar.

Comunicaciones multisalto o *broadcast*: En las aplicaciones de las redes inalámbricas de sensores es característico el uso de algún protocolo que permita comunicaciones multisalto, aunque también es posible utilizar mensajería basada en difusión.

Limitaciones de Hardware: Para conseguir un consumo de energía razonable, se hace indispensable que el hardware sea lo más sencillo posible, así como su transceptor de radio.

Consecuentemente, existirá una capacidad de procesamiento limitada y una cantidad de memoria reducida.

Tamaño de mensajes reducidos: Usualmente los mensajes en estas redes son reducidos en comparación con las redes tradicionales.

Tráfico: En las redes inalámbricas de sensores, el tráfico principal generalmente fluye en sentido ascendente (*upstream*) desde los nodos sensores hasta el sumidero, aunque ocasionalmente el sumidero puede generar tráfico en sentido contrario (*downstream*) para realizar tareas de gestión y control. En el caso de tráfico ascendente, se realiza una comunicación de muchos sensores a uno. Además, dependiendo de la aplicación el tráfico puede ser entregado al sumidero de forma periódica, cada vez que ocurra un evento, de forma continua en tiempo real, o cualquier combinación de éstas.

Importes de producción: Para poder obtener datos con fiabilidad, una red de sensores inalámbrica tiene un número muy elevado de nodos sensores en su implementación práctica. La fabricación de nodos sensores es económica, si estos se producen en grandes cantidades. Existen kits de unos cuantos nodos sensores que son fabricados para fines experimentales, donde por el limitado número de nodos el precio de venta aumenta.

En [12] se realiza una comparación de las tecnologías inalámbricas tradicionales y se hace un resumen de las características de las WSN como se puede leer a continuación:

Las redes inalámbricas tradicionales, tales como redes móviles y ad-hoc (MANETs – *Mobile and Ad-hoc Networks*), redes celulares, y Bluetooth son redes inalámbricas de área local de corto alcance. Ellas trabajan principalmente a través de enrutamiento dinámico y tecnologías de gestión de movilidad para lograr su propósito de transmitir datos multimedia, por ejemplo, voz, imágenes y video.

El principal objetivo de las redes inalámbricas tradicionales es proveer alta calidad de servicio (QoS – *Quality of Service*) y de alta utilización del ancho de banda.

En una WSN se integran monitoreo, control y sistemas de comunicación inalámbrica. Comparadas con las redes inalámbricas tradicionales, las WSN tienen muchas

similitudes, así como también algunas diferencias significativas, tales como el gran número de nodos, limitados recursos en el hardware del nodo, limitaciones de la fuente de alimentación, topología dinámica y auto organización, etc.

El principal objetivo de las WSN, es utilizar eficientemente la energía, la cual es también la más importante diferencia entre las redes inalámbricas tradicionales y las WSN. Algunas de las principales características y restricciones de las WSN, presentadas de manera puntual y que se pueden encontrar de manera detallada, implícitos en los párrafos anteriores, se presentan en la Tabla 1-1:

Características de minimización generales	Características de Maximización generales	Características generales de red
Tamaño del nodo	Área de despliegue (depende de la aplicación)	Modelo de red auto organizado (red ad-hoc)
Costo de producción	Eficiencia de funcionamiento energético	Red centrada en los datos (la información que reporta el nodo es más importante que la *dirección* del nodo)
Limitados recursos del Hardware del nodo		Tolerancia a fallos (funciona en entornos hostiles)
Limitada capacidad de procesamiento		Movilidad (de los nodos sensores)
Limitada capacidad de la fuente de alimentación		Confiabilidad
Baja tasa de transmisión de datos		Flexibilidad
		Universalidad

Tabla 1-1 Resumen de las características de las WSN [12]

1.1.3 ARQUITECTURA DE LAS DE REDES INALÁMBRICAS DE SENSORES

1.1.3.1 Esquema de las redes inalámbricas de sensores

En el área de sensado (*sensor field*), los nodos sensores colectan y transmiten inalámbricamente los datos sensados hacia el nodo fuente (también se lo conoce como *Base Station, Gateway y Access Point*, además de nodo central o sumidero

como se menciona en [13]). Finalmente, el nodo fuente enviará los datos sensados a los usuarios mediante otro enlace de comunicaciones (por ejemplo, un enlace satelital, como se muestra en la Figura 1-3). Un nodo fuente puede ser cualquiera de una variedad de dispositivos, por ejemplo, una computadora portable, un PDA (*Personal Digital Assistant* – Ayudante Personal Digital), una estación terrena, etc.

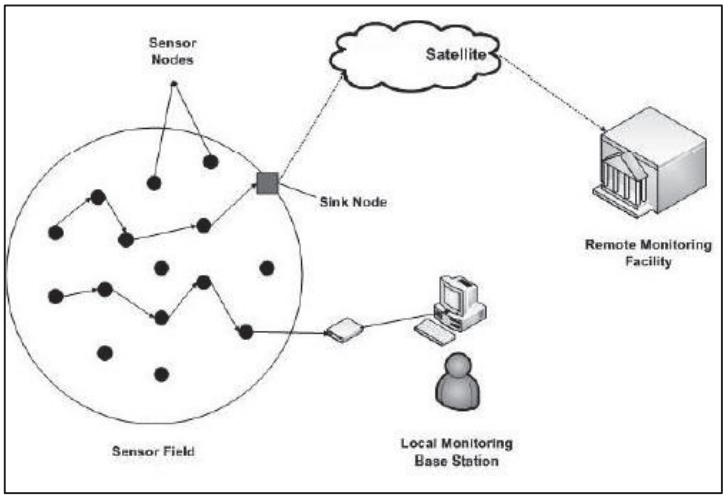


Figura 1-3 Estructura específica de una WSN [14]

Dependiendo de la aplicación de la WSN en cuestión, se puede tener más de un nodo fuente, en [15] existe un estudio sobre el tema de las WSN con múltiples nodos fuente.

Según [16], en las WSN los nodos sensores tienen la funcionalidad dual de ser originadores y enrutadores de datos. Por lo tanto, las comunicaciones se llevan a cabo por dos razones:

Función Fuente: Los nodos como fuente de información de eventos llevan a cabo funcionalidades de comunicación, con el fin de transmitir sus paquetes con la información recolectada al sumidero. No siempre el nodo va a tener conexión directa con el sumidero, por lo tanto, se requiere de enrutamiento.

Función de Enrutamiento: Los nodos sensores, también participan en el reenvío de los paquetes recibidos de otros nodos al siguiente destino en el camino de red multisalto hacia el sumidero.

Como se puede observar en la Figura 1-4, el sumidero puede comunicarse con el administrador de tareas/usuario final *(task manager/end-user)* ya sea vía Internet, vía satélite o cualquier tipo de red inalámbrica e inclusive sin una red intermedia, es decir, el sumidero puede estar directamente conectado a los usuarios finales.

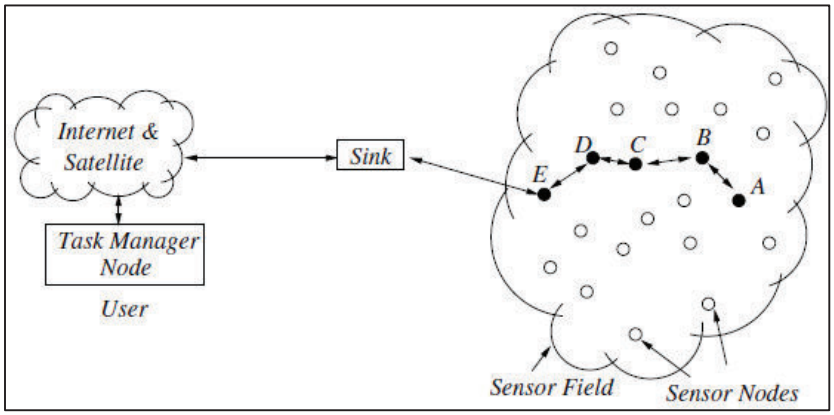


Figura 1-4 Arquitectura de una WSN según [16]

1.1.3.2 Componentes de los nodos sensores

Según Dargie y Poellabauer [17], se considera que los nodos sensores son el elemento central de una WSN. De hecho, es a través de un nodo sensor que el sensado, el procesamiento y las comunicaciones se llevan a cabo.

El nodo almacena y ejecuta los protocolos de comunicación y los algoritmos de procesamiento de datos. La calidad, tamaño y frecuencia de los datos sensados que pueden ser extraídos de la red, dependen de los recursos físicos disponibles para el nodo sensor [18].

Según [17] un nodo sensor está conformado por los subsistemas de sensado, procesamiento, comunicaciones y potencia. El diseñador tiene una plétora de opciones, en cuanto a decidir cómo construir y disponer juntos estos subsistemas unificados en un nodo programable. El subsistema de procesamiento es el elemento central del nodo y el procesador determina el equilibrio entre flexibilidad y eficiencia (en términos de energía y desempeño).

Existen varias opciones que pueden hacer las veces de procesador: microcontroladores, procesadores digitales de señal, circuitos integrados de

aplicación-específica, y arreglos de compuertas programables en campo (FPGA - *Field Programmable Gate Array*).

Se tiene un sinnúmero de maneras para conectar el subsistema de sensado con el procesador. Algunos sensores tienen su propio conversor analógico digital (ADC - *Analog to Digital Converter*) incorporado, el cual puede ser directamente conectado con el procesador a través de un protocolo estándar chip a chip. La mayoría de microcontroladores tienen uno o más ADC internos como interfaz de dispositivos analógicos.

Así mismo, el subsistema de comunicación puede ser interconectado con el subsistema de procesamiento de diferentes formas. Una alternativa es usar la interfaz periférica serial (SPI - *Serial Peripheral Interface*).

Algunos transceptores tienen su propia placa procesadora para llevar a cabo procesamiento de señal a bajo nivel perteneciente a las capas física y de enlace de datos, por lo tanto, alivia al procesador principal de sus ocupaciones.

El subsistema de comunicaciones es el subsistema de mayor consumo energético y su consumo de potencia debe ser regulado.

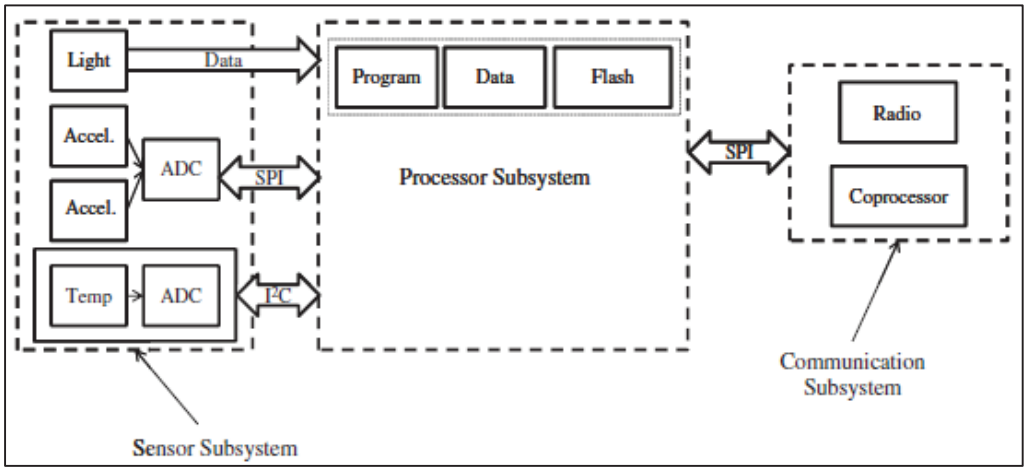


Figura 1-5 Arquitectura de un nodo sensor inalámbrico [17]

Casi todos los transceptores comercialmente disponibles proveen funcionalidades de control para conmutar el transceptor entre varios niveles activos de operación; estados de inactividad (*idle*) y reposo (*sleep*).

El subsistema de potencia provee potencia de corriente continua (DC – *Direct Current*) a todos los otros subsistemas para alimentar sus componentes activos tales como osciladores de cristal, amplificadores, registradores y contadores. Además, este subsistema provee conversores DC a DC de tal manera que cada subsistema puede obtener la cantidad adecuada de voltaje de alimentación.

En la Figura 1-5, empezando desde la esquina superior izquierda se puede ver un sensor de luz denominado *light* (luz) que envía datos al subsistema de procesamiento.

A continuación, se pueden observar dos sensores, uno de aceleración y uno de temperatura conectados a ADC. Algunos, sensores tienen su propio ADC incorporado, el cual puede ser directamente comunicado con el procesador a través de un protocolo estándar chip-a-chip.

La mayoría de microcontroladores tienen uno o más ADC internos como interfaz para dispositivos analógicos. El ADC convierte la salida de un sensor (la cuál es una señal análoga continua) en una señal digital mediante un proceso de 2 etapas: cuantización y muestreo (aquí suele ser necesario el uso del sobremuestreo por el ruido característico en entornos de implementación de WSN)

La transferencia rápida y energéticamente eficiente de datos entre los subsistemas de un nodo WSN es crítica respecto a la eficiencia promedio de la red que configura.

Sin embargo, el tamaño del nodo pone una restricción a los buses. Mientras las comunicaciones vía un bus paralelo son más rápidas que las de bus serial, un bus paralelo necesita más espacio.

Por el tamaño de los nodos sensores los buses paralelos no son soportados en su diseño, es así que las opciones se reducen a: interfaz periférica serial (SPI – *Serial Peripheral Interface*), las entradas/salidas de propósito general (GPIO – *General Purpose Input/output*), las entradas/salidas de datos seguros (SDIO – *Secure Data Input/output*), el circuito inter-integrado (I^2C – *Inter-Integrated Circuit*) y el bus serial

universal (USB – *Universal Serial Bus*). Entre estos, los buses más comunes son el SPI y el I^2C.

El principal objetivo del subsistema de procesamiento es procesar (ejecutar) instrucciones que tienen que ver con tareas de sensado, comunicación y auto-organización.

El subsistema de procesamiento consiste de un chip procesador, una memoria no volátil (usualmente se trata de una memoria flash interna) para guardar las instrucciones de programa, una memoria activa para almacenamiento temporal de los datos sensados, y un reloj interno entre otras cosas.

José Linares [18] explica que los nodos que conforman una WSN también se los denomina motas y típicamente están equipados por los siguientes componentes:

Sensores: Miden alguna de las magnitudes físicas, entre las cuales podemos nombrar a manera de ejemplo temperatura, humedad, luz, etc…

Actuadores: Dispositivo mecánico cuya función es proporcionar fuerza para mover o "actuar" sobre otro dispositivo mecánico.

Conversor análogo-digital: Su función consiste en convertir la magnitud analógica que proviene de los sensores en una señal digital que pueda ser procesada por un microcontrolador.

Habitualmente, el conversor analógico-digital está integrado en el microcontrolador.

Microcontrolador: Es el componente que tiene la tarea de procesar la información que viene de los sensores.

Radio-Transceptor de comunicaciones: Es el dispositivo encargado de enviar y recibir información desde/hacia otras motas utilizando el canal de radio.

Conector USB: Su función es conectar la mota con un PC para compilar la aplicación y extraer resultados.

1.1.3.3 Plataformas de nodos sensores

Según Akyildiz y Can Vuran [19] se ha obtenido información de una amplia variedad de plataformas desarrolladas en los años recientes incluyendo Mica2, Cricket, MicaZ, Iris, Telos, SunSPOT e Imote2. La Tabla 1-2 resume el hardware de varias motas.

Mote type	CPU speed (MHz)	Prog. mem. (kB)	RAM (kB)	Radio freq. (MHz)	Tx. rate (kbps)
Berkeley [3]					
WeC	8	8	0.5	916	10
rene	8	8	0.5	916	10
rene2	8	16	1	916	10
dot	8	16	1	916	10
mica	6	128	4	868	10/40
mica2	16	128	4	433/868/916	38.4 kbaud
micaz	16	128	4	2.4 GHz	250
Cricket [3]	16	128	4	433	38.4 kbaud
EyesIFX [17]	8	60	2	868	115
TelosB/Tmote [3]	16	48	10	2.4 GHz	250
SHIMMER [16]	8	48	10	BT/2.4 GHz[a]	250
Sun SPOT [9]	16–60	2 MB	256	2.4 GHz	250
BTnode [1]	8	128	64	BT/433–915[a]	Varies
IRIS [3]	16	128	8	2.4 GHz	250
V-Link [15]	N/A	N/A	N/A	2.4 GHz	250
TEHU-1121 [7]	N/A	N/A	N/A	0.9/2.4 GHz	N/A
NI WSN-3202 [6]	N/A	N/A	N/A	2.4 GHz	250
Imote [3]	12	512	64	2.4 GHz (BT)	100
Imote2 [3]	13–416	32 MB	256	2.4 GHz	250
Stargate [3]	400	32 MB	64 MB SD	2.4 GHz	Varies[b]
Netbridge NB-100 [3]	266	8 MB	32 MB	Varies[b]	Varies[b]

[a] BTnode and SHIMMER motes are equipped with two transceivers: Bluetooth and a low-power radio.
[b] The transmission rate of the Stargate board and the Netbridge depends on the communication device connected to it (MicaZ node, WLAN card, etc.).

Tabla 1-2 Resumen de las características principales del hardware de varias motas [19]

En la Figura 1-6, se puede apreciar una línea de tiempo con los años de lanzamiento de las diferentes plataformas de nodos sensores:

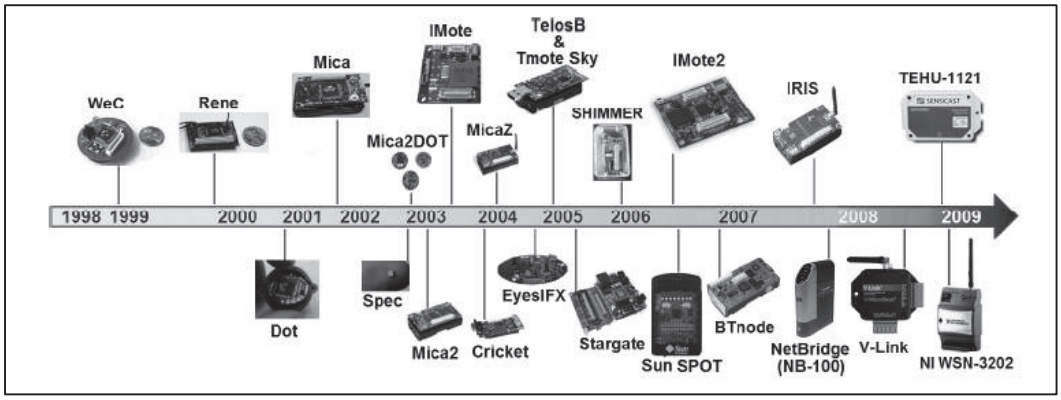


Figura 1-6 Años de lanzamiento de varias plataformas de nodos sensores [19]

1.1.3.3.1 La familia Mica

Esta familia consiste en los nodos: Mica, Mica2, MicaZ e IRIS[4] y son producidos por Crossbow. Cada nodo está equipado con microcontroladores AVR[5] Atmel de 8 bits, con una velocidad de 4 a 16 MHz y flash programable de 128 kB a 256 kB. Mientras que los microcontroladores son similares, la familia de nodos Mica ha sido equipada con una amplia gama de transceptores [19].

El nodo Mica incluye un transceptor de 916 MHz o de 433 MHz a 40 kbps, mientras la plataforma Mica2 está equipada con un transceptor de 433/868/916 MHz a 40 kbps [19].

Por otro lado, los nodos MicaZ e IRIS están equipados con transceptores compatibles con el estándar IEEE[6] 802.15.4, los cuales operan a 2.4 GHz con una tasa de datos de 250 kbps.

Cada plataforma tiene memoria limitada en términos de RAM[7] (4-8 KB) y de memoria de datos (512 KB). Además, cada versión está equipada con un conector de 51 pines que es usado para conectar placas sensoras y de programación adicionales a la mota [19].

1.1.3.3.2 Telos/Tmote

Las motas Tmote Sky (también llamadas TelosB) de Sentilla (antes Moteiv) tienen una arquitectura similar a la MicaZ de Crossbow. Mientras el transceptor se mantiene intacto, las motas Telos/Tmote tienen más memoria RAM desde que usan un microcontrolador de 8 MHz llamado MSP430 con 10kB de RAM. Además, las plataformas Telos/Tmote son compatibles con varios sensores incluyendo luz,

---

[4] *Integrated Resource Information System* (Sistema de Información de Recursos Integrados) [24].
[5] Los creadores AVR no dan una respuesta definitiva en cuanto a lo que el término "AVR" significa. Sin embargo, se acepta comúnmente que AVR significa “el procesador **R**ISC de **A**lf (Egil Bogen) y **V**egard (Wollan).El término “AVR” se usa comúnmente para referirse a la línea de microcontroladores de 8-bits de la compañía ATMEL [97]
[6] *Institute of Electrical and Electronics Engineers* (Instituto de Ingenieros Eléctricos y Electrónicos).
[7] *Random Access Memory* (Memoria de Acceso Aleatorio).

infrarrojo, humedad y temperatura, así como también un conector USB[8], el cuál elimina la necesidad de sensores adicionales o placas de programación. Aun así, conectores de 6 y 10 pines son incluidos para sensores adicionales [19].

1.1.3.3.3 EYES

Su desarrollo tomó 3 años y su arquitectura es similar a la de las motas Telos/Tmote, EYES[9] posee un microcontrolador de 16 bits, 60kB de memoria para uso de programas y 2kB de memoria para datos. Además, los siguientes sensores están embebidos en la mota: brújula, acelerómetro, temperatura y presión.

La plataforma EYES incluye el transceptor TR1001, el cuál soporta una velocidad de transmisión por encima de los 115.2 kbps con un consumo de potencia de 14.4 mW, 16.0 mW, y 15.0 uW durante los modos de recepción, transmisión y reposo respectivamente. La plataforma incluye una interfaz serial RS232 [10] para la programación [19].

1.1.3.3.4 Stargate

La placa Stargate es una plataforma de alto desempeño diseñada para el sensado, procesamiento de señal, control y gestión de la red de sensores. Stargate está basado en un procesador Intel PXA[11] 255 Xscale 400 MHz RISC, el cuál es el mismo procesador encontrado en muchas computadoras de bolsillo, incluyendo la COMPAQ iPAQ[12] y la Dell Axim. Stargate posee 32 MB de memoria flash, 64 MB de SDRAM[13] y un conector en la placa para las motas de la familia Mica, así como también tarjetas PCMCIA (*Personal Computer Memory Card International Association*) Bluetooth o IEEE 802.11.

---

[8] *Universal Serial Bus* (Bus Serial Universal).
[9] ***E**nerg**Y** **E**ficient **S**ensor networks* (Redes de sensores de consumo eficiente de energía) [98].
[10] ***R**ecommended **S**tandard* **232** (Estándar recomendado 232), es un protocolo de comunicación estándar para conectar ordenadores y dispositivos periféricos permitiendo el intercambio de datos de forma serial.
[11] Familia de **P**rocesadores de **A**plicaciones, basados en la tecnología de Intel Xscale [100].
[12] Es un ordenador personal de bolsillo cuyo sistema operativo es Windows Mobile [101].
[13] ***S**ynchronous **D**ynamic **R**andom **A**ccess **M**emory* (Memoria dinámica de acceso aleatorio) [102].

Por lo tanto, dicha placa puede trabajar como un Gateway inalámbrico y como un *hub* para procesamiento de algoritmos de red.

Cuando se la conecta a una webcam u otro dispositivo de captura, este puede funcionar como un sensor multimedia de media-resolución, a pesar de que su consumo de energía es todavía alto.

Stargate NetBridge fue desarrollado como sucesor de Stargate y está basado en el procesador Intel IXP[14] 420 Xscale que trabaja a 266 MHz. Entre sus características podemos encontrar: un puerto Ethernet alámbrico, dos puertos USB 2.0, además de 8 MB de memoria flash, 32 MB de RAM y sistema de disco USB 2.0 de 2 GB, donde se puede correr un sistema operativo GNU/Linux. Usando los puertos USB, un nodo sensor puede ser conectado para funcionalidades de Gateway [19].

1.1.3.3.5 Imote e Imote2

Intel ha desarrollado dos generaciones de prototipos de sensores inalámbricos, que son conocidos como Imote e Imote2 para sensado de alto desempeño y aplicaciones de Gateway.

Imote está construido alrededor de un microcontrolador inalámbrico integrado que consiste de un procesador ARM7[15] de 8 bits a 12 MHz, un radio Bluetooth, 64 kB de RAM, y 32 kB de memoria flash, así como también varias opciones de I/O (Entradas/Salidas). La arquitectura del software está basada en TinyOS[16].

La segunda generación Imote2, está construida alrededor de un procesador de baja potencia de 32 bits llamado PXA271 Xscale a 320/416/520 MHz, el cual está habilitado para realizar operaciones de procesamiento digital de señales para almacenamiento y compresión, y posee un radio ChipCon IEEE 802.15.4 CC2420.

---

[14] Familia de **P**rocesadores de **R**ed, basados en la tecnología de Intel *Xscale*
[15] Familia de procesadores embebidos de 32 bits, ampliamente utilizado (10 billones de unidades lanzadas desde su lanzamiento en 1994) [103].
[16] TinyOS es un sistema operativo, diseñado para sistemas embebidos inalámbricos de baja potencia [104].

Además, posee una RAM incorporada de mayor capacidad y memoria flash (32 MB), soporte de alternativas de radio, y una variedad de Entradas/Salidas de alta velocidad para conectar sensores digitales o cámaras. Su tamaño es también muy limitado, 40 x 33 mm, y puede correr un sistema operativo GNU/Linux y aplicaciones Java [19].

1.1.3.3.6 Waspmote

En el año 2009 la empresa Libelium de Zaragoza – España lanzó al mercado su mota denominada Waspmote, cuyo módulo principal dispone de un microcontrolador ATmega1281 que funciona a 8 MHz y dispone de 8 KB de memoria RAM y 128 KB de memoria FLASH.

En conjunto con el microcontrolador, la placa principal cuenta con un reloj en tiempo real (con su batería de respaldo) y de un slot microSD [20] [21].

Para comercializar sus dispositivos, Libelium ha dejado libres varios conectores del dispositivo con el fin de que el usuario adquiera lo que necesite y lo adicione a su placa según sus necesidades.

Se puede añadir un módulo XBee[17] o un módulo Bluetooth para comunicaciones de radio, además de un módulo GSM/GPRS[18] para el envío y recepción de datos a través de la red GSM, adicionalmente se le puede añadir un módulo GPS con su correspondiente antena.

El principal inconveniente de Waspmote es que no es compatible con Contiki[19] ni con TinyOS, por lo que se depende del fabricante para realizar personalizaciones y actualizaciones [22]. El Ing. Carlos Egas Msc., codirector del presente proyecto posee estas motas y ha trabajado con ellas en varios proyectos de titulación.

---

[17] Es un módulo de comunicación inalámbrica construido en base al estándar 802.15.4 (ZigBee) [21].
[18] GSM es un estándar global de comunicaciones móviles para aplicaciones que no requieren altas tasas de datos. Originalmente fue diseñado para soportar tráfico de voz, con GPRS se le incluyo comunicaciones usando además paquetes de datos [22].
[19] Es un sistema operativo diseñado para el “Internet de las cosas”, conectando microcontroladores pequeños de baja potencia y bajo costo [105].

1.1.3.3.7 Zolertia

Las motas Z1 datan del año 2009 y son fabricadas por la empresa Zolertia. Son motas de bajo consumo de energía y sirven como plataforma de propósito general para diseñadores, investigadores, entusiastas y aficionados de las WSN. Z1 dispone de soporte para sistemas operativos de código abierto implementados por la comunidad WSN, como TinyOS y Contiki. Es compatible con protocolos estándar ZigBee y el IEEE 802.15.4[20] .

El núcleo de la arquitectura se basa en la familia de microcontroladores y transceptores radio MSP430+CC2420 de Texas Instruments, el cual lo hace compatible con motas basadas en esta misma arquitectura.

Sin embargo, la unidad microcontroladora (MCU – *Microcontroller Unit*) en Z1 es el MSP430F2xxx de segunda generación en lugar de MSP430F1xxx, como es habitual en otras motas como la TelosB de Crossbow, Tmote de Moteiv y similares. Este hecho implica algunas incompatibilidades debido a los cambios internos en el microcontrolador, sin embargo, estas pequeñas diferencias no se aprecian en el ámbito de aplicación si se utiliza un sistema operativo compatible como Contiki [23]. Las características de las motas Z1 son las siguientes:

Características principales de las motas Z1	
Temperatura de operación	De -40°C a 85°C
Conector de expansión	52 pines (más sensores)
Microcontrolador	Familia MSP430 de 2da gen. 16 bits/16 MHz
Frecuencia de trabajo	2,4 GHz
Transceptor de RF	CC2420
Tasa máx. efectiva de TX	250 Kbps
Compatibilidad	6loWPAN y ZigBee
Sensor de temperatura	Digital de bajo consumo TMP102

Tabla 1-3 Características principales de las motas Z1 (parte 1 de 2) [22]

[20] La principal diferencia entre ZigBee y 802.15.4 es que el primero trabaja en capa 3 y el segundo en la capa 2, ambas capas del modelo OSI [23].

Características principales de las motas Z1	
Acelerómetro	Digital de 3 ejes ADXL345
Acelerómetro	Digital de 3 ejes ADXL345
Interacción con el usuario	3 leds y 2 pulsadores
Puerto de intercambio de datos	Micro USB
Memoria Flash	16 MB
Antena externa	Opcional
Compatibilidad con sensores	Phidgets (entre otros)

Tabla 1-4 Características principales de las motas Z1 (parte 2 de 2) [22]

A continuación, se muestra una tabla comparativa de dos motas de la familia Mica, TelosB, Imote2, Zolertia y Waspmote.

Mota		MICA2	MICA2 Dot	TelosB	Imote2	Zolertia Z1	Waspmote
MCU	Chip	ATmega128L		Msp30f1611	FXA271	MSP430F2167	ATmega128L
	Tipo	8 bits		16 bits	32 bits	16 bits	8 bits
	Memoria Programa	128 KB		48 KB	N/A	96 KB	128 KB
	RAM	4 KB		10 KB	32768 KB	96 KB	128 KB
Almacenamiento Externo No Volátil	Chip	AT45DB014B		M25P80	N/A	N/A	N/A
	Conexión	SPI		SPI	N/A	N/A	SPI
	Tamaño	512 KB		1024 KB	32768 KB	N/A	SD hasta 2000 KB
Sistema de Alimentación	Tipo	2xAA	Coin Cell	2xAA	3xAAA	2xAA/A ó Coin Cell	Variable
	Capacidad Típica	2850 mAh	1000 mAh	2850 mAh	1100 mAh	1100 mAh	2000 mAh
RF	Chip	CC1000		CC2420	CC2420	CC2420	Variable (Xbee)
	Frecuencia	868/916,433 ó 315 MHz		2,4GHz	2,4GHz	2,4GHz	2,4GHz
	Datarate	38,4 Kbps		250 Kbps	250 Kbps	250 Kbps	250 Kbps
	Potencia Transmisión	5 dBm		0 dBm	0 dBm	0 dBm	18 dBm
	Alcance Exterior	152,4 m	152,4 – 304,8 m	75 – 100 m	~30 m	150 m	300 m
Año comercialización		2002	2002	2005	2007	2009	2009

Tabla 1-5 Tabla comparativa de las motas de la familia Mica, TelosB, Imote2, Zolertia y Waspmote [22]

El valor mostrado como “alcance exterior” no es absoluto, debido a que depende de las condiciones que impone el medio a las comunicaciones entre los nodos.

1.1.4 COMUNICACIONES EN LAS REDES INALÁMBRICAS DE SENSORES

1.1.4.1 Pila de protocolos de las redes inalámbricas de sensores

En la Figura 1-7 explicada en [16], se puede observar la pila de protocolos usados por el sumidero y todos los nodos sensores de la WSN.

En la pila de protocolos de la Figura 1-7 se combina la priorización del consumo de potencia y enrutamiento, integración de datos y protocolos de red, transferencia de potencia eficientemente a través del medio inalámbrico y promueve los esfuerzos cooperativos de los nodos sensores. Dentro de la pila de protocolos se encuentra las siguientes capas: capa física, capa de enlace de datos, capa de red, capa de transporte, capa de aplicación, así como plano de sincronización, plano de localización, plano de gestión de la topología, plano de gestión de potencia, plano de gestión de movilidad y plano de gestión de tareas.

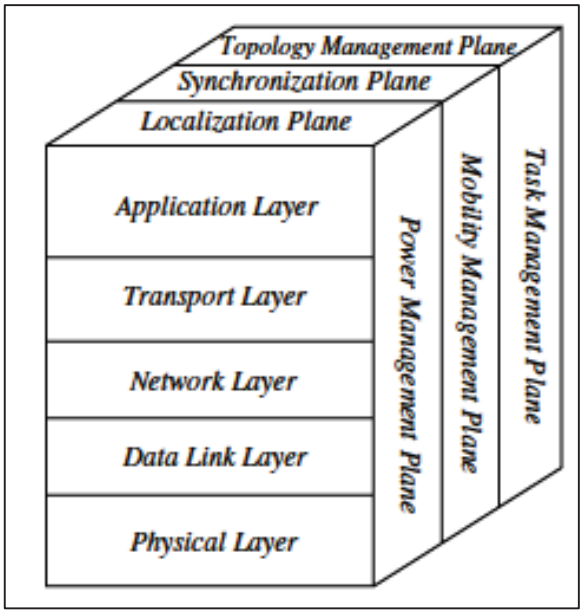


Figura 1-7 Pila de protocolos de una WSN [16]

A continuación, se presenta un resumen que explica brevemente la Figura 1-7, para luego detallar cada capa en los sub ítems del sub capítulo 1.1.4.1.

La capa física direcciona las necesidades de simples, pero robustas técnicas de modulación, transmisión y recepción. Debido a que el entorno es ruidoso y que los nodos sensores pueden ser móviles.

La capa de enlace de datos es la responsable de asegurar comunicaciones confiables a través de técnicas de control de errores y gestión de acceso al canal a través de MAC (*Media Access Control* – Control de Acceso al Medio) para minimizar la colisión por difusión con los vecinos. Dependiendo de las tareas de sensado, diferentes tipos de aplicación de software pueden ser construidos y usados en la capa de aplicación.

La capa de red se encarga del enrutamiento de los datos que le provee la capa de transporte. La capa de transporte ayuda a mantener el flujo de datos si la aplicación de la WSN lo requiere así. Además, los planos de potencia, movilidad y gestión de tareas monitorean la potencia, el movimiento y la distribución de tareas entre los nodos sensores. Estos planos ayudan a los nodos sensores a coordinar las tareas de sensado y minimizar el consumo promedio de energía.

El plano de gestión de potencia su cometido es gestionar el consumo de energía del nodo. Por ejemplo, el nodo sensor puede apagar su receptor después de recibir un mensaje de uno de sus vecinos, para evitar tener mensajes duplicados. También, cuando el nivel de potencia de un nodo sensor es bajo, dicho sensor difunde a sus vecinos que su potencia es baja y que no puede participar en los mensajes de enrutamiento. De hecho, la potencia sobrante es reservada para el sensado.

El plano de gestión de movilidad detecta y registra los movimientos de los nodos sensores, de tal manera que una ruta de regreso hacia el usuario es siempre mantenida, y los nodos sensores pueden mantener la pista de sus vecinos. Mediante el conocimiento de estos nodos sensores vecinos, los nodos sensores pueden balancear su potencia y uso de tareas.

El plano de gestión de tareas balancea y planifica las tareas de sensado dadas a una región específica. No todos los nodos sensores en dicha región son requeridos para llevar a cabo tareas de sensado al mismo tiempo.

Como resultado, algunos nodos sensores llevan a cabo más tareas que otros, dependiendo de su nivel de potencia.

De hecho, éstos planos de gestión son necesarios para que los nodos sensores trabajen de manera conjunta y lograr así la eficiencia energética, a través del enrutamiento de datos en una red de sensores móviles, y compartiendo recursos entre los nodos sensores.

Sin embargo, al no existir estos planos de gestión, cada nodo sensor tan solo trabajaría individualmente. Además, el tiempo de vida útil de toda la WSN puede ser prolongado debido a que sería más eficiente si los nodos sensores pueden colaborar con los otros.

Se detalla a continuación las funciones de las capas presentadas en la Figura 1-7.

1.1.4.1.1 Capa Física

La capa física es responsable por la selección de frecuencia, generación de frecuencia portadora, detección de señal, modulación y encriptación de datos. La generación de frecuencia y la detección de señal tienen más que ver con el hardware subyacente y el diseño del transceptor [16].

1.1.4.1.2 Capa de enlace de datos

La capa de enlace de datos es responsable de multiplexar el flujo de datos, detección de tramas de datos, control de errores y de acceso al medio. Adicionalmente, asegura las conexiones confiables punto a punto y punto a multipunto en una red de comunicaciones [16].

1.1.4.1.2.1 MAC

El protocolo de control de acceso al medio en una red de sensores inalámbrica multisalto y auto organizada, debe cumplir los siguientes objetivos. El primer objetivo consiste en la creación de una infraestructura de red. De hecho, cuando se tiene miles de nodos sensores, éstos pueden ser densamente esparcidos en un campo de sensado.

Consecuentemente, el esquema MAC debe establecer enlaces de comunicación para la transferencia de datos y así formar la infraestructura básica necesaria para la comunicación inalámbrica salto-a-salto que provea la capacidad de auto

organización. El segundo objetivo se enfoca en compartir recursos de comunicación eficientemente entre los nodos sensores, los cuales incluyen tiempo, energía y frecuencia.

Sin importar el esquema de acceso al medio, la eficiencia energética es la de mayor importancia. Un protocolo MAC debe indudablemente apoyar modos de operación de ahorro de energía para el nodo sensor. De hecho, el más obvio concepto de conservación energética consiste en apagar el transceptor cuando no es requerido.

A pesar de que este método de ahorro de energía aparentemente provee significantes ganancias en energía, puede dificultar la conectividad de la red. Por ejemplo, una vez que el transceptor es apagado, el nodo sensor no puede recibir ningún paquete de sus vecinos, es decir aparecerá desconectado de la red.

Por otra parte, encender y apagar el módulo de radio tiene gastos generales, en términos de consumo de energía, debido a los procedimientos de encendido y apagado requeridos por hardware y software. De hecho, si el módulo de radio es apagado durante cada intervalo en el que se halle desocupado, durante un periodo de tiempo el nodo sensor podría terminar gastando más energía que si estuviera encendido.

Como resultado la operación en un modo de ahorro de energía, es eficiente en términos de energía, solo si el tiempo gastado en ese modo es más grande que un cierto límite. Puede existir un número de modos útiles de operación para el nodo sensor inalámbrico, dependiendo del número de estados del microprocesador, memoria, conversor analógico digital y el transceptor. Cada uno de estos modos, puede ser caracterizado por su consumo de potencia y la sobrecarga de latencia, los cuales se definen como la transición de potencia de un modo a otro [16].

1.1.4.1.2.2 Control de errores

Otra importante función de la capa de enlace de datos se encarga del control de errores de los datos transmitidos. Dos modos importantes de control de errores en redes de comunicaciones son FEC (*Forward Error Correction* – Corrección de errores

hacia adelante) y ARQ (*Automatic Repeat Request* – Petición de repetición automática).

La utilidad de ARQ en las aplicaciones de una red de sensores es limitada por el costo adicional de retransmisión y sobrecarga. Por otra parte, la complejidad de decodificación es mayor en FEC, a medida que las capacidades de corrección de errores necesitan ser incorporadas.

Consecuentemente, los códigos de control de errores simples con baja complejidad de codificación y decodificación presentan la mejor solución para las redes de sensores. En el diseño de tal esquema, es importante tener un buen conocimiento de las características del canal y las técnicas de implementación [16].

1.1.4.1.3 Capa de red

Los nodos sensores son esparcidos densamente en un campo cualquiera (lugar donde se colocan los nodos, por ejemplo, un bosque) cerca de, o dentro de algún fenómeno (por ejemplo, variaciones de temperatura por un incendio en el bosque mencionado).

La información recolectada relacionada al fenómeno debe ser transmitida al sumidero, el cual podría estar localizado lejos del campo de sensado. Sin embargo, el rango de comunicaciones limitado por los nodos sensores previene la comunicación directa entre cada nodo sensor y el nodo fuente.

Por tal motivo, se requiere de protocolos eficientes de enrutamiento inalámbrico multisalto entre los nodos sensores y el nodo sumidero a través de nodos sensores intermedios como retransmisores.

Las técnicas de enrutamiento existentes, que han sido desarrolladas para las redes inalámbricas ad-hoc, usualmente no satisfacen los requerimientos de las WSN. La capa de red de las WSN es usualmente diseñada de acuerdo a los siguientes principios:

- La eficiencia de potencia es siempre una consideración importante.

- Las redes de sensores son mayoritariamente centradas en los datos.
- Además del enrutamiento, los nodos de retransmisión pueden agregar datos de múltiples vecinos a través del procesamiento local.
- Debido al número extenso de nodos en una WSN, el ID único para cada nodo puede no ser provisto y los nodos pueden necesitar direccionamiento basado en sus datos o localización.

Otro importante problema para enrutar en las WSN, ocurre debido a que el enrutamiento puede ser basado en consultas centradas en los datos. Basado en la información requerida por el usuario, el protocolo de enrutamiento debe direccionar diferentes nodos que proveerían la información requerida. Específicamente, los usuarios están más interesados en consultar un atributo del fenómeno, que consultar un nodo individual. Por ejemplo, “las áreas donde la temperatura esta sobre los 21ºC” es una consulta más común que “la temperatura leída por el nodo #47”.

Otra función importante de la capa de red consiste en proveer interconexión con redes externas, tales como: con otras redes de sensores, sistemas de comando y control, y la Internet.

En un escenario, el nodo sumidero puede ser usado como *gateway* para otras redes, mientras que, en otro escenario, sería usado para crear un backbone mediante la conexión conjunta de nodos fuente y proveer el acceso hacia otras redes vía el gateway [16].

1.1.4.1.3.1 El TSP aplicado a las WSN

En el TSP clásico, un agente viajero debe encontrar el camino de menor costo para recorrer un conjunto de ciudades, visitando cada una de ellas una sola vez y finalmente regresar a su ciudad de origen.

Al aplicarse el problema del agente viajero a una WSN, las ciudades se convierten en nodos sensores, el agente viajero pasa a ser un paquete de datos, la ciudad de origen del recorrido se torna en el nodo sumidero y los costos de las rutas a recorrerse para resolver el problema, podrían ser cuantificados al evaluar la calidad

del enlace según los parámetros que la aplicación de la WSN considere críticos (consumo de energía, BER[21], LQI[22], etcétera).

Al evaluar rutas eficientes, el problema es concerniente a la capa de red (enrutamiento), además en 3.1.3.1.4 se detalla el uso de enrutamiento estático para direccionar los paquetes según los resultados de la resolución del TSP en la WSN simulada.

Como ya se señaló anteriormente, el tiempo de vida útil para los nodos sensores aumenta si colaboran entre ellos compartiendo recursos de comunicación, por tal motivo el estudio de la resolución del TSP en una WSN está orientado a comunicar a todos los nodos que pertenezcan a la WSN de manera eficiente considerando el nodo fuente como el nodo principal.

1.1.4.1.4 Capa de transporte

La capa de transporte es necesaria cuando se desea acceder a la red a través de Internet u otras redes externas. TCP (*Transmision Control Protocol* - Protocolo de control de la transmisión), con sus mecanismos de ventana deslizante, no satisface los desafíos únicos planteados por el entorno de las WSN.

A diferencia de protocolos tales como TCP, los esquemas de comunicación extremo-a-extremo en WSN no están basados en un direccionamiento global. Dichos esquemas, deben considerar que el direccionamiento basado en datos o ubicación, es utilizado para indicar los destinos de los paquetes de datos.

Además, factores como consumo de potencia, escalabilidad y características como enrutamiento centrado en datos, significan que las redes de sensores necesitan un manejo diferente en la capa de transporte. Por lo tanto, estos requerimientos fomentan la necesidad de crear nuevos tipos de protocolos en la capa de transporte, por lo tanto este tema se puede plantear en investigaciones futuras.

---

[21] *Bit Error Ratio* – Tasa de bits errados
[22] *Link Quality Indicator* – Indicador de calidad del enlace

El desarrollo de protocolos para la capa de transporte, es una tarea desafiante debido a que los nodos sensores pueden ser influenciados por restricciones del hardware, tales como la potencia y memoria limitadas.

Como resultado, cada nodo sensor está imposibilitado de almacenar grandes cantidades de datos como un servidor de Internet, y los acuses de recibo son muy costosos para las redes de sensores.

Por lo tanto, nuevos esquemas que dividan la comunicación extremo-a-extremo, probablemente en los sumideros, puedan ser necesarios donde los protocolos tipo UDP (*User Datagram Protocol* - Protocolo de Datagrama de Usuario) son usados en la red de sensores.

Para la comunicación dentro de una WSN, los protocolos de la capa de transporte son requeridos para dos funcionalidades principalmente: confiabilidad y control de congestión. Los recursos limitados y los altos costos de energía evitan que los mecanismos de confiabilidad extremo-a-extremo sean empleados en las WSN. En su lugar, los mecanismos de confiabilidad localizados son necesarios.

Por otra parte, la congestión que pueda ocurrir por el alto tráfico durante los eventos, debería ser mitigada por los protocolos de la capa de transporte. Debido a que los nodos sensores son limitados en términos de procesamiento, almacenamiento y consumo de potencia, los protocolos de la capa de transporte aspiran a explotar las capacidades colaborativas de estos nodos sensores y mover la inteligencia al sumidero antes que a los nodos sensores [16].

1.1.4.1.5 Capa de aplicación

La capa de aplicación incluye la funcionalidad principal, así como también varias funcionalidades de gestión. Además del código de la aplicación que es específico para cada implementación, las funcionalidades de gestión de red y de procesamiento de consultas también residen en esta capa.

La arquitectura de pila de capas ha sido inicialmente adoptada en el desarrollo de las WSN debido a su éxito con Internet. Sin embargo, las implementaciones a gran

escala de las aplicaciones WSN revelan que el canal inalámbrico tiene un impacto significativo en los protocolos de la capa más alta.

Además de las funcionalidades de comunicación, las WSN han sido equipadas con varias funcionalidades que ayudan a la operación de las soluciones propuestas. En una WSN, cada dispositivo sensado es equipado con su propio reloj local para operaciones internas.

Cada evento relacionado con la operación del dispositivo de sensado incluyendo la detección, procesamiento y comunicación, está asociado con la información de sincronización controlada a través del reloj local.

Ya que los usuarios están interesados en la información colaborativa de múltiples sensores, la información de sincronización asociada con los datos en cada dispositivo de sensado necesita ser consistente. Por otra parte, la WSN debe ser capaz de ordenar correctamente los eventos sensados por los sensores distribuidos y así modelar con precisión el entorno físico. Estos requerimientos de sincronización han conducido al desarrollo de los protocolos sincronización de tiempo en las WSN.

La cercana interacción con los fenómenos físicos requiere que la información de la ubicación esté asociada al tiempo. De hecho, las WSN están cercanamente relacionadas con los fenómenos físicos de sus alrededores. La información reunida necesita vincularse con la ubicación de los nodos sensores para proveer una vista precisa del campo de sensado observado.

Por otra parte, las WSN pueden usarse para rastrear ciertos objetos en aplicaciones de monitoreo, por lo que también se requiere que la información de la ubicación sea incorporada a los algoritmos de rastreo. Además, los servicios basados en la ubicación y los protocolos de comunicación requieren información de la posición. Por lo tanto, los protocolos de localización han sido incorporados a la pila de comunicaciones.

Posteriormente, varias soluciones del manejo de la topología se requieren para mantener la conectividad y la cobertura de la WSN. Los algoritmos del manejo de la

topología proveen métodos eficientes para el despliegue de la red, lo cual resulta en un tiempo de vida más prolongado y una cobertura de red más eficiente. Además, los protocolos de control de la topología ayudan a determinar los niveles de potencia transmitida, así como la duración de las actividades de los nodos sensores para minimizar el consumo de energía mientras se asegura la conectividad de la red. Adicionalmente, protocolos de agrupamiento se usan para organizar la red en grupos a fin de mejorar la escalabilidad y el tiempo de vida de la red.

La integración de cada uno de los componentes para una operación eficiente depende de las aplicaciones que corren sobre la WSN. Esta naturaleza dependiente, define varias propiedades únicas comparadas a las soluciones de red tradicionales. A pesar que, las investigaciones y desarrollo iniciales de las WSN se han centrado en la transferencia de datos en escenarios inalámbricos, existen otros escenarios. Estos, incluyen sensores inalámbricos y redes de actuadores, los cuales consisten de actuadores además de sensores que convierten la información sensada en acciones para actuar sobre el entorno, y redes inalámbricas de sensores multimedia, las cuales soportan tráfico multimedia en términos de información audio-visual, además de datos escalares. Igualmente, las WSN se han usado en ambientes difíciles, tales como ambientes subterráneos y submarinos dando lugar a la creación de redes inalámbricas de sensores submarinos y redes inalámbricas de sensores subterráneos [16].

1.1.4.2 Estandarización de las comunicaciones

1.1.4.2.1 IEEE 802.15.4

Según el Dr. Ian Akyildiz [24], la heterogeneidad en las plataformas de sensores disponibles resulta en problemas de compatibilidad para la realización de las aplicaciones previstas. Por lo tanto, es necesaria la estandarización de ciertos aspectos de las comunicaciones.

Con este fin, se formó el equipo IEEE 802.15.4 para la especificación de una tecnología de transceptor inalámbrica de baja velocidad de transmisión, con duración de batería extendida y de muy baja complejidad. Se escogieron tres diferentes

bandas para las comunicaciones: 2.4 GHz (global), 915 MHz (las Américas), y 868 MHz (Europa).

Mientras la capa física usa modulación por desplazamiento de fase binario (BPSK – *Binary Phase Shift Keying*) en las bandas de 868/915 MHz y modulación por desplazamiento de fase en cuadratura con corrimiento (OQPSK – Offset *Quadrature Phase Shift Keying*) en la banda de 2.4 GHz, la capa MAC provee comunicaciones para las topologías malla, estrella y basada en clústeres con la ayuda de controladores.

El rango de transmisión de los nodos va entre los 10 m y 100 m con tasas de datos de 20 a 250 Kbps, La mayoría de las plataformas desarrolladas al momento cumplen con el estándar IEEE 802.15.4. Este estándar ha adquirido una gran audiencia y ha llegado a ser un estándar de facto para las capas PHY y MAC en comunicaciones de baja potencia.

En la parte superior del estándar IEEE 802.15.4 varios equipos de estandarización han sido formados para la proliferación del desarrollo de redes de baja potencia en varias áreas. Es bien conocido que estándares como Bluetooth y WLAN no compaginan con aplicaciones de sensores de baja potencia. Por otro lado, intentos de estandarización, tales como ZigBee (Véase 1.1.4.2.2), WirelessHART[23], WINA[24] y SP100.11a[25], han direccionado específicamente las necesidades típicas de control inalámbrico y monitoreo de aplicaciones, para disponer de una rápida mejora de las WSN en la industria. Además, los esfuerzos de estandarización como 6LoWPAN (*IPv6 over Low power Wireless Personal Area Networks* – IPv6 sobre redes

---

[23] Es un protocolo de comunicaciones para redes inalámbricas en malla que trabajan con aplicaciones de automatización de procesos [80].

[24] *Wireless Industrial Network Alliance* (Alianza de redes industriales inalámbricas), su objetivo es promulgar los beneficios del uso de tecnologías inalámbricas en aplicaciones industriales, enfocándose en el usuario final [81].

[25] Es un protocolo propuesto por la ISA (*International Society of Automation* – Sociedad Internacional de Automatización) basado en 802.15.4, propuesto como solución para quipo de automatización industrial que requiera de redes inalámbricas [82]

inalámbricas de área personal de baja potencia) se enfocan en proveer compatibilidad entre las WSN y redes existentes tales como Internet.

1.1.4.2.2 ZigBee

En la tesis doctoral [25] se da a conocer que la alianza ZigBee, junto con el estándar IEEE 802.15.4 ha definido una pila de protocolos estándar para utilizarse en las WSN.

IEEE 802.15.4 define la capa física y de acceso al medio de la arquitectura, mientras que ZigBee específica la capa de red, el marco de la aplicación y los perfiles de los dispositivos utilizados, así como también servicios de seguridad. En la Figura 1-8, se muestra las diferentes capas de la arquitectura ZigBee, que se han diseñado para maximizar el tiempo de vida de las baterías en aplicaciones con baja tasa de envío de datos que requieren comunicaciones fiables.

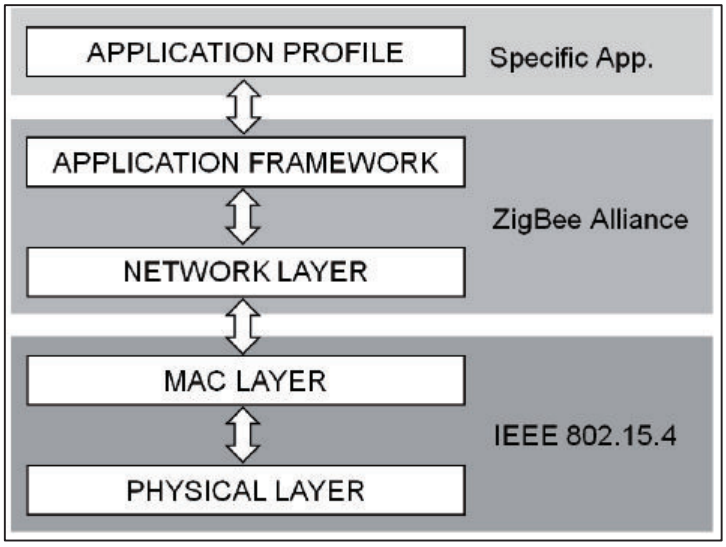


Figura 1-8 Arquitectura ZigBee [25]

Por otro lado, en [26] se advierte que el estándar ZigBee está definido específicamente en conjunción con el estándar IEEE 802.15.4, por consiguiente, ambos son usualmente confundidos. Además, se señala que cinco áreas de aplicación principal fueron concebidas para ZigBee: automatización de hogar, consumo de energía inteligente, automatización de la construcción, servicios de telecomunicación y cuidado de salud personal.

Para albergar una larga variedad de aplicaciones, se ha definido tres tipos de tráfico:

Primero, el tráfico de datos periódicos se requiere para aplicaciones de monitoreo, donde los sensores proveen información continua, ignorando el fenómeno físico. El intercambio de datos se controla a través del controlador de red o el enrutador.

Segundo, el tráfico de datos intermitentes se aplica a la mayoría de aplicaciones basadas en eventos y es accionada a través de la aplicación o de un factor externo (por ejemplo, la alerta accionada por un sensor ante la presencia excesiva de CO_2, como no existen incendios perpetuos que exijan trabajar a la WSN todo el tiempo, su tráfico se vuelve intermitente). Este tipo de tráfico se maneja a través de cada nodo enrutador y para ahorrar energía los dispositivos deben operar en modo desconectado, mientras que la mayoría del tiempo los nodos operan en modo *sleep*. Todas las veces que se necesita transmitir la información, el transceptor se enciende y los dispositivos se asocian con la red.

Finalmente, el tráfico de datos repetitivos de baja latencia se define por ciertas comunicaciones como el clic de un mouse que necesita ser completado dentro de un determinado tiempo. Este tipo de tráfico se trata a través de una estructura de trama basada en sondeo definida por el estándar IEEE 802.15.4.

Adicionalmente, la capa de red ZigBee provee funcionalidades de gestión para la operación de la red. Dependiendo de la operación de red, la pila de comunicación de cada dispositivo puede configurarse, ya que los dispositivos ZigBee pueden ser parte de redes diferentes durante su tiempo de vida, el estándar también define un mecanismo de direccionamiento flexible.

Como resultado, el ID único de cada dispositivo no se usa para la comunicación, sino que se asigna una dirección más corta para mejorar la eficiencia durante la comunicación.

En una arquitectura de árbol, la dirección del dispositivo también identifica su “padre”, esta dirección se usa para propósitos de enrutamiento. La capa de red ZigBee también provee sincronización entre dispositivos y controladores de red. Finalmente,

las rutas multisalto se generan en la capa de red ZigBee de acuerdo a los protocolos definidos.

Como se muestra en la Figura 1-9, el estándar ZigBee también define ciertos componentes en la capa de aplicación. Esta capa consiste en la subcapa APS (*Application Support Sub-layer* – Subcapa de Soporte de Aplicación), el objeto dispositivo ZigBee (ZDO – *ZigBee Device Object*) y los objetos de aplicación definidos en la manufacturación. Las aplicaciones se implementan a través del ZDO.

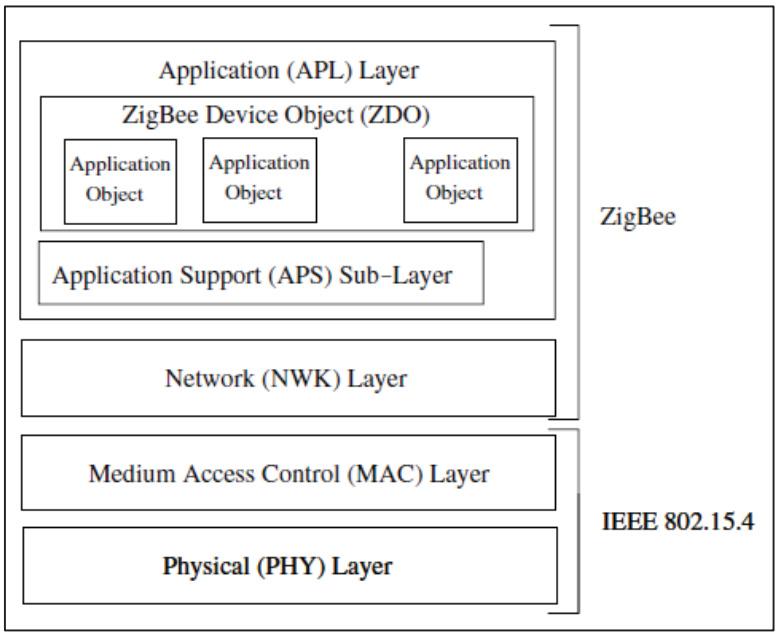


Figura 1-9 Pila de protocolos ZigBee y IEEE 802.15.4 [26]

El ZDO define funciones provistas por el dispositivo para la operación de red. Específicamente, el rol de dispositivos tales como el coordinador de red o el enrutador es definido a través del ZDO.

Por otra parte, cuando un dispositivo necesite ser asociarse, los requerimientos se manejan a través del ZDO. Finalmente, la subcapa APS provee capacidad de descubrimiento a los dispositivos de modo que mantiene una lista o tabla de sus vecinos y las funcionalidades provistas por estos.

Esta información también es usada para emparejar los requerimientos de asociación de los vecinos con funciones específicas.

1.1.4.2.3 6LoWPAN

Según Antonio Manuel Ortiz [25] se especifica 6LoWPAN (*IPv6 over Low power Wireless Personal Area Networks* – IPv6 sobre redes inalámbricas de área personal de baja potencia) como una arquitectura que define el uso del direccionamiento IPv6 para las redes de sensores. Permitiendo así su inclusión en la red global, favoreciendo el acceso a los nodos desde cualquier lugar.

Para la definición de la capa física y de acceso al medio, utiliza al igual que ZigBee, el estándar IEEE 802.15.4. Mientras que, en la capa de red, utiliza direccionamiento IPv6 adaptado a estas redes mediante la capa LoWPAN, la cual proporciona el encapsulamiento y los métodos necesarios para permitir la coexistencia de 802.15.4 e IPv6.

La capa de transporte puede utilizar UDP[26] (*User Datagram Protocol* – Protocolo de Datagrama de Usuario) o ICMP[27] (*Internet Control Message Protocol* – Protocolo de Mensajes de Control de Internet), dependiendo de los requerimientos de cada aplicación.

La Figura 1-10 muestra los componentes principales de esta arquitectura.

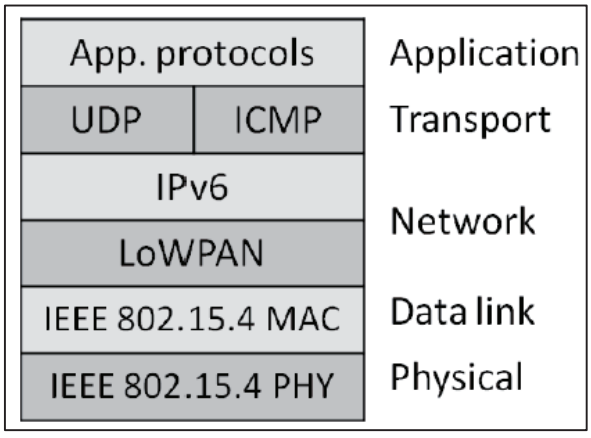


Figura 1-10 Arquitectura 6LoWPAN [25]

[26] Protocolo que sin establecer previamente una conexión puede enviar datagramas a través de la red, gracias a la información que tiene en su cabecera.
[27] Protocolo que trabaja en conjunto con el protocolo IP, para auxiliarle en el control y notificación de errores.

1.1.4.3 Topologías en las redes inalámbricas de sensores

En [25] se enuncia que la estructuración lógica de la red WSN suele ser llevada a cabo generalmente por protocolos de red y acceso al medio. Al carecer de infraestructuras físicas, las redes de sensores pueden adoptar la estructura que más convenga en cada aplicación. A continuación, se muestran las topologías más comunes en redes de sensores:

1.1.4.3.1 Topología en estrella

Son sistemas de un solo salto en el que todos los nodos pueden comunicarse de forma directa con la estación base. Una analogía al mundo de las redes en general podría ser la arquitectura cliente-servidor de las redes Ethernet [25].

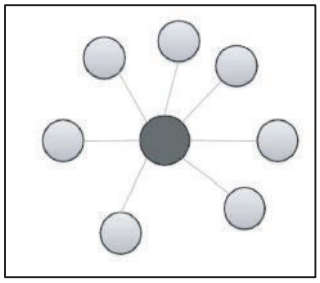


Figura 1-11 Topología estrella [25]

Como se puede observar en la Figura 1-11, la estación base (nodo fuente) se encuentra resaltado y puede comunicarse en un solo salto a cualquier nodo de la WSN.

1.1.4.3.2 Topología en árbol

Pueden ser consideradas como una generalización de las redes en estrella. Los caminos definidos en estas redes van desde cualquier nodo hasta la estación base, pudiendo atravesar varios nodos intermedios hasta llegar a ella [25].

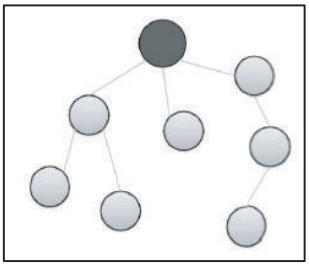


Figura 1-12 Topología en árbol [25]

En la Figura 1-12, se puede observar como todas las rutas llevan al nodo resaltado (nodo fuente).

1.1.4.3.3 Topología en malla

Estas redes permiten a los datos saltar de nodo en nodo. De esta forma, un nodo es capaz de comunicarse con cualquier otro, de modo que los datos pueden atravesar nodos intermedios hasta llegar a su destino [25]. La Figura 1-13 muestra esta topología

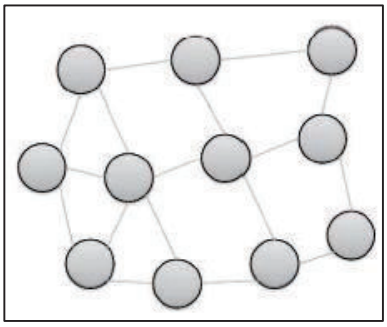


Figura 1-13 Topología en malla [25]

1.1.4.3.4 Topologías basadas en clúster

Esta clase de topología separa los nodos en grupos gobernados por una cabeza de clúster. Los cabezas de clúster pueden comunicarse entre sí, de forma que un nodo que quiera enviar datos a otro, se los enviará a su cabeza, o a una cabeza del clúster vecino que conozca la ruta hacia el destino [25].

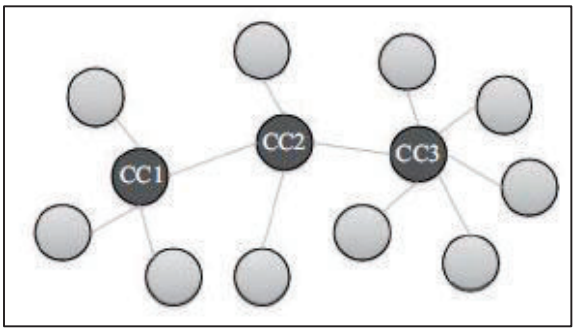


Figura 1-14 Topología basada en clúster [25]

En la Figura 1-14 se puede observar las cabezas de clúster resaltadas y nombradas como CC1, CC2 y CC3. El resto de nodos puede comunicarse a través de las cabezas hasta cualquier nodo de la WSN.

1.2 INTRODUCCIÓN A LA TEORÍA DE GRAFOS

1.2.1 DEFINICIONES FUNDAMENTALES DE LA TEORÍA DE GRAFOS

Según Vitaly Voloshin, profesor del Departamento de Matemáticas de la Universidad Troy [27] un grafo, explicado de forma sencilla, es un conjunto de puntos y líneas que conectan algunos pares de puntos.

En teoría de grafos, los puntos son llamados vértices y las líneas son llamadas aristas (*edges*). De tal manera que, la Figura 1-15 consta de cinco vértices y siete aristas.

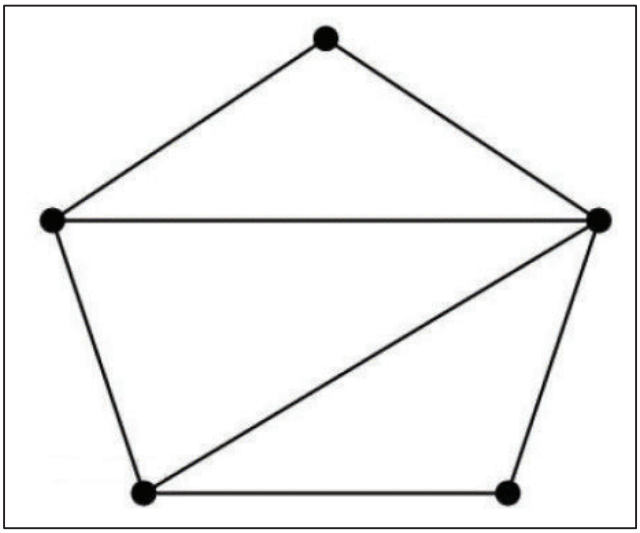


Figura 1-15 Ejemplo de un grafo [27]

Por otro lado, en [28] se afirma que por lo general los nodos o vértices son entes de procesamiento o estructuras que contienen algún tipo de información. Las aristas en cambio representan conexiones o relaciones entre los nodos.

Según Voloshin [27] la notación estándar es la siguiente: letras mayúsculas para los conjuntos (todos los conjuntos son finitos), letras minúsculas para los elementos del conjunto, llaves para listar los elementos de un conjunto. El número de elementos del conjunto A se denota como $|A|$ (i.e: $|\cdot|$ es la cardinalidad del conjunto), y el conjunto vacío como $\emptyset$. Si un conjunto contiene otros conjuntos como elementos, entonces se le llama familia.

En un grafo, el conjunto de los vértices podría ser denotado como X y se escribe como $X = \{x_1, x_2, \dots, x_n\}$ donde x_i es el vértice i-ésimo y n es el número de vértices.

El conjunto de aristas se denota por la letra E donde $E = \{e_1, e_2, \dots e_m\}$ donde e_i es la arista i-ésima y m es el número de aristas. Cada arista e_i es identificada por el par de vértices respectivos los cuales están conectados por e_i.

Matemáticamente se define un grafo G, como un conjunto de vértices X, y un conjunto de aristas E. Como resultado, se describe un grafo con la siguiente fórmula $G = (X, E)$, por lo que el grafo de la Figura 1-15 se podría representar como se observa en la Figura 1-16:

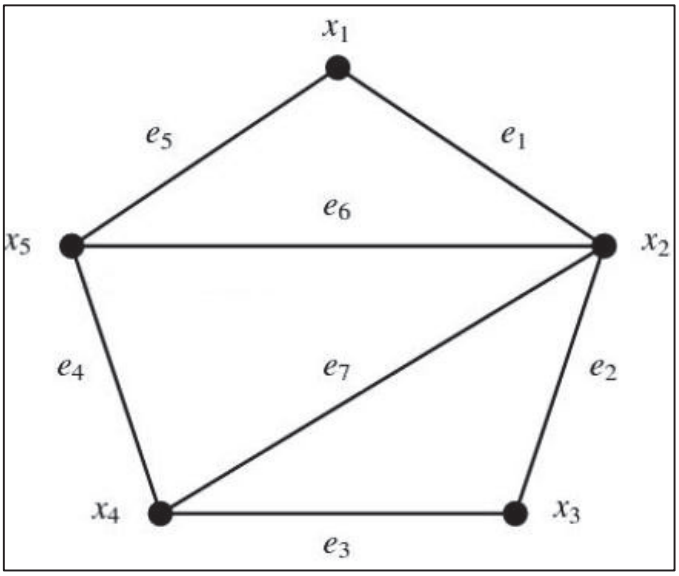


Figura 1-16 Representación gráfica del grafo G = (X, E). [27]

1.2.1.1 Conceptos básicos de la Teoría de Grafos

Los siguientes conceptos fueron tomados de [28] [29] y [30].

1.2.1.1.1 Digrafo o grafo dirigido

Es aquel grafo que utiliza flechas para conectar los nodos, debido a que las aristas tienen una dirección [28]. Es a causa de esta dirección que este tipo de grafos reciben su nombre, además para el presente proyecto se asume con ayuda de Castalia que las aristas de ida poseen la misma calidad que las de regreso.

1.2.1.1.2 Grafo no dirigido

Corresponde a lo contrario de un dígrafo, es decir, las aristas no tienen dirección y se utiliza líneas para la representación de las aristas en este tipo de grafos [28].

1.2.1.1.3 Grafo pesado

Frecuentemente, las aristas tienen algún tipo de información asociada (distancia, costo, confiabilidad, etc), y como resultado se genera un grafo pesado [28].

1.2.1.1.4 Camino

Para Voloshin [29], un camino es un grafo en el cuál todos los vértices pueden ser ordenados (ordenados de izquierda a derecha) $X = \{x_1, x_2, \dots, x_5\}$.

De tal manera que exista precisamente una arista conectando a cada dos vértices consecutivos y no exista otras aristas, tal como se muestra en la Figura 1-17.

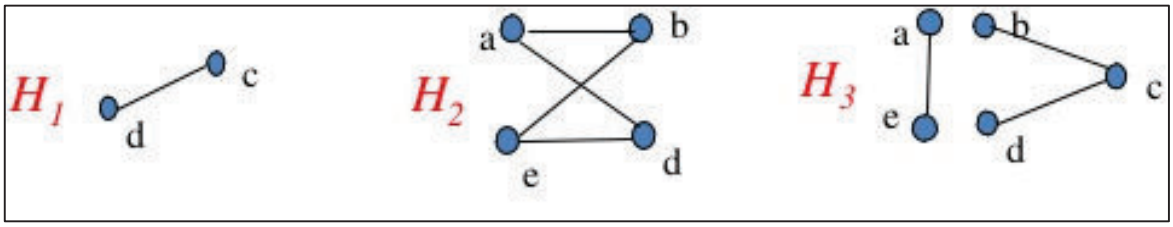


Figura 1-17 Los grafos H_1 y H_2 son conexos, el grafo H_3 no es conexo [31]

1.2.1.1.4.1 Longitud del camino

Corresponde al número de aristas en un camino [29].

1.2.1.1.5 Ciclo y Tour

Un ciclo es un camino que termina en el mismo nodo donde comenzó. Si el camino recorre todos los nodos se llama tour [29].

1.2.1.1.6 Grafo conexo

Se dice que un grafo es conexo cuando se puede llegar desde cualquier nodo hasta cualquier otro nodo mediante un camino.

Si el caso anterior no se cumple, entonces se concluye que el grafo no es conexo, pero puede dividirse en componentes conexas [28]. El concepto de grafo conexo suele aplicarse generalmente a grafos no dirigidos [31].

1.2.1.1.7 Componentes conexas

Son subconjuntos de nodos y aristas del grafo original que si son conexos. Por ejemplo, en la Figura 1-18, para el grafo H_3, a-e y b-c-d son componentes conexas [28].

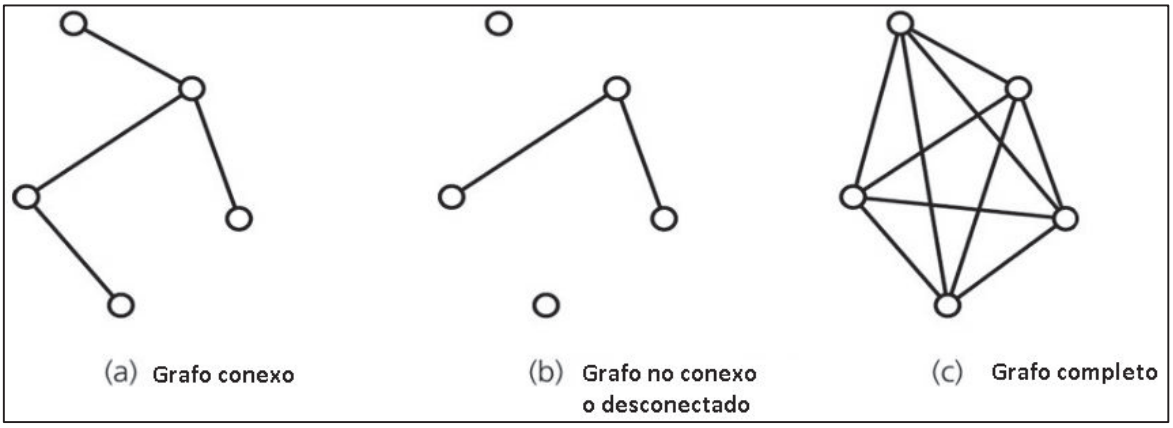


Figura 1-18 a) Grafo conexo, b) Grafo no conexo y c) Grafo completo [32]

1.2.1.1.8 Grafo completo

Suele aplicarse a grafos no dirigidos, en los que existe una arista para cada par de nodos [31]. Si el grafo no es completo, al utilizar costos de enlace exageradamente altos en enlaces inexistentes, se puede convertir en teoría cualquier grafo a un grafo completo.

1.2.1.1.9 Árbol

Corresponde a un grafo conexo sin ciclos (Véase 1.2.1.1.5) [30].

Como se puede observar en la Figura 1-19, inclusive un conjunto de nodos interconectados en línea recta son considerados un árbol.

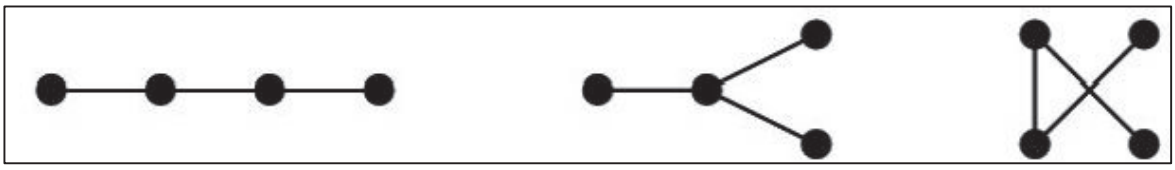


Figura 1-19 Ejemplos de grafos tipo árbol [30]

Adicionalmente, de [27] se extraen las siguientes definiciones que son útiles en el estudio de la teoría de grafos:

1.2.1.1.10 Vértices Adyacentes y Disjuntos

Se señala que los vértices son adyacentes cuando dos vértices están conectados por una arista, caso contrario se los llama vértices disjuntos.

1.2.1.1.11 Grado del vértice

Dado un vértice x, el número de todos los vértices adyacentes a él es llamado el grado del vértice x y se lo denota como $d(x)$. Por ejemplo, para la Figura 1-16, $d(x_1) = 2$ y $d(x_2)$=4.

1.2.1.1.12 Máximo grado del grafo

Consiste en el máximo grado de entre todos los vértices del grafo y se lo denota como $\Delta(G)$. Para la Figura 1-16 $\Delta(G) = \max\ d(x_i) = d(x_2) = 4$, donde $i = 1, \dots n$. En un *tour*, siempre el máximo grado de su grafo correspondiente es igual a dos.

1.2.2 PROBLEMAS PRINCIPALES DE PROCESAMIENTO EN LA TEORÍA DE GRAFOS

De acuerdo a Ernesto Coto en [33], existe una gran variedad de problemas relacionados con grafos y variedad de algoritmos para procesamiento de grafos.

Sin embargo, no todo problema de grafos es sencillo de resolver y en muchas ocasiones tampoco es sencillo determinar qué tan difícil puede ser resuelto. Por tal motivo, el autor hace una clasificación según la dificultad y cita algunos de los problemas más representativos de la teoría de grafos como se puede ver a continuación en 1.2.2.1 y 1.2.2.2.

1.2.2.1 Clasificación de los problemas de grafos según su dificultad

De acuerdo al grado de dificultad para resolver los problemas de grafos, se los puede clasificar en:

1.2.2.1.1 Fáciles

Un problema fácil de procesamiento de grafos es aquel que se puede resolver utilizando un programa eficiente. Frecuentemente, su tiempo de ejecución es lineal en el peor caso, o limitado por un polinomio de bajo grado en el número de nodos o el número de aristas.

Generalmente, también se puede decir que el problema es fácil si podemos desarrollar algoritmos de fuerza bruta que, aunque sean lentos para grandes grafos, es útil para grafos pequeños e inclusive de tamaño medio.

Entonces, una vez que se sabe que el problema es fácil, se buscan soluciones eficientes para escoger la mejor de ellas [33].

1.2.2.1.2 Tratables

Un problema tratable de procesamiento de grafos ocurre cuando se conoce un algoritmo que garantiza que sus requerimientos en tiempo y espacio están limitados por una función polinomial en el tamaño del grafo (número de nodos + número de aristas).

Todo problema fácil es tratable, pero existen distinciones debido a que el desarrollo de una solución eficiente es extremadamente difícil o imposible. Las soluciones a algunos problemas intratables nunca han sido escritas en programas, debido a sus tiempos de ejecución tan altos que no puede contemplarse su utilización en la práctica [33].

1.2.2.1.3 Intratables

Un problema intratable de procesamiento de grafos corresponde aquel que no se conoce algún algoritmo que garantice obtener una solución del problema en una cantidad razonable de tiempo.

Muchos de estos problemas tienen la característica de que podemos utilizar un método de fuerza bruta para probar todas las posibilidades de calcular la solución, y se consideran intratables porque existen demasiadas posibilidades a considerar.

Esta clase de problemas es extensa y muchos expertos piensan que no existen algoritmos eficientes para solucionar estos problemas. El término NP-duro[28] describe los problemas de esta clase, el cual representa un altísimo nivel de dificultad [33].

---

[28] Tipo especial de problemas computacionales que **N**o se pueden resolver en un tiempo razonable (**P**olinomial).

Para Martín Martínez-Rangel en [34], la Teoría de la complejidad computacional forma parte de la teoría de la computación que estudia los recursos requeridos durante el cálculo para resolver un problema.

Los recursos comúnmente estudiados son: el tiempo (número de pasos de ejecución de un algoritmo para resolver un problema) y el espacio (cantidad de memoria utilizada para resolver un problema).

En esta área de trabajo, existe una clasificación de tipos de problemas que se abordan dentro de la teoría de la complejidad. El primer grupo de problemas son aquellos agrupados en el conjunto “P”, cuyos algoritmos que los resuelven son definidos por funciones polinomiales (el tiempo de ejecución está en función del número valores en la entrada), un segundo grupo es aquel denominado como “NP”, problemas de decisión que tienen un algoritmo de tipo polinomial en una máquina no determinista que los resuelva.

El tercer grupo es denominado como NP-Completo, y son aquellos problemas NP más difíciles de resolver, y por último se encuentra el grupo de NP-Duros o problemas intratables, debido al espacio de soluciones que se tiene a medida que crece el número de instancias involucradas en cada uno de los problemas.

Para darnos cuenta de la intratabilidad de este tipo de problemas, se puede señalar que un problema de 20 máquinas x 10 tareas tiene 7.261x10183 posibles soluciones, lo que en principio supera la edad del universo dado en microsegundos, más aun, muchas de estas soluciones pueden ser no factibles debido a las restricciones de precedencia y disyuntivas que se tienen que considerar. La clase NP-Completo puede definirse alternativamente como la intersección entre NP y NP-Duro.

1.2.2.1.4 De dificultad desconocida

Existen problemas de procesamiento de grafos cuya dificultad es desconocida. No existe un algoritmo eficiente conocido para resolverlos, ni son conocidos como NP-duro. El problema de isomorfismo de grafos (Véase 1.2.2.11) pertenece a esta clase de problemas [33].

1.2.2.2 Problemas de procesamiento de grafos

Así mismo Coto en [33] enuncia algunos de los problemas más conocidos de procesamiento de grafos, los cuales se mencionan a continuación:

1.2.2.2.1 Conectividad Simple

Consiste en estudiar si el grafo es conexo, es decir, si existe al menos un camino entre cada par de vértices.

1.2.2.2.2 Detección de Ciclos

Consiste en estudiar la existencia de al menos un ciclo en el grafo.

1.2.2.2.3 Camino Simple

Consiste en estudiar la existencia de un camino entre dos vértices cualesquiera.

1.2.2.2.4 Camino y Tour de Euler

Consiste en estudiar la existencia de un camino que conecte dos vértices dados, usando cada arista del grafo exactamente una sola vez. Si el camino tiene como inicio y final el mismo vértice, entonces se desea encontrar un tour de Euler.

1.2.2.2.5 Camino y Tour de Hamilton

Consiste en estudiar la existencia de un camino que conecte dos vértices dados que, visite cada nodo del grafo exactamente una vez. Si el camino tiene como inicio y final el mismo vértice, entonces se desea encontrar un tour de Hamilton.

1.2.2.2.6 Árbol de Expansión Mínima

Consiste en encontrar, en un grafo pesado, el conjunto de aristas de peso mínimo que conecta a todos los vértices.

1.2.2.2.7 Caminos cortos a partir de un mismo origen

Consiste en encontrar cuales son los caminos más cortos conectando a un vértice x_i cualquiera, con cada uno de los otros vértices de un dígrafo pesado.

Este es un problema que por lo general se presenta en redes de computadores, representadas como grafos. De hecho, para los propósitos del presente proyecto se ha representado la WSN como un grafo.

1.2.2.2.8 Pareamiento (Matching)

Dado un grafo, consiste en encontrar cual es el subconjunto más largo de sus aristas con la propiedad de que no existan dos aristas conectadas al mismo vértice.

Se sabe que este problema clásico es resoluble en tiempo proporcional a una función polinomial en el número de vértices y de aristas (es decir, es NP-duro), pero aún no existe un algoritmo rápido que se ajuste a grandes grafos.

1.2.2.2.9 Asignación o pareamiento bipartito (bipartite weigthed matching)

Consiste en encontrar un pareamiento perfecto de peso mínimo en un grafo bipartito. Un grafo bipartito es aquel cuyos vértices se pueden separar en dos conjuntos, de tal manera que todas las aristas conecten a un vértice en un conjunto con otro vértice en el otro conjunto.

1.2.2.2.10 El camino más largo

Consiste en encontrar cual es el camino más largo que conecte a dos nodos dados en el grafo. Aunque parece sencillo, este problema es una versión del problema del tour de Hamilton y es NP-duro.

1.2.2.2.11 Isomorfismo de grafos

Consiste en estudiar la posibilidad de hacer dos grafos idénticos con solo renombrar sus nodos. Se conocen algoritmos eficientes para solucionar este problema, para varias clases particulares de grafos, pero no se tiene solución para el problema general (NP-duro).

1.3 EL PROBLEMA DEL AGENTE VIAJERO (TRAVELING SALESMAN PROBLEM)

Según Erenesto Coto [35], encontrar un tour hamiltoniano para un grafo cualquiera es un problema intratable, esto se debe a que la única solución conocida para el caso general del problema consiste en buscar la solución entre todas las posibilidades de ciclos en el grafo. Como la gran mayoría de los problemas intratables, el problema de encontrar un tour hamiltoniano es un problema NP-duro.

Otros problemas que son variaciones del tour hamiltoniano, heredan el mismo nivel de dificultad. Por ejemplo, uno de los problemas más famosos de procesamiento de grafos que está íntimamente relacionado con el tour hamiltoniano es el Problema del Agente Viajero.

El problema del TSP consiste en encontrar el tour de costo mínimo que pase por todos los nodos del grafo, donde los nodos por lo general representan ciudades y los pesos de las aristas son distancias o costos de viaje de una ciudad a otra. Es evidente su relación con el problema del tour hamiltoniano, y si no es posible encontrar una solución eficiente para éste, mucho menos para el TSP.

En la Figura 1-20 se muestra un grafo de ejemplo y su TSP resuelto a la derecha.

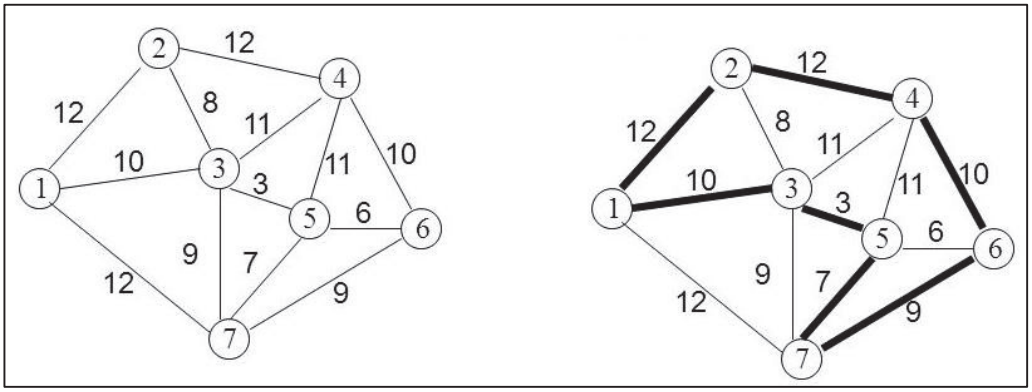


Figura 1-20 A la izquierda se puede apreciar un grafo no dirigido y su TSP resuelto resaltado en negrilla a la derecha [36].

Se ha intentado atacar el problema de TSP utilizando otras técnicas. Por ejemplo, se plantea una posible solución utilizando algoritmos de búsqueda local, pero solo funciona para acelerar el algoritmo en casos particulares del problema. También se pueden utilizar algoritmos genéticos para obtener una aproximación al costo mínimo del TSP.

Su definición matemática la extraemos de [37]: Sea $G = (X, E)$ un grafo (dirigido o no dirigido) y F la familia de todos los tours hamiltonianos en G. Para cada arista $e \in E$, un costo (peso) c_{e_i} es prescrito. Entonces el TSP consiste en encontrar un tour hamiltoniano en G de tal manera que la suma de los costos de las aristas del tour es la más pequeña posible.

Sin perder generalidad, se puede asumir que G es un grafo completo (esta es una condición para que el TSP sea resoluble en G), de otra manera se pueden reemplazar las aristas faltantes con aristas de un costo muy grande. Sea el conjunto de los vértices $V = \{e_1, e_2, \dots e_m\}$ y la matriz $C = (c_{ij})_{nn}$ (llamada matriz de costos, matriz de distancias, matriz de pesos o matriz de adyacencias), entonces la entrada ij-ésima c_{ij} corresponde al costo de la arista que une el vértice i y el vértice j en G. Esto se pude entender mejor con los gráficos de la Figura 1-21.

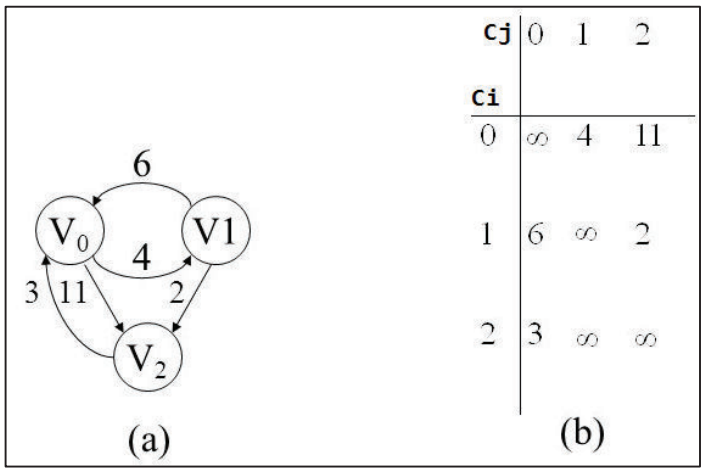


Figura 1-21 (a) Digrafo G. (b) Matriz de costos para G, convirtiendo a G en un digrafo completo [38].

Además, dependiendo de la naturaleza de la matriz de costos (equivalentemente a la naturaleza de G), el TSP es dividido en dos clases. Si C es simétrica (esto es, G es no dirigido) entonces el TSP es llamado el problema del agente viajero simétrico (STSP – *Symmetric Traveling Salesman Problem*). Si C no es necesariamente simétrico (equivalentemente a que el grafo G es dirigido) entonces este es llamado el problema del agente viajero asimétrico (ATSP – *Asymmetric Traveling Salesman Problem*)

De acuerdo a [5], el enrutamiento del camino más corto es conveniente para la eficiencia de energía promedio en una WSN, ya que la energía necesitada para transmitir un paquete está correlacionada con la longitud del camino. Sin embargo, esto puede cargar demasiado a algunos nodos, causando así caídas prematuras y creando hoyos en la red, los cuales en el peor de los casos pueden dejar a la red desconectada.

En [5] también se da a conocer que el problema para calcular la mayoría de las rutas óptimas en una WSN es un problema del tipo NP-duro, de tal forma que para el presente proyecto al plantearse el uso de 18 nodos sensores, según [39] se tendría una complejidad de (18-1)! = 17! = 355687428096000 iteraciones, usando un algoritmo de fuerza bruta (es decir, evaluando cada una de las posibles rutas hasta obtener la ruta óptima).

Como resultado, si se usa un método de resolución por fuerza bruta, el problema se volvería irresoluble, y por lo tanto se busca aplicar métodos alternativos de los cuales se tratará en el siguiente capítulo.

2 CAPÍTULO 2: ANÁLISIS COMPARATIVO DE LOS ALGORITMOS PARA RESOLVER EL TSP: MST Y BRANCH AND BOUND

De acuerdo a Tunc Ikikardes [40], en una WSN los nodos deben organizarse a sí mismos automáticamente, y como resultado son necesarios algoritmos que puedan implementarse localmente.

Además, debido al bajo consumo de energía los algoritmos que trabajan estáticamente o aquellos que necesitan menos re – enrutamiento, son mejor apreciados.

Por tal motivo, se ha escogido dos algoritmos para la resolución del TSP en una WSN, donde *Branch and Bound* (B&B) corresponde a un algoritmo "exacto" y la resolución mediante el uso del *Minimum Spanning Tree* (MST) a un algoritmo "aproximado" (heurístico).

El algoritmo de B&B podría implementarse localmente en el nodo fuente o en el servidor al que se conecta la WSN, mientras que el algoritmo por MST debido a su baja complejidad (véase el apartado 2.1 y sus subtemas) puede ser implementado en uno o varios nodos según la aplicación de la WSN.

2.1 MINIMUM SPANNING TREE Y EL TRAVELLING SALESMAN PROBLEM

En la catedra de *Network Design and Optimization* en impartida en la Universidad de Melbourne [41], uno de los métodos heurísticos expuestos para resolver el TSP, se fundamenta en el Árbol de Expansión Mínima (MST – *Minimum Spanning Tree*).

En un artículo publicado por la Universidad de Buenos Aires [42] se define el término heurística en ciencias computacionales como un tipo de algoritmo que ignorando cierta información, se libra de gran parte del esfuerzo que debió haberse requerido para leer los datos y hacer cálculos con ellos. Por otra parte, la solución producida por tal heurística, es independiente de la información ignorada, y de este modo no se

ve afectada por cambios en tal información. Idealmente, ignora información que resulta muy caro colectar o mantener o que contribuye en menor grado a la precisión de la solución.

Al ser el TSP un problema del tipo NP, uno de los flancos por los que se ataca la resolución del mismo consiste en un conjunto de técnicas heurísticas, de las cuales se toma la resolución del MST como una de ellas.

2.1.1 INTRODUCCIÓN AL MINIMUM SPANNING TREE

En [43], se establece que si el grafo no es dirigido y se desea encontrar la manera menos costosa de conectar todos los puntos, entonces aparece el problema de encontrar un árbol de expansión mínima.

Un árbol de expansión de un grafo conexo consiste en un subgrafo que contiene todos los nodos del grafo y no tiene ciclos. El árbol de expansión mínima de un grafo pesado no dirigido, es el árbol de expansión cuyo peso (la suma de los pesos de todas sus aristas) no es mayor al de ningún otro árbol de expansión.

De hecho, el problema de encontrar el MST de un grafo pesado no dirigido arbitrario tiene una gran cantidad de aplicaciones importantes y se dispone (desde 1920) de algoritmos para encontrarlo; sin embargo, la eficiencia de las implementaciones varía ampliamente y los investigadores aún buscan mejores métodos.

2.1.1.1 Algoritmo de Prim

De acuerdo a Ikikardes [44], este algoritmo fue desarrollado para grafos no-dirigidos y pesados, con un solo nodo fuente, y fue descubierto por Robert Prim en 1957.

El algoritmo inicia desde un nodo fuente, el cual contiene en sí mismo un árbol de expansión parcial inicial, y en cada paso el algoritmo inserta el nodo que tiene el enlace de mínimo costo al árbol de expansión parcial actual.

Es así que, el árbol de expansión parcial crece hasta que todos los nodos son añadidos y el árbol de expansión mínima es obtenido. El algoritmo es intuitivo y no se lo considera un método sofisticado para obtener el árbol de expansión mínima.

2.1.1.1.1 Complejidad

El algoritmo de Prim encuentra el MST en su implementación básica en un tiempo de $O_{(mn)}$ (expresado utilizando la notación de la gran O)[29], donde "n" es el número de nodos en la WSN y $O_{(m)}$ es la complejidad en tiempo de cada iteración (siendo "m" el número de iteraciones).

Sin embargo usando estructuras de datos como pilas binarias o de Fibonacci, la complejidad en tiempo puede ser reducida a $O_{(m \log n)}$ o a su vez $O_{(m+n \log n)}$ respectivamente [44].

2.1.1.1.2 Pseudocódigo

2.1.1.1.2.1 Algoritmo de Prim en su implementación básica:

Se traduce el pseudocódigo en [45]:

```
Sea T un vértice x
Sea G el grafo
Mientras (T posea menos de n vértices)
{
        Encontrar la arista más pequeña que conecte T a (G excluyendo T)
        Añadir la arista a T
}
```

T en el pseudocódigo expuesto, representa una estructura de datos del tipo arreglo, o inclusive podría ser un arraylist que guarde los valores de las instancias de la clase vértice, en el caso de que se use un lenguaje orientado a objetos.

Entre las ventajas que se obtiene al utilizar un lenguaje orientado a objetos, cabe destacar que la longitud del enlace (o costo, como se denomina a la calidad del enlace en el presente proyecto), puede ser almacenada como atributo de la clase.

---

[29] La notación $O_{(f(n))}$ (O en función de n) denota la cantidad de operaciones básicas del algoritmo analizado. Por ejemplo, un algoritmo de ordenamiento conocido como el método de la burbuja requiere de n^2 operaciones para ordenar una lista de n números, por lo tanto f(n)= n^2 y la notación de la gran O para el algoritmo es O(n^2) [84]

2.1.1.1.2.1.1 Ejemplo

Un ejemplo del proceso del algoritmo de Prim (básico) consta en la Figura 2-1:

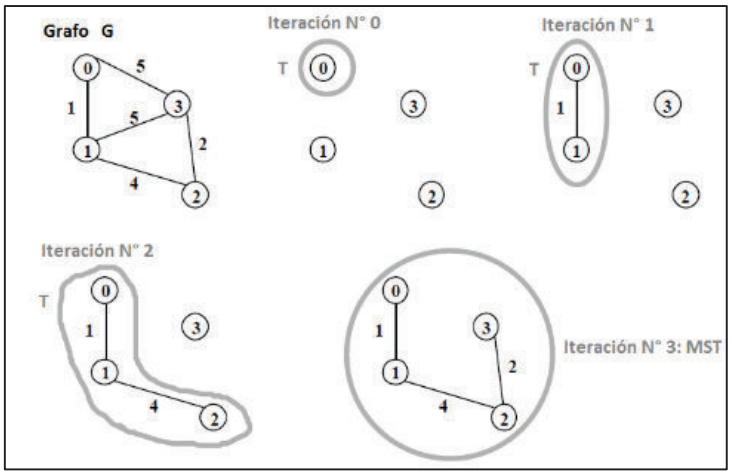


Figura 2-1 Proceso de obtención del MST utilizando el algoritmo de Prim en su implementación básica [43].

2.1.1.1.2.2 Algoritmo de Prim usando pilas[30]:

Se traduce el pseudocódigo en [45]:

{Hacer una pila de valores (vértice, arista, peso (arista))
    Inicializar la pila (v, -,infinito) para cada vértice
Sea T un árbol vacío
Mientras (T tenga menos de n vértices)
{Permitir que (v, e, peso(e)) tenga el menor peso almacenado en la pila
    Remover (v, e, peso(e)) de la pila
    Añadir v y e a T
    Para cada arista f=(u,v)
    Si u aún no está en T
    {   Encontrar valor(u, g, peso(g)) en la pila
        Si peso(f) < peso (g)
        {Reemplazar (u, g, peso(g)) con (u,f,peso(f))
        } } }

[30] Traducido del inglés *heap*

2.1.1.1.2.2.1 Ejemplo:

Un ejemplo del proceso del algoritmo de Prim (con pilas) consta en la Figura 2-2.

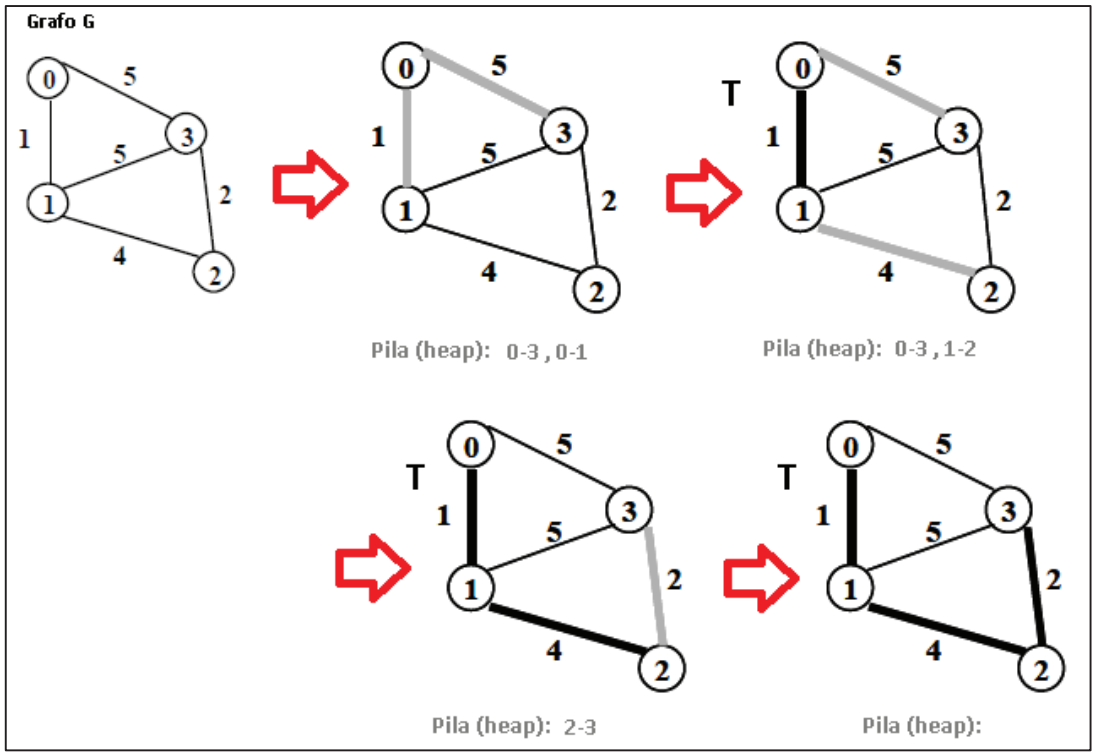


Figura 2-2 Proceso de obtención del MST utilizando el algoritmo de Prim en su implementación usando pilas (heap) [43].

2.1.1.2 Algoritmo de Borůvka

Según la investigación realizada en [44], este es el algoritmo más antiguo para encontrar el MST, inventado por el matemático checo Otokar Borůvka en 1926.

El algoritmo comienza en su etapa inicial, con todos los nodos que no tienen ninguna conexión a otro(s), en la segunda etapa del algoritmo, cada nodo selecciona la mejor arista de su vecindario.

Los nodos resultantes de la unión descrita son denominados supernodos, y en futuras iteraciones cada supernodo escoge su arista de menor peso hacia otro supernodo, de tal manera que ambos supernodos quedan unidos por esta arista.

En la última iteración solo un supernodo prevalece, siendo este el árbol de expansión mínima para el grafo inicial.

2.1.1.2.1 Complejidad

El algoritmo encuentra el árbol de expansión mínima en un tiempo dado por $O_{(m \log n)}$, donde “n” es el número de nodos de la WSN.

El lazo para encontrar los supernodos y unirlos se repite al menos $O_{(\log n)}$ veces, así como a cada iteración el número de aristas restantes se reducen en un factor de al menos dos y $O_{(m)}$ es la complejidad en tiempo de cada iteración [44].

2.1.1.2.2 Pseudocódigo

Se traduce el pseudocódigo en [45]:

{Sea G el grafo
Hacer una lista L de n árboles, cada uno de un solo vértice.
Mientras (L tenga más de un árbol)
{Para cada T en L, encontrar la arista más pequeña que conecte T a (G-T)
Añadir todas esas aristas al MST
(Causando pares de árboles en L para unirlos)
}}

2.1.1.2.2.1 Ejemplo

En Figura 2-3, Figura 2-4 y Figura 2-5 se muestra el proceso del algoritmo de Borůvka.

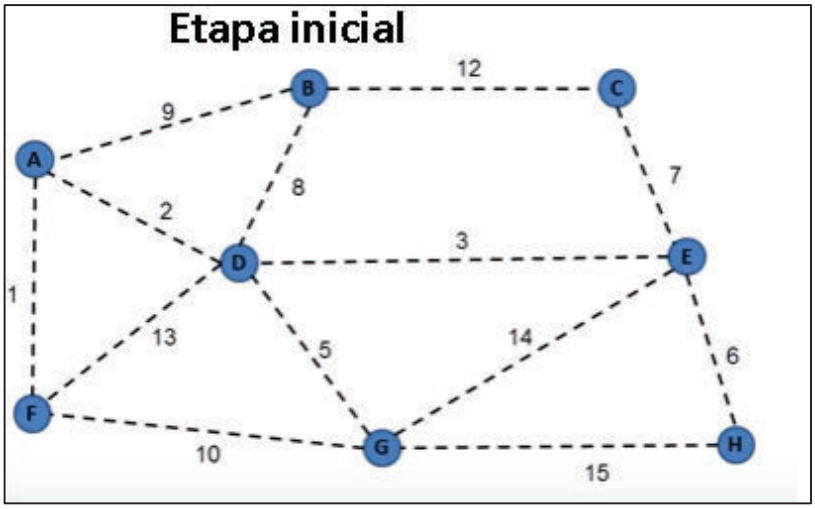


Figura 2-3 Proceso de obtención del MST utilizando el algoritmo de Borůvka en su etapa inicial [46]

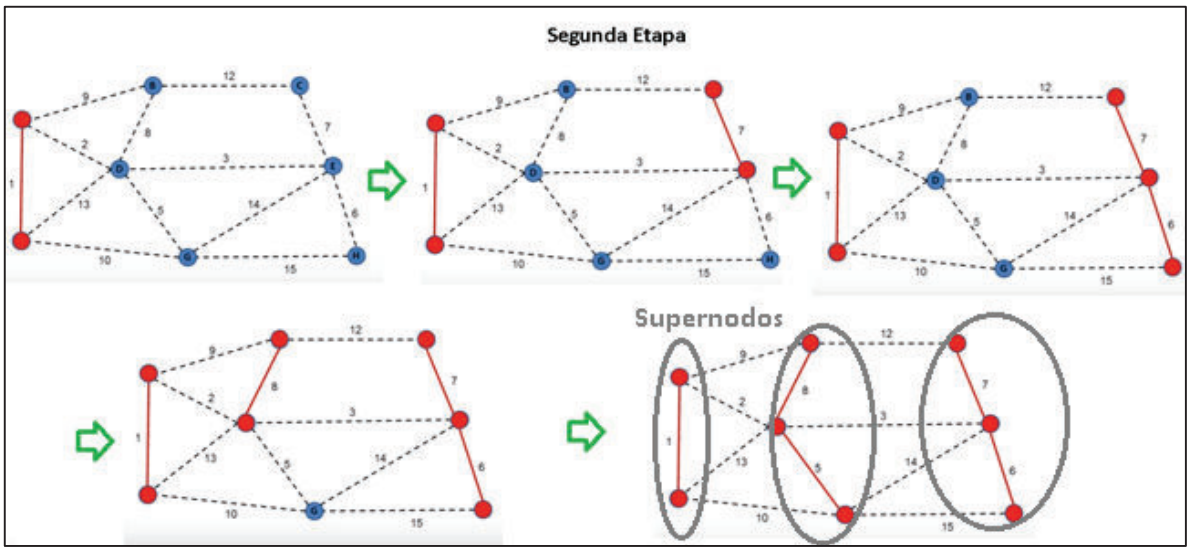


Figura 2-4 Proceso de obtención del MST utilizando el algoritmo de Borůvka en su segunda etapa: obtención de los supernodos [46].

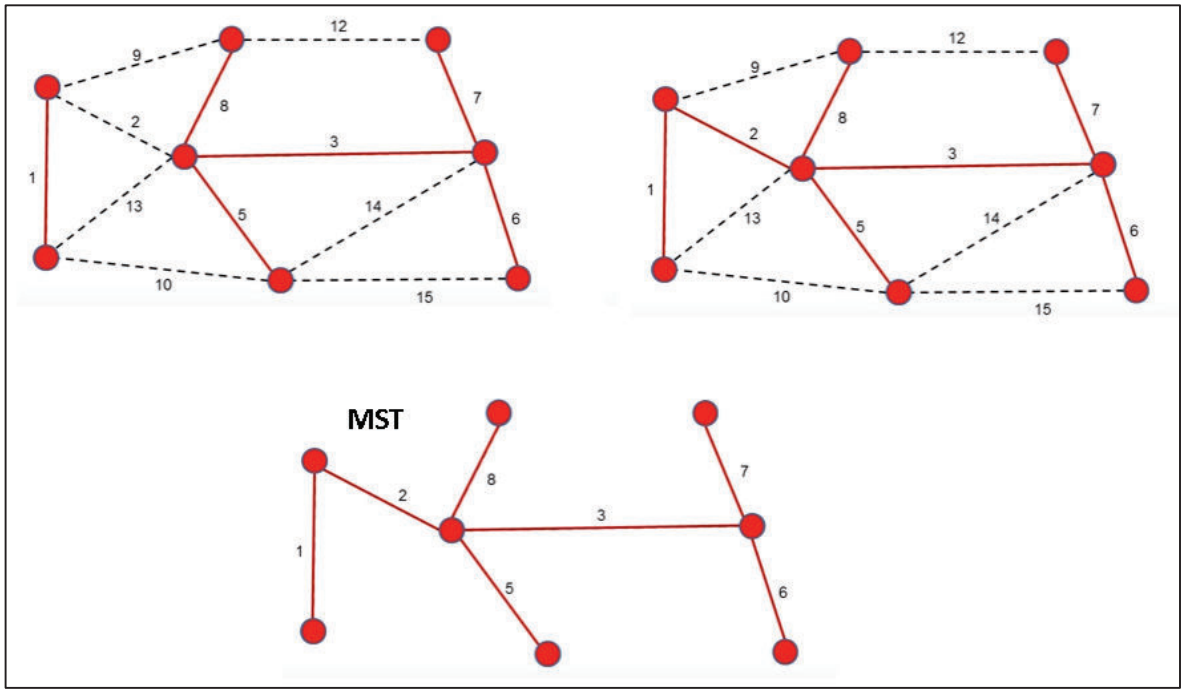


Figura 2-5 Proceso de obtención del MST utilizando el algoritmo de Borůvka en su última etapa: unión de los supernodos [46].

2.1.1.3 Algoritmo de Kruskal

Este algoritmo fue desarrollado por el matemático Joseph Kruskal en 1956, y puede ser aplicado en un grafo pesado no dirigido. Se interpreta el algoritmo como una variación de Prim enfocado en las aristas.

El algoritmo inicia con un subgrafo del grafo original (G), este subgrafo es inicialmente un bosque que contiene todos los nodos sueltos sin aristas y solo añade aquellas que unan dos diferentes árboles parciales, hasta que todos los nodos son unidos y el subgrafo forma un árbol de expansión que es el MST [44].

2.1.1.3.1 Complejidad

De acuerdo a Eppstein [45], se presume que el trabajo para saber si dos vértices finales están desconectados, debería ser lenta (un tiempo lineal por iteración, o un tiempo total $O_{(mn)}$.

Pero en realidad se puede hacer uso de estructuras de datos más complicadas que permiten realizar cada prueba del algoritmo en un tiempo casi constante, esta estructura se conoce como *union-find*. La parte más lenta del algoritmo es la del ordenamiento, que toma un tiempo de $O_{(m \log n)}$.

2.1.1.3.2 Pseudocódigo

Se traduce el pseudocódigo en [45]:

{Sea G el grafo
Ordenar las aristas de G en orden creciente según su peso
Mantener un subgrafo S de G, inicialmente vacío
Para cada arista e ordenada
{Si los extremos de e están desconectados en S
    { Añadir e a S
    }
    Retornar S }

Cuando las tareas aumentan en complejidad, como en la segunda línea del pseudocódigo donde se indica es necesario ordenar las aristas del grafo en orden creciente, las tareas se facilitarían al utilizar estructuras de datos complejas.

Una de estas estructuras de datos es *unión-find*, que ya se mencionó anteriormente y su explicación se puede encontrar en [47].

2.1.1.3.2.1 Ejemplo

Un ejemplo del proceso del algoritmo de Kruskal consta en la Figura 2-6.

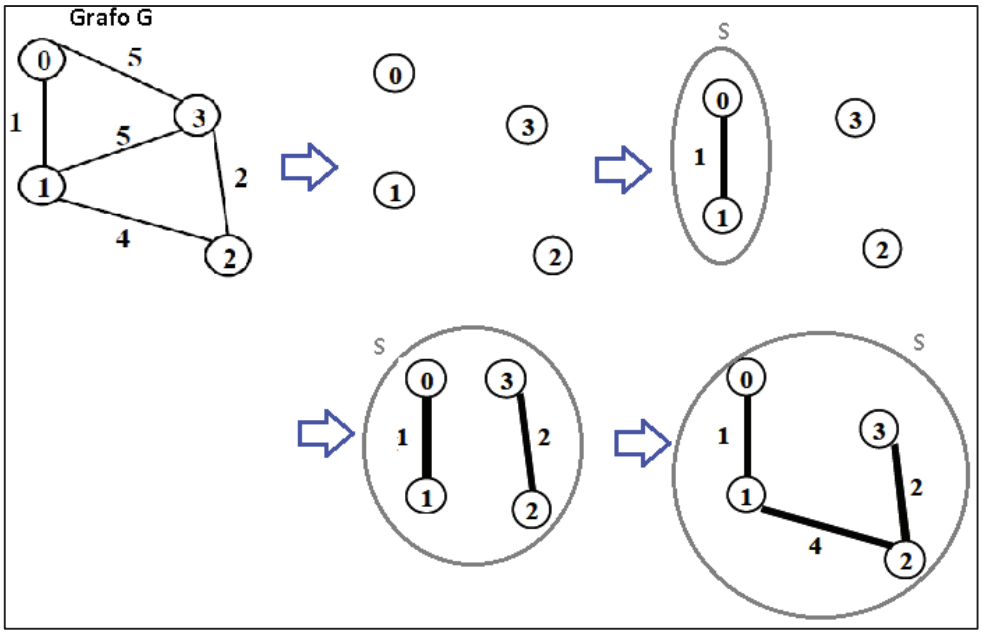


Figura 2-6 Proceso de obtención del MST utilizando el algoritmo de Kruskal [43].

2.1.2 ANÁLISIS COMPARATIVO DE LOS ALGORITMOS EXPUESTOS PARA OBTENER EL MST EN UNA WSN

Conforme a Ernesto Coto en [43], el algoritmo de Prim construye un MST con una arista a la vez, encontrando así una nueva arista que agregar a un MST que va creciendo en cada paso.

El algoritmo de Kruskal también construye un MST con una arista a la vez, con la diferencia que este encuentra una arista que conecte dos MST que van creciendo dentro de un bosque de MST crecientes, formado de los nodos del grafo original.

Finalmente, el algoritmo de Borůvka crea un bosque de supernodos que se van fusionando hasta obtener el MST.

Como se constata en los ítems anteriores, el algoritmo de Prim en su versión básica es el más fácil de implementar y por ende es el menos complejo ya que no necesita estructuras de datos complejas como Borůvka, el cuál utiliza listas o Kruskal que usa *unión-find*.

Debido a que según [48] Zolertia, según [49] TelosB y según [50] Imote2 soportan el lenguaje de programación C, el uso de un algoritmo que requiera estructuras de datos complejas podría complicar el desarrollo de la solución.

La dificultad dependerá del ingenio y la experiencia del programador, ya que sí se pueden construir estructuras de datos en C, tal como se puede apreciar en [51] y la bibliografía citada en el mismo.

La complejidad de ejecución del algoritmo y su complejidad en tiempo resultaron ser inversamente proporcionales para el caso del algoritmo de Prim, ya que si se utiliza la estructura de datos "heap" (*pila),* su complejidad en tiempo es la misma que la de los algoritmos de Borůvka y Kruskal. Es decir: $O_{(m \log n)}$ donde m es el número de iteraciones y n es el número de vértices.

Con respecto al algoritmo de más rápida ejecución, [52] establece un análisis de los 3 algoritmos en cuestión (usando el algoritmo de Prim en su versión básica) y los resultados obtenidos se resumen en la Figura 2-7:

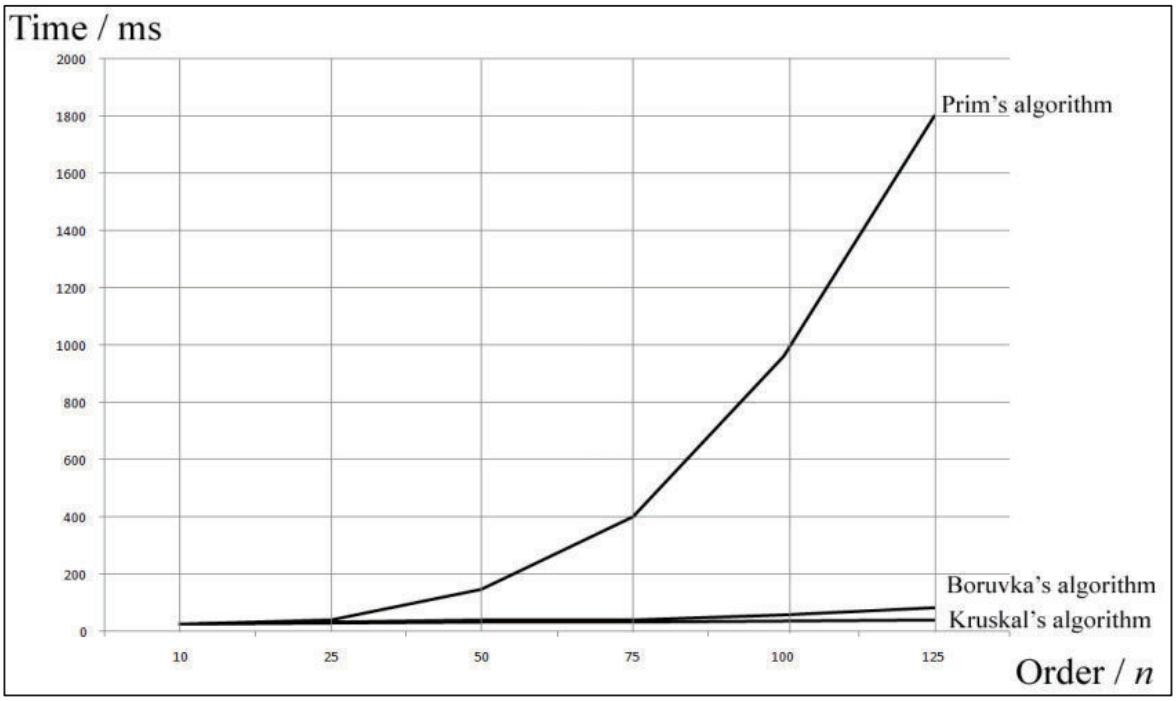


Figura 2-7: Número de nodos v.s. Tiempo de ejecución para los algoritmos de Prim (en su versión básica), Borůvka y Kruskal, según [52].

Como se puede observar en la Figura 2-7, cuando el número de nodos es menor a 25 los tres algoritmos tienen un comportamiento similar. Sin embargo, el algoritmo de Prim tiene ventaja en su versión básica por ser el de menor complejidad.

Por lo tanto, al definir el uso de una WSN de 18 nodos sensores, se utiliza el algoritmo de Prim en su versión básica como el que mejor se adapta a los requerimientos de la implementación propuesta y con el algoritmo en cuestión se procederá a resolver el TSP de forma heurística.

2.1.3 RESOLUCIÓN DEL TSP USANDO EL MINIMUM SPANNING TREE

Para Dallaali [41], se pueden tomar dos caminos para resolver heurísticamente el TSP:

1) Uso del MST como punto de inicio.
2) Uso del MST como una “buena” solución factible.

Luego al recorrer todas las posibles permutaciones factibles:

3) Las permutaciones deben convertir el MST en un tour del grafo original (G).
4) Las restricciones del subtour deben ser revisadas siempre, para asegurarse de que las soluciones son factibles. En otras palabras, las restricciones del subtour deben ser satisfechas.

Lo enunciado anteriormente se puede comprender de mejor manera, con el siguiente ejemplo:

2.1.3.1 MST como punto de inicio

En la Figura 2-8 se puede observar un grafo G de cinco nodos, denotados como A, B, C, D y E; además se puede apreciar el costo de los enlaces que comunican cada uno de los nodos mencionados.

Se encuentra el MST empleando el algoritmo de Prim en su versión básica, al terminar de realizar el procedimiento resumido en la Figura 2-1 se obtiene el MST de la Figura 2-9:

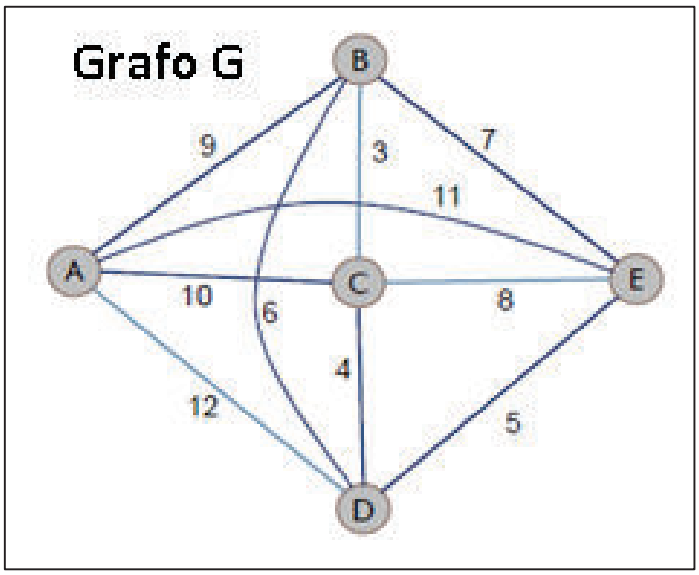


Figura 2-8: Grafo G, pesado y no dirigido [41].

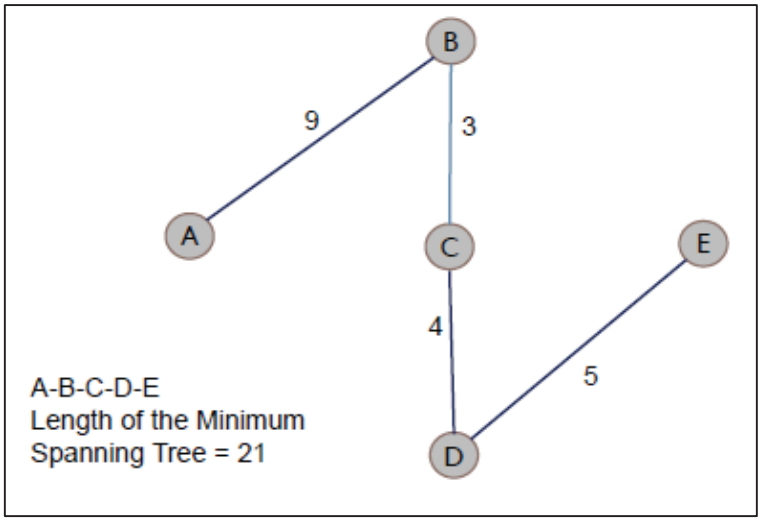


Figura 2-9: MST de G con un costo total de 21, tomado de [41].

2.1.3.2 MST como una "buena" solución factible

En este punto se podría recorrer el grafo en el mismo orden en el que se fueron añadiendo aristas para completar el MST, formando el tour que resuelve el TSP, tal como se plantea en el código publicado en [53].

Al modificar el código mencionado y ponerlo a prueba, se observa que se obtienen buenas aproximaciones al tour óptimo cuando se trabaja con alrededor de 5 a 7 vértices, sin embargo, al resolver el TSP para un número mayor de vértices, el resultado dista en gran medida del tour óptimo.

Como consecuencia de los resultados obtenidos, se ha optado por utilizar el MST como límite superior (véase 2.2.1.3.4).

2.1.3.3 MST convertido en tour a partir de permutaciones

Las permutaciones de las aristas se realizan utilizando el algoritmo 2opt[31], que consiste en reemplazar pares de aristas y comprobar si el costo total del TSP resuelto mejora respecto al último costo obtenido. El proceso se muestra en Figura 2-10

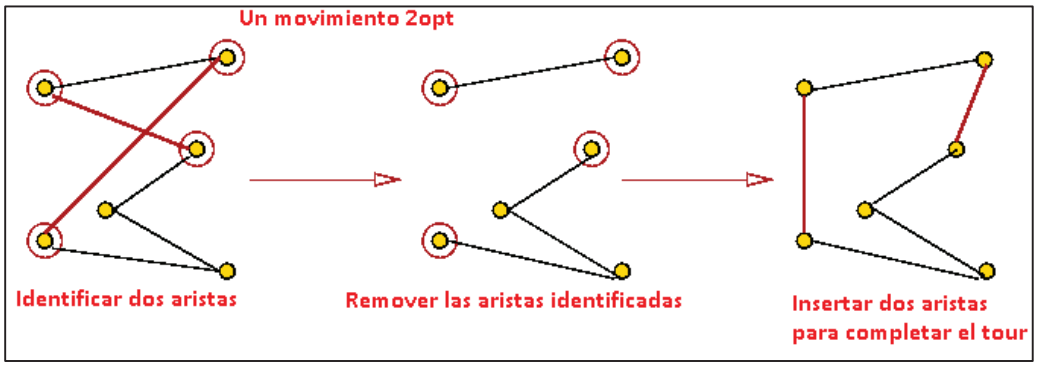


Figura 2-10 Algoritmo 2opt para realizar permutaciones, en busca del tour óptimo [54]

Alexander Paterson actualiza el código expuesto en [53] con el código publicado en [55], se modifica este código para cumplir los requerimientos del presente proyecto y al realizarse pruebas de resolución del TSP con 18 vértices se obtienen mejores resultados, cabe recalcar que es necesario controlar el cumplimiento del límite superior de la solución óptima.

Es importante tener en cuenta que para implementar satisfactoriamente el algoritmo 2opt es fundamental que el grafo sea completo (Véase Figura 1-22), es decir, los enlaces inexistentes se representarán con costos muy altos (idealmente infinitos).

Volviendo al ejemplo de las Figuras 2-8 y 2-9, se procede a recorrer las permutaciones factibles utilizando el algoritmo 2opt:

---

[31] Si un ciclo hamiltoniano se cruza a sí mismo, puede ser fácilmente acortado, basta con eliminar las dos aristas que se cruzan y reconectar los dos caminos resultantes mediante aristas que no se corten, obteniendo un ciclo final es más corto que el inicial. Un movimiento 2-opt consiste en eliminar dos aristas, para posteriormente conectar los 4 vértices que quedaron desconectados de una forma distinta, obteniendo una nueva ruta.

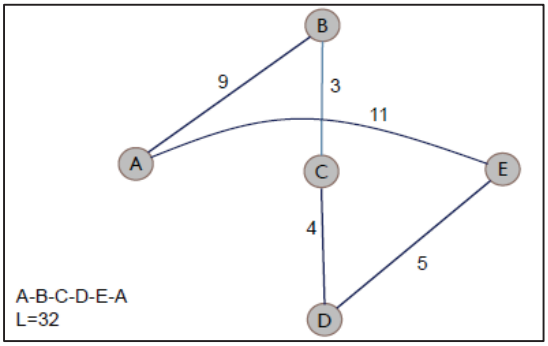


Figura 2-11: Permutación factible de costo total = 32 unidades, tomado de [41].

Como se puede observar en la Figura 2-11, la ruta entre el nodo A y el nodo E cruza a través de la ruta entre el nodo B y el nodo C, por lo tanto las rutas mencionadas pueden ser optimizadas utilizando el algoritmo 2opt, y el resultado se aprecia en la Figura 2-12

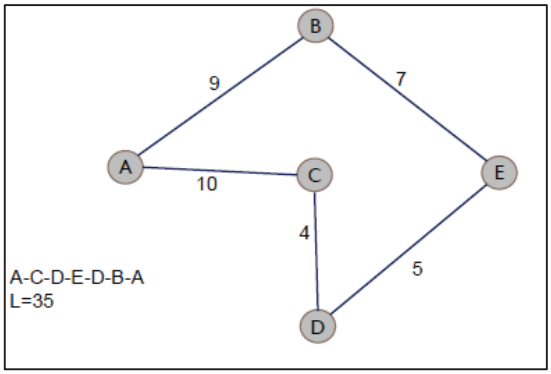


Figura 2-12: Permutación factible de costo total = 35 unidades, tomado de [41].

Otra de las permutaciones factibles es la presentada en la Figura 2-13, donde se puede observar que la ruta entre el nodo A y el nodo C, y la ruta entre el nodo B y el nodo D se cruzan.

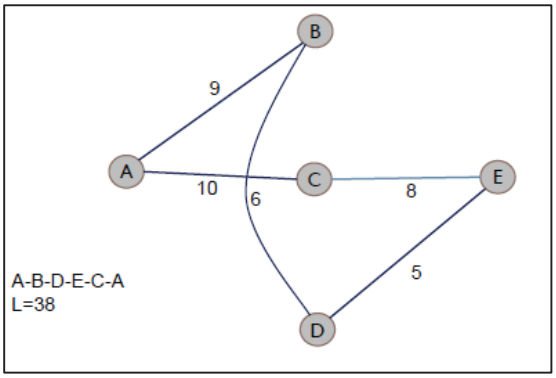


Figura 2-13: Permutación factible de costo total = 38 unidades, tomado de [41].

Al aplicar el algoritmo 2opt en el grafo de la Figura 2-13, se obtiene el grafo de la Figura 2-14, en donde los nodos A, B, C y D se han reconectado en un ciclo sin cruces.

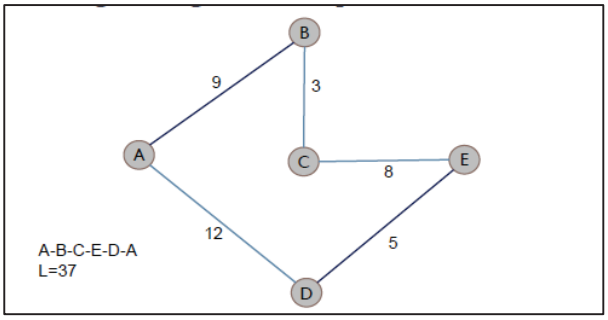


Figura 2-14: Permutación factible de costo total = 35 unidades, tomado de [41].

2.1.3.4 Cumplimiento de restricciones (Límite superior)

En *Depth Last* [55], se demuestra que para un grafo G, el tour que resuelve el TSP de forma aproximada en G, es menor o igual al doble del costo total del MST en G.

Siendo:

- G = El grafo analizado.
- T^* = El tour óptimo que resuelve el TSP.
- e_i = Una arista del grafo G.
- ST_i = Un spanning tree (árbol de expansión).
- MST = El mínimum spanning tree (árbol de expansión mínima) de G.
- P_i = Peso de la i-esima-arista.
- T_a = Tour aproximado al T^*.

Un árbol de expansión se obtiene de extraer una arista del tour óptimo, como se aprecia en Ecuación 2-1

$$T^* - e_j = ST_i$$

Ecuación 2-1 Un Spanning Tree en función del tour óptimo

La sumatoria de los pesos de las aristas del tour óptimo excluyéndole una arista, es mayor o igual al MST que la sumatoria de los pesos de las aristas del MST, véase Ecuación 2-2.

$$\therefore \sum_{p_i=0}^{n} (T^* - e_j) \geq \sum_{p_i=}^{m} MST$$

Ecuación 2-2 Relación entre el tour óptimo y el MST

Por otro lado, como se observa en Ecuaciòn 2-3, se sabe que la suma de los pesos de las aristas del MST es menor a la suma de los pesos de las aristas referentes al tour óptimo.

$$\therefore \sum_{p_i=0}^{n} T^* > \sum_{p_i=0}^{m} MST$$

Ecuación 2-3 Relación entre el tour óptimo y el MST

Debido a que en un tour el grado de todos sus nodos es igual a dos, mientras en un árbol el grado de algunos nodos no va a ser necesariamente igual a dos. Se asume que el tour aproximado al tour óptimo es el MST recorrido dos veces (ida y vuelta) para regresar al vértice inicial. Por lo tanto, la suma de los pesos de las aristas del tour aproximado al tour óptimo es menor o igual al doble de la sumatoria de los pesos de las aristas concernientes al MST.

La Ecuación 2-4 se encuentra explicada (solo usando palabras) en [53] y se utiliza como fundamento del método heurístico propuesto.

$$\therefore \sum_{p_i=0}^{n} T_a \leq \sum_{p_i=0}^{m} (2 \times MST)$$

Ecuación 2-4 Relación entre el tour aproximado al tour óptimo y el MST

En Ecuación 2-3 se puede observar que T^* es mayor al MST, mientras en Ecuación 2-4 se observa que $(2 \times MST)$ es mayor a T_a, entonces $(2 \times T^*)$ será mayor que T_a, tal como se observa en Ecuación 2-5.

$$\therefore \sum_{p_i=0}^{n} T_a \leq \sum_{p_i=0}^{m} (2 \times T^*)$$

Ecuación 2-5 Relación entre el tour aproximado al tour óptimo y tour óptimo

En el código de la simulación se implementa la Ecuación 2-4, con el objetivo de parar la búsqueda de la solución al TSP al obtenerse un tour que comience y termine en el nodo sumidero, y además cumpla la condición de que el costo obtenido sea menor o igual al doble de la sumatoria de las aristas incluidas en el MST.

Los detalles del código implementado pueden observarse en ANEXO 4.

2.2 BRANCH AND BOUND Y EL TRAVELLING SALESMAN PROBLEM

Según [56], *Branch and Bound* (B&B) es un bien conocido método de búsqueda que tiene sus orígenes en el trabajo realizado sobre el TSP. B&B está organizado de manera que realiza una búsqueda exhaustiva de la mejor solución en un conjunto especificado.

Cada paso de ramificación divide el espacio de búsqueda en uno o más subconjuntos, en un intento de crear subproblemas que pueden ser más fáciles de resolver que el original. Mediante la realización repetitiva de pasos de ramificación se crean una colección de subproblemas que necesitan ser resueltos, cada uno de ellos definidos por un subconjunto de tours que incluyen ciertas aristas y excluyen ciertas otras.

Antes de buscar un subproblema y posiblemente continuar dividiéndolo, un límite es computado sobre el costo de los tours.

El propósito del paso de acotación es intentar evitar una búsqueda infructuosa de un subproblema que no contiene una solución mejor de aquella que ya ha sido descubierta.

La idea consiste en que si el límite es más grande que o igual a el costo de un tour que ya hemos encontrado, entonces podemos descartar el subproblema sin ningún peligro de perder el mejor tour.

A continuación, se explica de manera genérica en que consiste el algoritmo de *Branch and Bound*.

2.2.1 INTRODUCCIÓN AL ALGORITMO BRANCH AND BOUND

Según Floudas [57], se enuncia el método *Branch and Bound* que consiste básicamente en aplicar las nociones básicas de separación, relajación y sondeo como se explica a continuación:

2.2.1.1 Separación

Si denotamos el problema a resolverse como (P) y a su conjunto de soluciones factibles (las soluciones que satisfacen todas las restricciones del problema) como $SF(P)$.

Un conjunto de subproblemas $(P_1), (P_2), \dots, (P_n)$ de (P) son definidos como una separación de (P) si se mantienen las siguientes condiciones:

(i) Una solución factible de cualquiera de los subproblemas $(P_1), (P_2), \dots, (P_n)$ es una solución factible de (P);y

(ii) Cada solución factible de (P) es una solución factible de exactamente uno de los subproblemas.

2.2.1.1.1 Observación #1

Las condiciones anteriores (i) y (ii) implican que las soluciones factibles de los subproblemas denotados como $SF(P_1),\ SF(P_2), \dots, SF(P_n)$ son una partición de $SF(P)$.

Como resultado de ello, el problema original (P) es llamado *el "*problema nodo padre" mientras que los subproblemas $(P_1), (P_2), \dots, (P_n)$ son llamados los problemas nodos hijos [57].

2.2.1.2 Relajación

Un problema de optimización, denotado como (PR), es definido como una relajación del problema (P) si el conjunto de soluciones factibles de (P) es un subconjunto del conjunto de soluciones factibles de (PR) [57]; esto se expresa en Ecuación 2-6:

$$SF(P) \subseteq SF(PR)$$

Ecuación 2-6: Las Soluciones Factibles del Problema, están incluidas en las Soluciones Factibles del Problema Relajado, tomado de [57].

2.2.1.2.1 Observación #1

La definición de relajación del párrafo anterior implica las siguientes relaciones entre el problema (P) y el problema relajado (PR) [57]:

(i) Si (PR) no tiene solución factible, entonces (P) no tiene solución factible.

(ii) Si denotamos la solución óptima de (P) como Z_P y la solución óptima del (PR) como Z_{PR} entonces, se debe cumplir la Ecuación 2-7:

$$Z_{PR} \leq Z_P$$

Ecuación 2-7: La solución óptima del Problema Relajado es menor o igual a la solución óptima del Problema en cuestión, tomado de [57].

Esto significa que la solución del problema relajado (PR) provee el límite inferior para la solución del problema (P).

(iii) Si una solución óptima del (PR) es factible para el problema (P), entonces esta es una solución óptima de (P).

2.2.1.2.2 Observación #2

El principal problema en un algoritmo *Branch and Bound* radica en generar una relajación del problema (P) [57].

2.2.1.2.3 Observación #3

La selección de una relajación entre un número de alternativas está basada en la compensación entre dos criterios en disputa.

El primer criterio corresponde a la capacidad de resolver el problema relajado fácilmente.

El segundo criterio está asociado con el tipo y la calidad del límite inferior que el problema relajado produce para el problema (P).

En general, mientras más fácil de resolver es el problema relajado (PR), es más grande la separación entre la solución óptima de (P) y el límite inferior provisto por el (PR) [57].

2.2.1.3 Sondeo

Sea (SC) el subproblema candidato en la resolución de (P). Se desea determinar si la región factible del (SC), conocida como $F(SC)$, contiene una solución óptima de (P), y encontrarla si esta lo contiene. Se puede considerar que un (SC) ha sido sondeado si una de las siguientes dos condiciones se cumple [57]:

(i) Se puede determinar que la solución factible $F(SC)$ no puede contener una mejor solución que la mejor solución encontrada hasta el momento; o
(ii) Una solución óptima del (SC) es encontrada.

En cualquiera de los casos, el subproblema candidato ha sido considerado y no necesita separación adicional.

2.2.1.3.1 Criterios de Sondeo

Existen tres criterios generales de sondeo basados en la relajación, para un algoritmo *Branch and Bound* [57]. Para explicar estos criterios, se utilizará la siguiente notación:

- (RSC), una relajación del subproblema candidato (SC),
- Z_{RSC}, el valor óptimo de (RSC),
- Z_{SC}, el valor óptimo del (SC),
- Z^*, la mejor solución encontrada hasta el momento ($Z^* = \infty$ si no se ha encontrado solución factible de (P) hasta el momento).

2.2.1.3.1.1 Criterio de Sondeo #1

Si (RSC) no tiene solución factible, entonces el (SC) no tiene solución factible y por lo tanto puede ser sondeado [57].

2.2.1.3.1.2 Criterio de Sondeo #2

Para el caso en el que se cumple la Ecuación 2-8:

$$Z_{RSC} \geq Z^*$$

Ecuación 2-8: La solución óptima de la Relajación del Subproblema Candidato es mayor o igual a la mejor solución encontrada hasta el momento, tomado de [57].

Se concluye que el (SC) no puede ser sondeado [57].

2.2.1.3.1.3 Criterio de Sondeo #3

Si una solución óptima de la (RSC) es encontrada de tal manera que es factible en el (SC). Entonces el problema candidato es sondeado.

Nótese que esta solución, también sería factible en (P) y por lo tanto puede actualizar el Z^* si su valor es menor que el Z^* actual.

Existe, cierto número de algoritmos *Branch and Bound*, los cuales no resuelven la (RSC) óptimamente, pero en su lugar aplican suficientes condiciones para cumplir los criterios de sondeo o no solo resuelven la (RSC) óptimamente, sino que aplican pruebas posteriores de optimización con la intención de mejorar los límites inferiores obtenidos mediante la relajación [57].

2.2.2 INTRODUCCIÓN AL LÍMITE INFERIOR DE HELD - KARP

Son necesarios tener claros ciertos conceptos previos antes de entender cómo se calcula el límite inferior de Held - Karp:

2.2.2.1 1 –Tree

Según Rahul Simha [54], al considerar un grafo cuyos vértices van numerados del cero al “n”, de tal forma que un 1-tree (*one tree*) es un subgrafo formado por un árbol de expansión de vértices del uno al “n” y las dos aristas de menor costo que provienen del vértice cero.

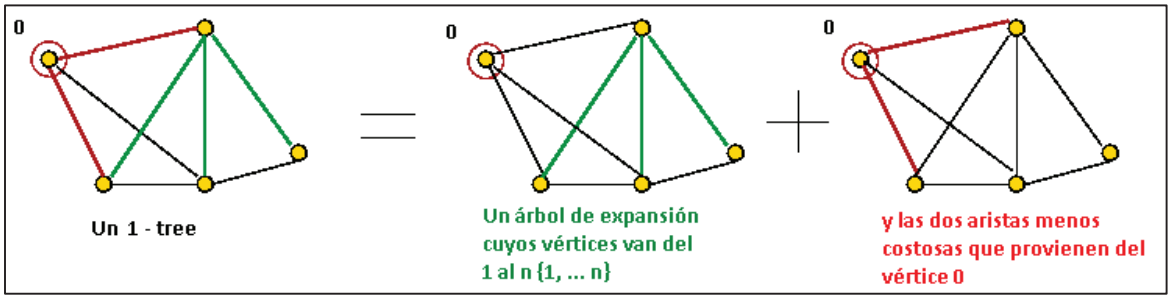


Figura 2-15 Resumen de la elaboración de un 1 – tree [54]

El procedimiento para obtener un 1-tree según la Figura 2-15 se resume en los siguientes pasos:

1) Remover temporalmente el vértice cero (y sus aristas) para encontrar un árbol de expansión usando los vértices del uno al “n” {1,…, n}.
2) Añadir los dos vértices que posean el menor costo y que su origen se establezca en el vértice cero.

Es importante señalar que cada tour (incluyendo el tour óptimo) es un 1 – tree.

2.2.2.1.1 Minimum 1- tree (1 – tree mínimo)

El 1-tree mínimo se define como el 1 – tree de menor costo entre todos los 1 – tree posibles para el grafo en cuestión.

El procedimiento para obtener un 1-tree mínimo se resume en los siguientes pasos:

1) Encontrar el árbol de expansión mínima para el grafo estudiado, sin tomar en cuenta el vértice cero.
2) Añadir los dos vértices de menores costos que provengan desde el vértice cero.

Se debe tener en cuenta que aunque el 1 –tree mínimo se convierte en un límite inferior para el tour óptimo, existe la limitante de que el árbol de expansión mínima puede rechazar el uso de aristas que el tour debe tomar [54].

Esto se puede entender de mejor manera en el ejemplo presentado en la Figura 2-16, a continuación:

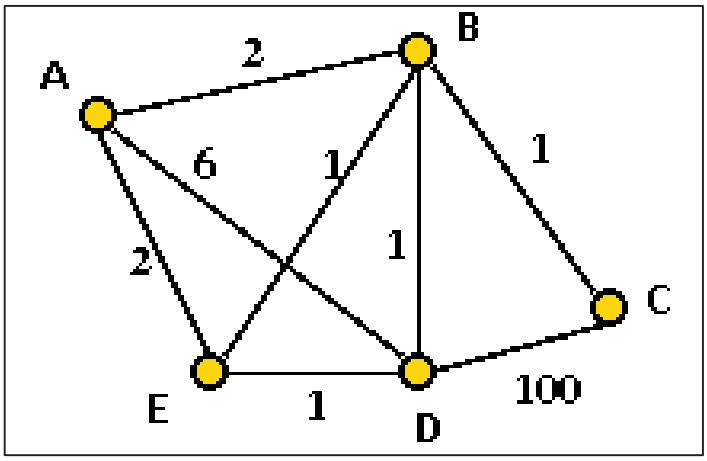
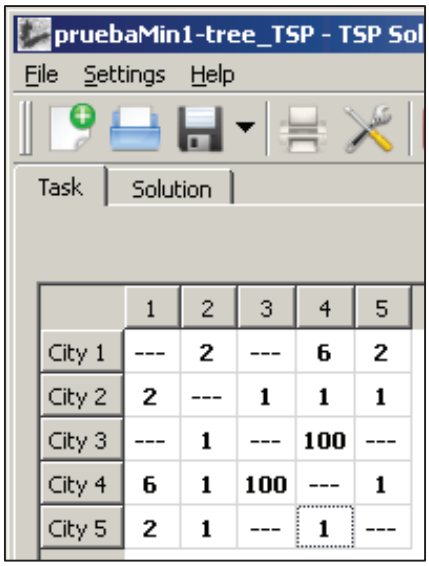


Figura 2-16 Ejemplo de grafo [54], se añade la nomenclatura de los vértices.

Usando el programa "TSPSG"[32] (cómo se muestra en Figura 2-17), se escribe la matriz de costos de la Figura 2-16 y se resuelve el TSP obteniendo un costo total de 106 unidades[33], tal como se observa en la Figura 2-18.



	1	2	3	4	5
City 1	---	2	---	6	2
City 2	2	---	1	1	1
City 3	---	1	---	100	---
City 4	6	1	100	---	1
City 5	2	1	---	1	---

Figura 2-17 Matriz de costos en el programa TSPSG

```
Optimal path:
  City 1 -> City 2 -> City 3 -> City 4 -> City 5 -> City 1
The price is 106 units.
```

Figura 2-18 Resultado obtenido con el programa TSPSG: A -> B -> C -> D -> E -> A, con un costo total de 106 unidades.

Con el mínimo 1 – tree desde el vértice "A" se obtiene un costo total de 7 unidades, tal como se explica a continuación y tal como consta en la Figura 2-19.

Siendo:

- x = La suma del precio de las dos aristas de menor costo que provienen del vértice "A"
- y = El costo total del árbol de expansión mínima, excluyendo al vértice "A" del grafo en cuestión.

[32] TSP *Solver and Generator*
[33] Las unidades dependen de cómo se evalúe el grafo en cuestión, por ejemplo para el presente proyecto estas unidades representan la calidad del enlace según los paquetes recibidos o perdidos por cada nodo de la WSN.

Costo total del mínimo 1 – tree $= (x) + (y) = (2 + 2) + (1 + 1 + 1) = 7$

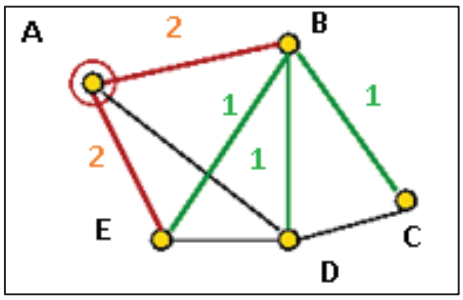


Figura 2-19 Mínimo 1 – tree resaltado en rojo y verde, para el ejemplo planteado en [54]

Según lo enunciado anteriormente, el límite inferior para el tour óptimo lo conforma siete unidades, pero se observa que en la realidad el tour óptimo posee un costo de ciento seis unidades. A pesar de este impase en los cálculos, se utiliza el límite inferior de Held – Karp, que se explicará en el siguiente apartado.

2.2.2.2 El límite inferior de Held – Karp

Según una publicación de la Universidad George Washington en [54], el mejor algoritmo óptimo conocido, es el algoritmo de Held – Karp. Esto se comprueba con lo enunciado en [39], donde se afirma que el límite inferior de Held – Karp es usado para juzgar el desempeño de cualquier nueva solución heurística propuesta para resolver el TSP.

Debido a este nivel de precisión que presenta la relajación del TSP mediante el límite inferior de Held –Karp, se ha escogido este límite inferior como la cota inferior para la resolución del TSP usando el método de *Branch and Bound*. Es así que se tiene un equilibrio respecto a lo enunciado en el apartado 2.2.1.2.3, donde se enuncia una relación inversamente proporcional entre la exactitud de la resolución del TSP y la dificultad en la implementación de la solución.

Erik-Jan [58], da a entender que la idea del límite inferior de Held – Karp consiste en poder alterar el grafo original (G), de tal manera que el grafo modificado (G') genere diferentes mínimos 1-tree que el grafo original. Como resultado, se evitara el impase entre el costo total del mínimo 1 - tree y el costo total del tour óptimo (Véase el ejemplo en 2.2.2.1.1).

La manera en que se modifica al grafo original (G) para obtener el grafo modificado (G') se lo realiza mediante la adición de pesos a los vértices del grafo original, tal como se puede observar en el siguiente ejemplo tomado de [54]. Dichos pesos fueron añadidos arbitrariamente por motivos didácticos (posteriormente se calculará los valores de los pesos para obtener el límite inferior de Held - Karp).

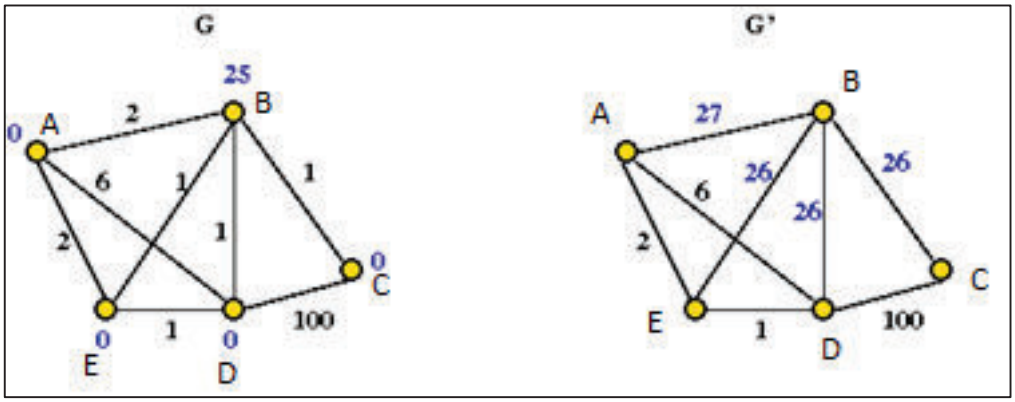


Figura 2-20 Grafo y su modificación, ejemplificando el método de Held-Karp [54].

Como se puede observar en la Figura 2-20, se ha obtenido el grafo modificado para este ejemplo, siguiendo el siguiente proceso (resumido en Ecuación 2-9):

1. G' tiene exactamente el mismo número de vértices y aristas que G.
2. Siendo:
    - e_{ij} = el peso de la arista que va del vértice "i" al vértice "j", en el grafo original (G).
    - c_{ij} = el peso de la arista que va del vértice "i" al vértice "j", en el grafo modificado (G').
    - π_i= el peso del vértice "i".

    Entonces:

$$c_{ij} = e_{ij} + \pi_i + \pi_j.$$

Ecuación 2-9: Relación entre las aristas del grafo modificado (G') y el grafo original (G), teniendo en cuenta los pesos de los vértices, tomado de [54].

En la sección anterior se pudo resolver el TSP con un costo de 106 unidades para el grafo original G (sin pesos en sus vértices), mientras el costo total para el mínimo 1-tree del grafo original G (sin pesos en sus vértices) fue de 7 unidades.

Luego de añadir pesos a los vértices del grafo original y recalcular las aristas según Ecuación 2-4, se obtiene un costo total para la resolución del TSP de 60 unidades y un costo total del mínimo 1-tree de 82 unidades, tal como se detalla en Figura 2-21, Figura 2-22 y Figura 2-23:

	1	2	3	4	5
City 1	---	27	---	6	2
City 2	27	---	26	26	26
City 3	---	0	---	0	0
City 4	6	26	100	---	1
City 5	2	26	---	1	---

Figura 2-21 Matriz de costos del grafo modificado (G'), en la aplicación TSPSG.

```
Optimal path:
  City 1 -> City 5 -> City 2 -> City 3 -> City 4 -> City 1
The price is 60 units.
```

Figura 2-22 Resolución del TSP para el grafo modificado (G'), con la aplicación TSPSG (Tour óptimo: A -> E -> B -> C -> D -> A).

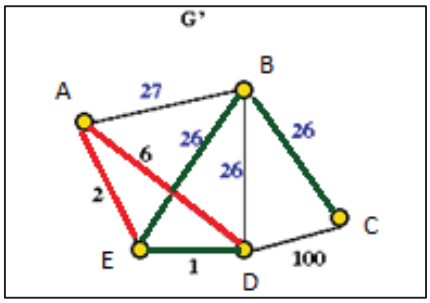


Figura 2-23 Mínimo 1 – tree para el grafo modificado (G') resaltado en rojo y verde, tomado de [54].

Se observa que el mínimo 1 – tree ha mejorado como límite inferior, respecto al tour óptimo en el grafo modificado (G'), siendo el costo total del tour óptimo en el grafo modificado (G') de valor 60 unidades y el mínimo 1-tree para el mismo grafo (G'), un valor de 61 unidades. Consecuentemente, al usar el método de Held – Karp para obtener el límite inferior, este se aproxima de mejor manera a la solución óptima buscada.

Sin embargo, el método de Held – Karp va más allá y detalla un procedimiento para que los pesos asignados a los vértices otorguen resultados aún más precisos.De acuerdo a [54] se resume y explica el procedimiento detallado con rigor específicamente en [59].

Siendo:

- T = Un 1-tree.
- T' = Un tour.
- G = El grafo original.
- G' = El grafo modificado.
- d_i^T = El grado del vértice "i" en el 1-tree.
- L^* = El tour óptimo.
- $L\ (T, G)$ = El costo total del 1-tree, usando el grafo original.
- $L\ (T', G)$ = El costo total del tour, usando el grafo original.

Entonces, ya que todo tour es un 1-tree, se deduce Ecuación 2-10:

$$min_T\ L\ (T, G) \leq min_{T'}\ L\ (T', G).$$

Ecuación 2-10: Desigualdad entre el tour de mínimo costo y el mínimo 1-tree en el grafo original (G), tomado de [54].

Ahora para un 1-tree en función de los grafos original y modificado, se obtiene Ecuación 2-11:

$$L\ (T, G') = L\ (T, G) + \sum_{i \varepsilon T} (d_i^T)\pi_i.$$

Ecuación 2-11: Costo total de un 1-tree en el grafo modificado (G'), en función del costo total del 1 –tree en el grafo original (G), además de los grados y los pesos de los vértices [54].

Y de manera similar, para un tour se obtiene Ecuación 2-12:

$$L\ (T', G') = L\ (T', G) + \sum_{i \varepsilon T'} (2\pi_i).$$

Ecuación 2-12: Costo total de un tour en el grafo modificado (G'), en función del costo total del tour en el grafo original (G), y los pesos de los vértices [54].

Los siguientes pasos se toman de [60], para llegar al mismo resultado que se resume en [54]. Esto se lo realiza con el fin de que el proceso de obtención de la Ecuación 2-25 sea más entendible, sin necesidad de recurrir a demostraciones formales.

Para el mínimo 1-tree en el grafo modificado, se tiene que debido a que el tour óptimo en el grafo original se define como se muestra en Ecuación 2-13:

$$L^* = min_{T'} L\,(T', G)$$

Ecuación 2-13: Definición del tour óptimo en el grafo original (G), tomado de [60].

Entonces, de la Ecuación 2-5 se obtiene la Ecuación 2-14:

$$min_{T'} L\,(T', G') \geq min_T\, L\,(T, G').$$

Ecuación 2-14: Se aplica Ecuación 2-5 en el grafo modificado (G') [60].

Se realizan dos reemplazos como se observa en Ecuación 2-15 y Ecuación 2-16:

$$min_{T'}\,\{\mathrm{L}\left(\mathrm{T}', \mathrm{G}\right) + \sum_{i\varepsilon T'}(2\pi_i)\} \geq min_T\{\mathrm{L}\,(\mathrm{T}, \mathrm{G})\; + \sum_{i\varepsilon T}(d_i^T)\pi_i.\}.$$

Ecuación 2-15: Se reemplazan Ecuación 2-7 y Ecuación 2-6 en Ecuación 2-9 [60].

$$L^* + \sum_{i\varepsilon T'}(2\pi_i) \geq min_T\{\mathrm{L}\,(\mathrm{T}, \mathrm{G})\; + \sum_{i\varepsilon T}(d_i^T)\pi_i.\}.$$

Ecuación 2-16: Se reemplaza Ecuación 2-8 en Ecuación 2-10 [60].

Se despeja el tour óptimo como consta en Ecuación 2-17.

$$L^* \geq min_T\{\mathrm{L}\,(\mathrm{T}, \mathrm{G}) + \sum_{i\varepsilon T}(d_i^T)\pi_i.\} - 2\sum_{i\varepsilon T'}(\pi_i).$$

Ecuación 2-17 Se despeja el tour óptimo de la Ecuación 2-16.

Para comprobar se reordena Ecuación 2-17, obteniendo la Ecuación 2-18:

$$min_T\, L\,(T, G)\; + \sum_{i\varepsilon T}(d_i^T - 2)\pi_i \leq \quad L^*.$$

Ecuación 2-18: Se cumple lo enunciado en Ecuación 2-2 [60].

Siendo la Ecuación 2-19 la función que define los pesos aleatorios, se la relaciona con la Ecuación 2-18 para obtener la Ecuación 2-20.

$$W(\pi)\; = \; min_T\, L\,(T, G)\; + \sum_{i\varepsilon T}(d_i^T - 2)\pi_i.$$

Ecuación 2-19: Definición de los pesos de los vértices como la función W(π) [54].

$$W(\pi)\; \leq \quad L^*.$$

Ecuación 2-20: Relación entre la función de pesos de los vértices y el tour óptimo [54].

Por lo tanto, el mejor límite inferior por el método de Held – Karp, proviene de la maximización de $W(\pi)$ para todos los valores de π. Maximizar $W(\pi)$ implica un proceso de optimización, para el cual se aplica el método del subgradiente, explicado en [54] [59] y [60].

Para la aplicación de dicho método se recurre a un proceso exhaustivo de cálculos cuya explicación está fuera del alcance del presente proyecto de titulación. Consecuentemente, se ha optado por esbozar una explicación a breves rasgos de cómo se optimiza $W(\pi)$ usando el subgradiente. Según [54], se define:

- $V_{T(\pi)} = (dT1, \ldots, dTn)$, el vector de los grados de los vértices del 1-tree.
- $C_{T(\pi)} =$ el costo del mínimo 1-tree usando π.

Entonces, se expresa la función de los pesos como consta en Ecuación 2-21:

$$W(\pi) = C_{T(\pi)} + \pi V_{T(\pi)}$$

Ecuación 2-21: W(π) en función de los grados y los costos del 1 – tree y el mínimo 1-tree respectivamente [54].

Ahora, suponiendo que π' es un vector en el π-espacio, se tiene Ecuación 2-22:

$$W(\pi') \geq W(\pi)$$

Ecuación 2-22: Relación entre W(π') y W(π), tomado de [54].

Entonces, Held – Karp demuestra la Ecuación 2-23, basándose en Ecuación 2-22:

$$(\pi' - \pi) V_{T(\pi)} \geq 0$$

Ecuación 2-23: Orientación de W(π'), tomado de [54].

La Ecuación 2-22 implica que los valores más altos de $W(\pi')$ están en la mitad derecha del espacio dividido por $W(\pi)$. Para poder visualizar lo enunciado es necesario imaginar una recta numérica cortada por el plano $W(\pi')$.

Entonces, mediante Held-Karp se concluye además que los valores que maximizan los valores que sirven para llegar a la solución óptima del TSP, se encuentran a la derecha del espacio dividido por el vector $V_{T(\pi)}$.

2.2.2.2.1 Optimización basada en gradientes

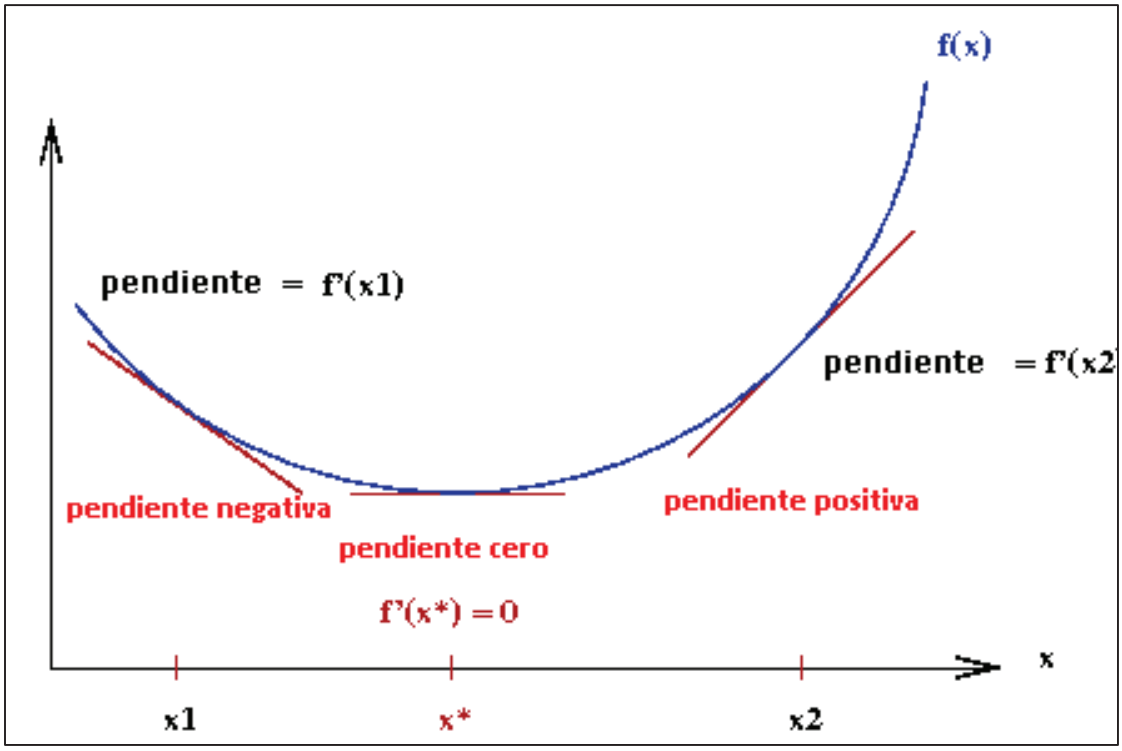


Figura 2-24 Gradientes para un espacio de dimensión simple, tomado de [54].

```
while not over
    if gradient < 0
        move rightwards
    else if gradient > 0
        move leftwards
    else
        stop
    endif
endwhile
```

Figura 2-25 Bosquejo del algoritmo para el trabajo con gradientes, tomado de [54].

En la Figura 2-24, sea f'(x) la derivada de f(x), entonces el gradiente en el punto "x" es el valor de f'(x).Por lo tanto la izquierda del valor óptimo x* el gradiente es negativo y a la derecha es positivo.

Para encontrar el valor óptimo se debe formular un algoritmo iterativo, como se muestra en la Figura 2-25. Se puede observar cómo se van probando gradientes

hasta llegar a la solución óptima, de tal manera que si el gradiente medido es menor que cero se ordena dar un “paso” hacia la derecha y viceversa.

2.2.2.2.1.1 Gradiente descendente

Para el caso de gradientes descendientes, se añade un factor de escalamiento, al que se denominará como “α” por el momento, en caso de que los valores de f'(x) sean de un diferente orden de magnitud [54], tal como consta en Figura 2-26.

```
while not over
    x = x - α f'(x)
endwhile
```

Figura 2-26 Bosquejo del algoritmo para el trabajo con gradientes descendentes, tomado de [54].

2.2.2.2.2 Optimización por subgradientes

En el caso de que la función no sea diferenciable, tal como sucede con W(π), el matemático ruso Polyak elaboró el algoritmo de optimización por subgradientes, donde para los pesos de los vértices las iteraciones se expresan en Ecuación 2-24 [54]:

$$\pi_I^{(m+1)} = \pi_I^{(m)} + \alpha^{(m)}(d_i - 2)$$

Ecuación 2-24: Ecuación para obtener mediante iteraciones el peso del vértice siguiente a probarse, según [54], en [59] “α” se representa como “t”.

La Ecuación 2-24 significa que:

- Se incrementan los pesos para los vértices con mínimo 1 – tree de grado mayor que 2.
- Se decrementan los pesos para los vértices con mínimo 1 – tree de grado mayor que 2.

Por lo tanto, mediante iteraciones lo que se trata es de forzar a que el mínimo 1 – tree se parezca lo más posible a un tour.

Polyak demostró que la iteración por subgradientes funciona si: los “tamaños de los pasos” (nombrado $\alpha^{(m)}$ en [54] y $t^{(m)}$ en [59]) son escogidos apropiadamente. Para el

presente trabajo se ha escogido el método N°1 propuesto en [59], para el cálculo de $t^{(m)}$ como se muestra en Ecuación 2-25:

$$t^{(m)} = \frac{\lambda_m (U - w(\pi^{(m)}))}{\sum_{i-1}^{n} (d_i^{(m)} - 2)^2}$$

Ecuación 2-25: Cálculo de los "tamaños de los pasos" según [59].

La Ecuación 2-25 también se encuentra en [60] junto a un ejemplo que permite entender de mejor manera las variables que se explican a continuación:

Como se explica en [61], se define "U" como un sobrestimado de la solución óptima para W(π). Por otro lado [62] menciona que, los resultados no dependen de manera crítica del valor de "U", por lo que se toma su valor como el doble del límite inferior para el presente trabajo.

Respecto a ʎ, en [62] se reporta un valor mínimo de 1/8 para resolver casos del TSP en un rango de 42 a 57 ciudades. En el presente trabajo se aproxima el valor de 1/8 (0.125) a 0.1 inicialmente, para después incrementarlo en pasos de 0.9 según [63] y [64].

2.2.3 RESOLUCIÓN DEL TSP USANDO EL ALGORITMO BRANCH AND BOUND

Para resolver el TSP usando el método de *Branch and Bound* se toma como base el trabajo teórico realizado en [59] y práctico en [63] y [64].

En dichas fuentes se hace uso del límite inferior de Held – Karp para mejorar el desempeño del algoritmo de *Branch and Bound*. Tal decisión se justifica según [59], donde la aplicación de métodos exactos para resolver problemas del tipo NP duro/completo requiere de un tiempo de ejecución exponencial.

Este es el caso del algoritmo de *Branch and Bound*, afortunadamente la evaluación del límite inferior de Held – Karp mediante la optimización por subgradiente no manifiesta una gran carga de computo cuando se usan 18 nodos, y de hecho podrían realizarse pruebas con centenares de nodos sensores.

Con los conocimientos explicados en los apartados 2.2.1 y 2.2.2, se resuelve el TSP en dos etapas. Como se establece en [59], consisten en primera instancia en computar un límite inferior para el TSP y en segunda instancia incorporar un procedimiento de *Branch and Bound* en el proceso de relajación para producir soluciones factibles del TSP en cuestión.

Los detalles de la implementación de la solución se pueden encontrar en ANEXO 5, donde se detalla el código implementado y se explica la lógica de las dos etapas mencionadas anteriormente.

2.2.3.1 El mínimo 1-tree como límite inferior del TSP (Primera aproximación)

Después de haber obtenido la matriz de costos correspondiente al grafo conformado por los nodos de la WSN a simularse, el siguiente paso es encontrar el costo total de un mínimo 1-tree sin usar Held-Karp (es decir, sin que los nodos posean pesos asociados).

Cuando se inicializan las variables del algoritmo, el límite inferior óptimo por defecto es infinito (Véase 2.2.1.3.1). El objetivo de encontrar el mínimo 1-tree, sin usar Held-Karp, es establecer el primer límite inferior aproximado con el que se pueda empezar a trabajar, ya que un mínimo 1-tree es por defecto un límite inferior en la resolución del TSP (Véase 2.2.2.1.1).

Cabe recalcar que además de no trabajar con Held-Karp en este punto, tampoco se empieza la puesta en marcha del algoritmo de *Branch and Bound* por el momento.

2.2.3.2 Mejoramiento del mínimo 1-tree como límite inferior: método de Held-Karp (Segunda aproximación)

El método de Held-Karp se encarga de calcular los pesos de los nodos basándose fundamentalmente en la Ecuación 2-25 (Véase 2.2.2.2), este cálculo se repite de tal manera que la duración del ciclo de iteraciones es controlada cambiando el valor de la variable lambda en Ecuación 2-25. Según el trabajo de Valenzuela en [59] lambda es adimensional y para este proyecto empieza en 0.1 unidades, para luego

multiplicarlo por 0.9 unidades en cada iteración, hasta que lamba llegue a las 0.000001 unidades.

En cada iteración se actualiza el costo de las aristas del grafo según la Ecuación 2-9, con estos valores actualizados se procede a actualizar también el costo total del mínimo 1-tree. En cada actualización del mínimo 1-tree se busca minimizar su costo total y como consecuencia, mejorar el límite inferior del TSP.

2.2.3.3 Puesta en marcha de B&B utilizando el límite inferior obtenido mediante Held-Karp (Segunda aproximación)

2.2.3.3.1 Evaluación del límite inferior (segunda aproximación) utilizando criterios de B&B

Al término de cada iteración de Held-Karp se obtiene un nuevo límite inferior, que inmediatamente es evaluado con dos criterios propios del algoritmo de *Branch and Bound*:

1) ¿El límite inferior obtenido con Held-Karp es mayor que el mejor límite inferior obtenido hasta ahora?
   De ser afirmativa la respuesta, no vale la pena seguir iterando con Held-Karp, ya que este problema o subproblema de B&B no nos conduce a la solución óptima, la cual busca el mínimo costo total con el que se resuelve el TSP. Si la respuesta es negativa se continúa con la segunda pregunta a continuación.

2) ¿Se puede mejorar el límite inferior obtenido en la última iteración?
   En este punto se sabe que se está trabajando sobre un límite inferior que puede o no ser inferior al mejor límite inferior obtenido hasta el momento. Aunque, no se conoce si en una iteración pasada de Held-Karp ya se obtuvo el mejor límite posible.

   Si ya se obtuvo el mejor límite inferior, no se sigue variando la variable lambda de la Ecuación 2-25 y se mantiene este límite hasta que el proceso de B&B determine la mejor solución al TSP, de ser negativa la respuesta, el proceso de Held-Karp continuará normalmente.

2.2.3.3.2 Generación de los subproblemas relajados y su resolución.

Cada vez que se resuelve un problema o subproblema encontrando el límite inferior por Held-Karp, se procede a almacenar la información obtenida en una cola de prioridades (ordenado de menor a mayor según el límite inferior obtenido).

A continuación, como se sabe que el tour que resuelva el TSP deberá obligatoriamente poseer todos los nodos de la WSN con grado igual a dos, entonces se procede a excluir los enlaces de aquellos nodos con grados mayores a dos, generando así subproblemas que deberán volver a ser evaluados con Held-Karp utilizando los criterios de B&B.

Para comprender de mejor forma lo enunciado en los ítems 2.2.3.1, 2.3.2 y 2.3.3, se procede a mostrar ejemplos de resolución del TSP con diferente número de nodos.

2.2.3.4 Ejemplos de resolución del TSP con diferente número de nodos

Cada tabla a continuación muestra el contenido de las estructuras de datos utilizadas en el código que resuelve el TSP usando B&B, para el presente trabajo.

matCostos: Es un arreglo de dos dimensiones que contiene la matriz de costos que se obtiene de evaluar la calidad de cada enlace en la simulación de la WSN.

El valor 2,15E+18, representa el valor “infinito” para dar a entender al programa que el enlace con ese costo es un enlace inexistente en la WSN. (Véase las Tablas 2-4, 2-8).

matCostosConPesosVertcs: Es un arreglo de dos dimensiones que contiene la matriz de costos actualizada después de obtener el valor de cada enlace teniendo en cuenta la matriz original y los costos calculados para los nodos según Held-Karp, según la Ecuación 2-9. (Véase Tabla 2-9).

También 2,15E+18, representa el valor “infinito” para dar a entender al programa que el enlace con ese costo es un enlace inexistente en la WSN.

vtorDeValAIndc: Es un arreglo que contiene que nodo se conecta a que nodo en el mínimo 1-tree resuelto. (Véase Tablas 2-1, 2-5 y 2-10),

vtorGrdsVrtcs: Es un arreglo que contiene el grado de cada nodo del mínimo 1-tree resuelto. (Véase Tablas 2-2, 2-6 y 2-11)

vctrPesosVrtcs: Es un arreglo que contiene los pesos asignados a los nodos de la WSN, calculados mediante el método de Held-Karp. (Véase Tablas 2-3, 2-7 y 2-12).

2.2.3.4.1 Resolución del TSP en una WSN de cuatro nodos

2.2.3.4.1.1 Primera Aproximación al límite inferior

matCostos	
[0]	Fila
[0] columna	2,15E+18
[1] columna	99.0
[2] columna	56.0
[3] columna	1.0
[1]	Fila
[0] columna	99.0
[1] columna	2,15E+18
[2] columna	21.0
[3] columna	71.0
[2]	Fila
[0] columna	56.0
[1] columna	21.0
[2] columna	2,15E+18
[3] columna	81.0
[3]	Fila
[0] columna	1.0
[1] columna	71.0
[2] columna	81.0
[3] columna	2,15E+18

Tabla 2-4 Matriz con los pesos de los nodos, para un grafo de 4 nodos.

vtorDeValAIndc	
[0]	2
[1]	3
[2]	1
[3]	0

Tabla 2-1 Matriz de conexiones para un grafo de 4 nodos.

vtorGrdsVrtcs	
[0]	2
[1]	2
[2]	2
[3]	2

Tabla 2-2 Matriz con los grados de un grafo de 4 nodos.

vctrPesosVrtcs	
[0]	0
[1]	0
[2]	0
[3]	0

Tabla 2-3 Matriz de costos para un grafo de 4 nodos.

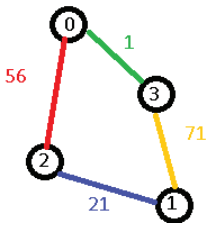


Figura 2-27 Primer mínimo 1-tree obtenido para un grafo de 4 nodos.

Al resolverse la Ecuación 2-25 en cada iteración, se puede observar el peso de los vértices calculado por Held-Karp es tan pequeño que tiende a cero (con valores en el orden de 1x10^-5), y por lo tanto el peso que se asigna a los nodos es de cero.

Además, como se puede constatar en la Figura 2-27, para este caso el mínimo 1-tree ya es de por sí un tour (el grado de todos sus nodos es igual a dos) de mínimo costo y por lo tanto el desarrollo implementado para el presente proyecto, muestra una de las soluciones aproximadas al TSP como la solución final, en Figura 2-28.

```
El mejor límite inferior hasta el momento es: 149.0
Tour TSP Resuelto:
ID del nodo/vértice: 0
ID del nodo/vértice: 2
ID del nodo/vértice: 1
ID del nodo/vértice: 3
ID del nodo/vértice: 0
```

Figura 2-28 Resultado final de la resolución del TSP para un grafo de 4 nodos, utilizando el desarrollo implementado para el presente proyecto.

2.2.3.4.1.2 Comprobación

Al ser un grafo pequeño, su TSP se puede resolver fácilmente por cualquier método (inclusive manualmente) y por lo tanto las respuestas coinciden.

Como se señaló anteriormente, mientras mayor es el número de nodos, es más difícil resolver el TSP en un tiempo polinomial. Se procede a resolver el mismo grafo en el programa de código abierto TSPSG [65]:

```
Selected route with (3;1) part.
1 alternate candidate for branching: (4;2).

Optimal path:
  City 1 -> City 4 -> City 2 -> City 3 -> City 1
The price is 149 units.
```

Figura 2-29 Resolución del TSP para un grafo de 4 nodos, utilizando el programa TSPSG.

En la Figura 2-29 se observa una parte del resultado final para la resolución del TSP en un grafo de cuatro vértices.

2.2.3.4.2 Resolución del TSP en una WSN de cinco nodos

2.2.3.4.2.1 Primera aproximación al límite inferior

matCostos	
[0]	Fila
[0] columna	2,15E+18
[1] columna	99.0
[2] columna	56.0
[3] columna	1.0
[4] columna	41.0
[1]	**Fila**
[0] columna	99.0
[1] columna	2,15E+18
[2] columna	21.0
[3] columna	71.0
[4] columna	51.0
[2]	**Fila**
[0] columna	56.0
[1] columna	21.0
[2] columna	2,15E+18
[3] columna	81.0
[4] columna	1.0
[3]	**Fila**
[0] columna	1.0
[1] columna	71.0
[2] columna	81.0
[3] columna	2,15E+18
[4] columna	51.0
[4]	**Fila**
[0] columna	41.0
[1] columna	51.0
[2] columna	81.0
[3] columna	51.0
[4] columna	2,15E+18

Tabla 2-8 Matriz de costos para un grafo de 5 nodos.

vtorDeValAIndc	
[0]	4
[1]	4
[2]	1
[3]	0
[4]	3

Tabla 2-5 Matriz de conexiones para un grafo de 5 nodos.

vtorGrdsVrtcs	
[0]	2
[1]	2
[2]	1
[3]	2
[4]	3

Tabla 2-6 Matriz con los grados de un grafo de 5 nodos.

vctrPesosVrtcs	
[0]	0
[1]	0
[2]	0
[3]	0
[4]	0

Tabla 2-7 Matriz con los pesos de los nodos, para un grafo de 5 nodos.

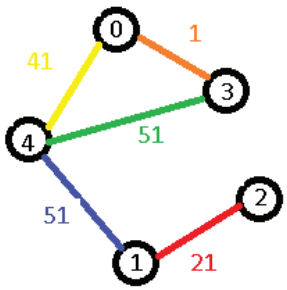


Figura 2-30 Primer min. 1-tree obtenido para un grafo de 5 nodos.

2.2.3.4.2.2 Segunda aproximación al límite inferior y aplicación de los criterios de B&B

matCostosConPesosVertcs	
[0] columna	2,15E+18
[1] columna	99.0
[2] columna	45.542225
[3] columna	1.0
[4] columna	51.457775
[1]	Fila
[0] columna	99.0
[1] columna	2,15E+18
[2] columna	10.542224999999900
[3] columna	71.0
[4] columna	61.457775
[0] columna	45.542.225
[1] columna	10.542224999999900
[2] columna	2,15E+25
[3] columna	70.542225
[4] columna	1.0
[0] columna	1.0
[1] columna	71.0
[2] columna	70.542225
[3] columna	2,15E+18
[4] columna	61.457775
[0] columna	51.457775
[1] columna	61.457775
[2] columna	81.0
[3] columna	61.457775
[4] columna	2,15E+23

Tabla 2-9 Matriz de costos (incluyendo los pesos de los 5 nodos del grafo por Held-Karp).

vtorDeValAIndc	
[0]	2
[1]	4
[2]	1
[3]	0
[4]	3

Tabla 2-10 Matriz de conexiones para un grafo de 5 nodos tratado con Held-Karp

vtorGrdsVrtcs	
[0]	2
[1]	2
[2]	2
[3]	2
[4]	2

Tabla 2-11 Matriz con los grados de los vértices para un grafo de 5 nodos modificado por Held-Karp.

VctrPesosVrtcs	
[0]	0.0
[1]	0.0
[2]	-10.457775
[3]	0.0
[4]	10.457775

Tabla 2-12 Matriz con los pesos de los vértices para un grafo de 5 nodos modificado por Held-Karp.

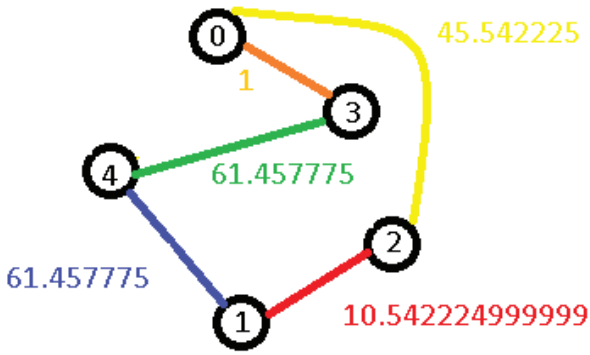


Figura 2-31 Mínimo 1-tree obtenido al modificar el grafo original de 5 nodos, utilizando Held-Karp.

En este caso se resuelve la Ecuación 2-25 en cada iteración y se va obteniendo valores diferentes para los pesos de los vértices. Por lo tanto, en este caso a diferencia del ejemplo con cuatro nodos, ya se puede observar la segunda aproximación al límite inferior usando Held-Karp, donde el grafo de la Figura 2-30 se termina convirtiendo en el tour de la Figura 2-31.

En lo que respecta a B&B, se evalúan los diferentes límites inferiores obtenidos usando Held-Karp utilizando los criterios del algoritmo de B&B, hasta llegar a un límite inferior de 180. Aunque hay que resaltar que debido a que al terminar las iteraciones de la segunda aproximación ya se obtiene un tour (el grado de todos los

nodos del grafo es dos) de mínimo costo, no se procede a generar subproblemas relajados y resolverlos.

```
run:
El mejor límite inferior hasta el momento es: 180.0
Tour TSP Resuelto:
ID del nodo/vértice: 0
ID del nodo/vértice: 2
ID del nodo/vértice: 1
ID del nodo/vértice: 4
ID del nodo/vértice: 3
ID del nodo/vértice: 0
```

Figura 2-32 Resultado final de la resolución del TSP para un grafo de 5 nodos modificado con Held-Karp, utilizando el desarrollo para el presente proyecto.

2.2.3.4.2.3 Comprobación

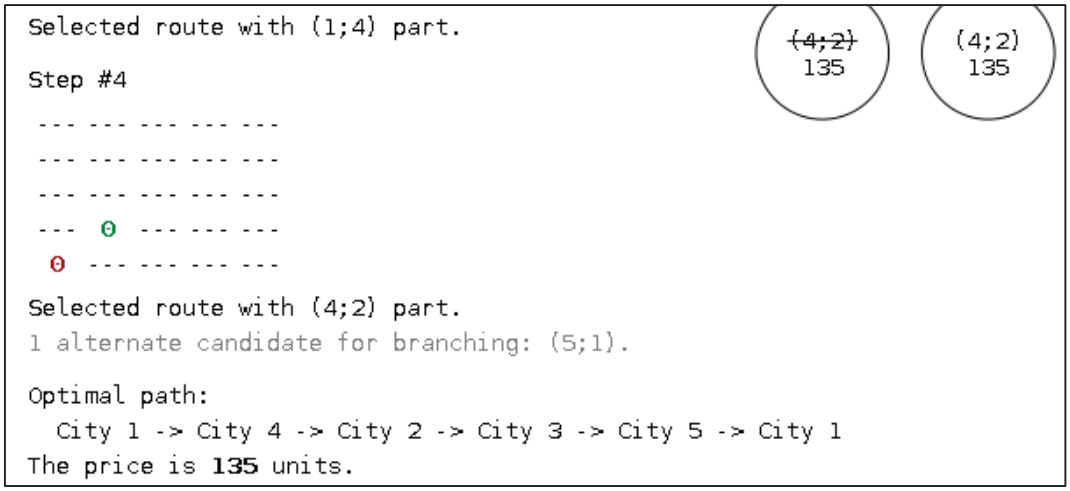


Figura 2-33 Resolución del TSP para un grafo de 5 nodos, utilizando el programa TSPSG.

Se comprueba la resolución del problema con el programa TSPSG, como consta en la Figura 2-33), este programa posee un criterio diferente para generar subproblemas relajados, y por lo tanto no coinciden con el resultado obtenido del desarrollo planteado en el presente proyecto (Véase Figura 2-32)

La diferencia en los resultados es explicable por que el criterio aplicado a la generación de subproblemas relajados para el presente proyecto de titulación no ha sido trabajado para que al ajustar los parámetros de la Ecuación 2-25 se tengan valores del TSP resueltos con un número de nodos igual a 5 (en este proyecto se trabaja con 18 nodos).

En un grafo de cinco nodos la resolución del TSP puede ser llevada a cabo de forma manual, sin necesidad de acudir a métodos sofisticados.

2.2.3.4.2.4 Nota Importante

Se debe considerar que, si el número de nodos cambia, también se debe alterar el valor de lambda y afinar la Ecuación 2-25 según las consideraciones en [59].

2.2.3.4.3 Resolución del TSP en una WSN de dieciocho nodos

Para un grafo de 18 nodos se tiene demasiados datos como para expresarlos o resumirlos de manera entendible, por lo tanto, basándose en los ejemplos anteriores se puede comprender de mejor forma los siguientes gráficos, sin necesidad de presentar todos los datos obtenidos (tan solo la matriz de costos contiene 324 valores).

2.2.3.4.3.1 Primera aproximación al límite inferior

Se obtiene el grafo de la Figura 2-34 al resolver el mínimo 1-tree sin utilizar el método de Held-Karp.

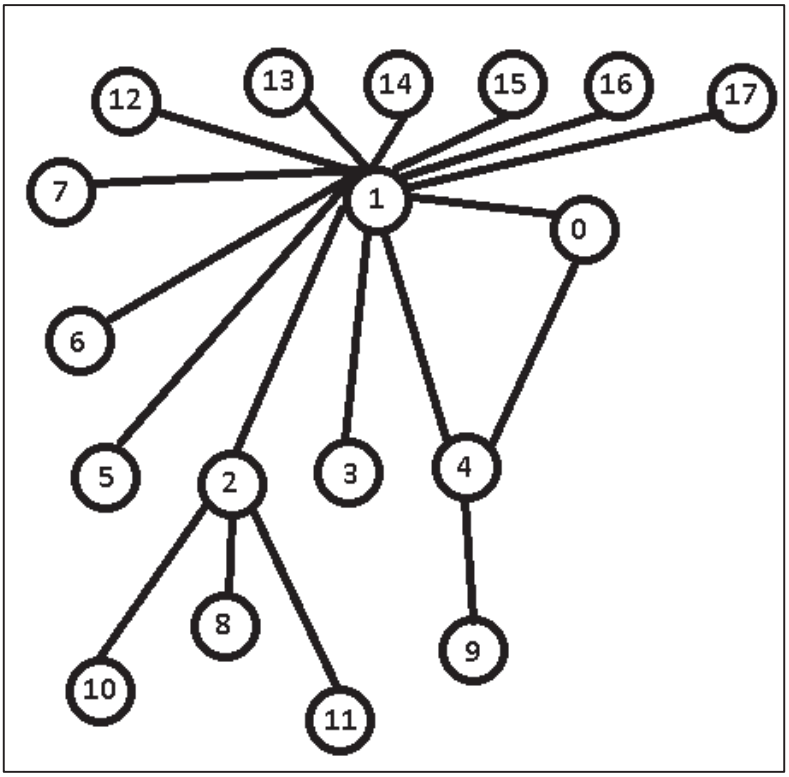


Figura 2-34 Primer mínimo 1-tree obtenido para un grafo de 18 nodos.

2.2.3.4.3.2 Segunda aproximación al límite inferior y aplicación de los criterios de B&B

Al utilizarse el método de Held-Karp en conjunto con los criterios de B&B para conseguir el límite inferior, se obtiene un grafo con varios nodos de grado mayor a dos.

Como se puede observar en la Figura 2-35, a diferencia del grafo en la Figura 2-34, las conexiones ya no se centralizan en el nodo 1.

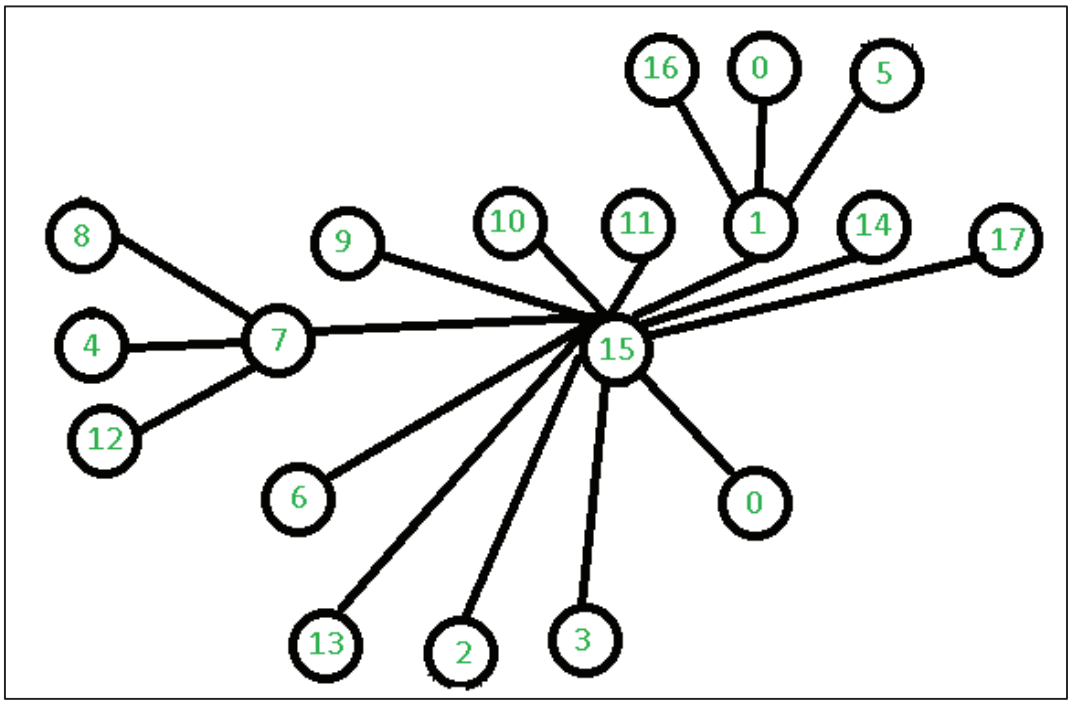


Figura 2-35 Mínimo 1-tree obtenido al modificar el grafo original de 18 nodos, utilizando Held-Karp.

2.2.3.4.3.3 Generación de los subproblemas relajados y su resolución

En la Figura 2-35, se observa que los nodos 15 y 1, son los nodos que comparten un enlace y a la vez poseen mayores grados en todo el grafo, por lo tanto se genera un subproblema relajado en el que se excluye el enlace entre los nodos mencionados.

Los nodos 15 y 7 también podrían ser excluidos pero el programa esta implementado de forma que busca secuencialmente (en orden, según el ordinal del nodo) los enlaces de grado más alto. Cada subproblema es tratado como el problema original, es decir, también se aplica la primera y segunda aproximación del límite inferior a cada subproblema relajado como se aprecia en las Figuras 2-36 y 2-37.

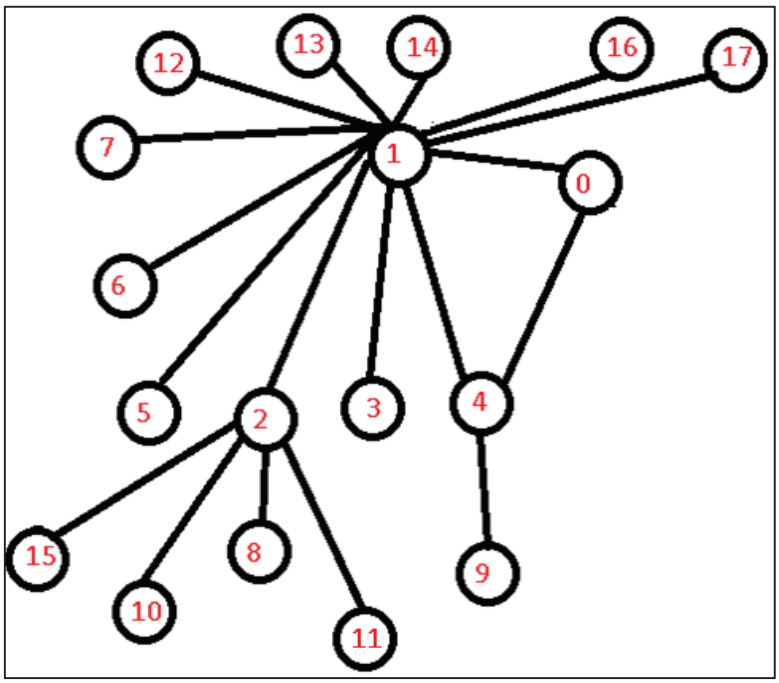


Figura 2-36 Grafo del suproblema relajado (no existe enlace entre los nodos 1 y 15) donde se aplica la primera aproximación.

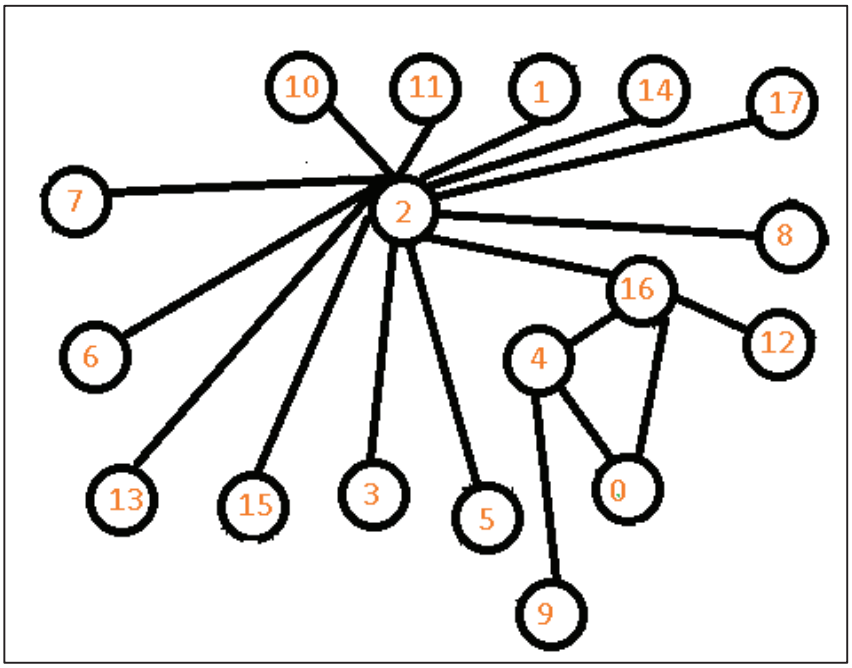


Figura 2-37 Grafo del suproblema relajado (no existe enlace entre los nodos 1 y 15) donde se aplica la segunda aproximación.

Y así mismo se continúa generando subproblemas de los subproblemas (teniendo en cuenta los criterios de ramificación de B&B), con el fin de relajar el problema original hasta que el TSP sea resuelto.

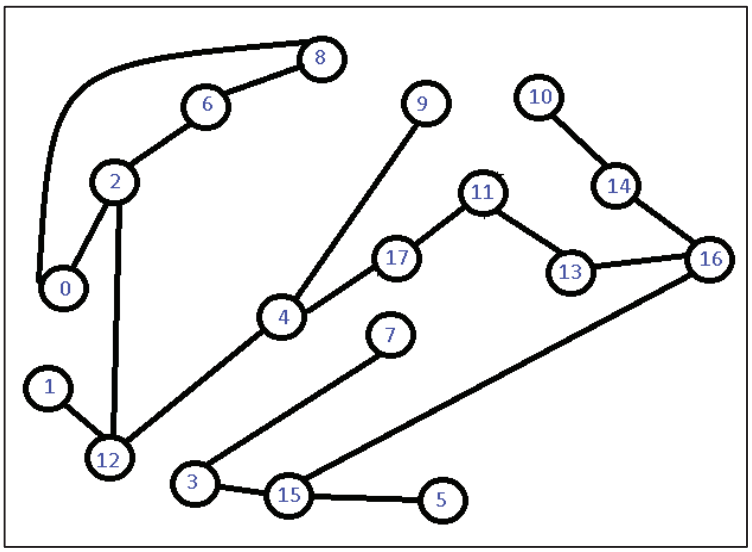


Figura 2-38 Grafo de un subproblema relajado a mitad del proceso para encontrar la solución al TSP planteado.

La Figura 2-38 muestra uno de los múltiples grafos obtenidos, que en cada ramificación de B&B se va acercando más a la solución del TSP. Anteriormente se mencionó (Véase 2.2.1.2) que la resolución del (sub)problema relajado implica la resolución del problema planteado

```
run:
El mejor límite inferior hasta el momento es: 53.0
El mejor límite inferior hasta el momento es: 18.0
Tour TSP Resuelto:
ID del nodo/vértice: 0
ID del nodo/vértice: 17
ID del nodo/vértice: 2
ID del nodo/vértice: 1
ID del nodo/vértice: 5
ID del nodo/vértice: 16
ID del nodo/vértice: 13
ID del nodo/vértice: 11
ID del nodo/vértice: 3
ID del nodo/vértice: 15
ID del nodo/vértice: 10
ID del nodo/vértice: 14
ID del nodo/vértice: 9
ID del nodo/vértice: 4
ID del nodo/vértice: 7
ID del nodo/vértice: 12
ID del nodo/vértice: 8
ID del nodo/vértice: 6
ID del nodo/vértice: 0
```

Figura 2-39 Resultado final de la resolución del TSP para un grafo de 18 utilizando el desarrollo para el presente proyecto.

Finalmente se obtiene la resolución del TSP, como se puede observar en la Figura 2-39 el orden de aparición del nodo es el orden que debe seguir el agente viajero.

2.2.3.4.3.4 Comprobación

Se resuelve el mismo problema con el programa TSPSG [65] y se obtiene el mismo resultado, tal como se puede observar en la Figura 2-40:

```
Optimal path:
  City 1 -> City 2 -> City 3 -> City 4 -> City 5 -> City 7 ->
City 8 -> City 9 -> City 10 -> City 13 -> City 18 -> City 16
-> City 11 -> City 15 -> City 14 -> City 17 -> City 6 -> City
12 -> City 1
The price is 18 units.
```

Figura 2-40 Resolución del TSP para un grafo de 5 nodos, utilizando el programa TSPSG.

3 CAPÍTULO 3: SIMULACIÓN Y ANÁLISIS DE RESULTADOS

3.1 INTRODUCCIÓN

3.1.1 REQUERIMIENTOS DE HARDWARE Y SOFTWARE

Todo el desarrollo se ha realizado teniendo en mente el uso primario de herramientas de código abierto o afines al espíritu ético de la libre difusión del conocimiento plasmados por la Fundación del Software Libre (*Free Software Fundation* – FSF) en su Licencia Pública General (*General Public License* – GPL).

Por tal motivo, el código implementado también ha sido declarado como código abierto bajo los términos de la GPL v.2, en aras de contribuir al desarrollo e investigación de la temática concerniente a la simulación de redes inalámbricas.

En las Tablas 3-1, 3-2 y 3-3 respectivamente, se presenta un resumen de los parámetros configurados para el hardware y de las aplicaciones instaladas en la máquina virtual, donde corre la simulación planteada en el presente proyecto.

HARDWARE	
Servidor	
Modelo	Proliant ML110 G7
Marca	HP
Microprocesador	Intel Xeon E31220 @ 3.10GHz (4 núcleos)
Memoria RAM	16 GB DDR3
Disco Duro	250 GB

Tabla 3-1 Resumen de los detalles de Hardware para el servidor que aloja la máquina virtual

SOFTWARE	
Máquina Virtual	Vmware Vsphere 5.5
Sistema Operativo	Ubuntu Desktop 14.04 LTS
Velocidad de Microprocesador	2.5 GHz
Memoria RAM	1000,8 MiB
Disco Duro	125.6 GB
Tipo de S.O.	32 bits

Tabla 3-2 Resumen de los detalles de software en la máquina virtual para la simulación (parte 1 de 2).

OMNeT++	
Versión	4.5
Número de Compilación	140714-c6b1772
CASTALIA	
Versión	3.2
JAVA	
Versión	Java Development Kit 1.8.0_91 (Oracle)
Número de Compilación	b14
Entorno de desarrollo para Java NETBEANS	
Versión	8.1 Patch 1
Número de Compilación	201510222201
BASH Shell	
Versión	4.3.11(1)-release

Tabla 3-3 Resumen de los detalles de software en la máquina virtual para la simulación (parte 2 de 2)

El alcance del proyecto define 18 nodos sensores a simularse, el cual es un número manejable para una sencilla computadora de escritorio, debido a que el número de iteraciones es lo suficientemente alto como para que no sea factible realizar resoluciones manuales pero que no requieren de un gran poder de cómputo con la tecnología actualmente a disposición de un estudiante promedio de ingeniería.

Esto se pudo comprobar a lo largo de la implementación y puesta a prueba del proyecto, donde se realizó cada uno de los subprocesos (que se explicarán en los ítems del subcapítulo 3.1.3.1), a través de una laptop marca Dell, modelo Vostro-1520 con 2 GB de memoria RAM y un procesador Core 2 Duo de 2.67 GHz. Además, se instaló el sistema operativo Ubuntu 12.04 LTS de 32 bits, sin mayores inconvenientes al momento de compilar y probar los subprocesos.

Se prefiere el uso de máquinas virtuales para poder revertir errores involuntarios con la ayuda de la herramienta de *snapshots* para realizar respaldos.

Sin embargo, cabe recalcar que, si se tratase de modificar el desarrollo implementado para simular centenares o miles de nodos sensores, (dependiendo del método de resolución del TSP elegido), sería necesario reevaluar los recursos propuestos para la máquina virtual en la Tabla 3-1 para lograr un desempeño que no afecte la correcta implementación de la simulación.

3.1.2 SIMULADOR

3.1.2.1 OMNeT++

OMNeT++ es un *framework*[34] y una librería[35] de simulación extensible, modular y basada en componentes C++, principalmente desarrollado para construir simuladores de redes, ya sean cableadas, inalámbricas, o de cualquier otro tipo [66] [67].

A su vez, OMNeT++ ofrece un entorno de desarrollo (Véase Figura 3-1) basado en el IDE[36] Eclipse, razón por la cual su manejo se facilita si ya se ha utilizado el IDE mencionado. Además, posee extensiones para la simulación en tiempo real, emulación de redes, integración de base de datos, integración de SystemC, y muchas otras funciones [66].

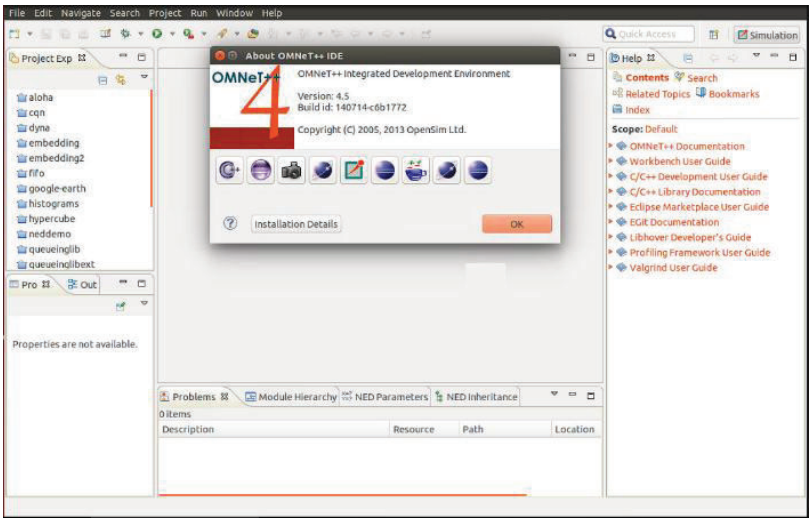


Figura 3-1 IDE de OMNeT++ versión 4.5, instalado en el servidor usado para el proyecto.

Los componentes, llamados en adelante "módulos", están programados en C++ y posteriormente se unen en componentes más grandes o modelos usando un lenguaje de alto nivel (NED - *Network Description File*). Gracias a la arquitectura

---

[34] Un *framework* es un conjunto de clases y métodos que sirven como "marco" para desarrollar aplicaciones. El *framework* gestiona el código desarrollado por el programador. [85] [86]

[35] Una librería es un conjunto de métodos y/o funciones invocados y controlados por el programador desde su código para simplificar tareas complejas con una simple llamada. [85] [86]

[36] *Integrated Develpment Enviroment* (Entorno de Desarrollo Integrado), es una aplicación diseñada para ayudar al desarrollador a construir sus soluciones de software.

modular de OMNeT++, el kernel de simulación (y modelos) pueden ser insertados fácilmente en aplicaciones propias [66].

OMNeT++ proporciona las herramientas básicas para realizar simulaciones, pero no se especializa en proporcionar un entorno para simulaciones específicas, tales como de redes de computadoras, simulación de colas, o simulaciones de arquitectura de sistemas o cualquier otra área [68].

Para ello, existen proyectos que se derivan de OMNeT++, tales como: INET Framework para redes alámbricas, inalámbricas y móviles, SimuLTE – Modelo de simulación a nivel de sistema LTE y simulador para INET & OMNeT++, OverSim para redes P2P, Veins para simulación de redes vehiculares y el simulador concerniente al presente trabajo, Castalia para WSN.

Actualmente, OMNeT++ se encuentra en su versión 5.0, publicada el 15 de Abril de 2016, lo que refleja es un proyecto activo y actual. Las características que el fabricante resalta respecto a las versiones 4.x son, cambios en el Canvas API (gráficos en 2D), soporte para gráficos en 3D basados en OpenSceneGraph, mejoras en el proceso de autenticación, creación de un nuevo entorno de ejecución basado en Qt[37] que eventualmente reemplazará a Tkenv[38], y eliminación de funcionalidades en desuso.

Al existir el peligro que alguna de las funcionalidades eliminadas tenga repercusiones directas o indirectas en la instalación, desarrollo e implementación de Castalia, se ha preferido usar la versión 4.5 de OMNeT++. Adicionalmente, se recomienda encarecidamente usar Qt si se piensa desarrollar una interfaz gráfica para el presente proyecto o para proyectos relacionados a OMNet++.

---

[37] De por sí no es un lenguaje de programación, más bien es un *framework* escrito en C++ para aplicaciones de escritorio, embebidas y móviles. Soporta varias plataformas (Linux, OS X, Windows, VxWorks, QNX, Android, iOS, BlackBerry, Sailfish OS) y principalmente se lo utiliza para implementar interfaces gráficas de usuario en C++ [87].

[38] Tkenv es una interfaz gráfica de usuario portátil desplegada en forma de “ventanas”. Tiene la capacidad de proporcionar una imagen detallada del estado de la simulación en cualquier momento durante su ejecución [88].

La experiencia de investigadores como Vikas Mainanwal de la Universidad Técnica de Punjab, que gentilmente ha compartido su experiencia instalando el OMNet++ en Ubuntu 14.04, ha sido tomada en cuenta para ejecutar los comandos descritos en [69], obteniendo resultados exitosos. También, cabe recalcar que siempre se debe instalar primero OMNet++ antes de instalar Castalia.

3.1.2.2 Castalia

Fue desarrollado por el NICTA (*National Information Communication Technology Australia*) para simular Redes Inalámbricas de Sensores (WSN), Redes de área corporal (BAN - *Body Area Network*) y en general de redes de dispositivos embebidos de baja potencia. Es utilizado por investigadores y desarrolladores para probar sus algoritmos distribuidos y protocolos en modelos de canal y radio inalámbricos realistas, con un comportamiento de nodo realista especialmente relacionado en lo que respecta al acceso al medio [70].

Castalia es una herramienta de código abierto genérica, es decir en ella no se puede probar código o hardware para replicarlo exactamente en un modelo de nodo sensor específico [71].

3.1.2.2.1 Estructura

Para obtener un comportamiento del nodo realista, Castalia simula todos los componentes de los nodos sensores reales (en el capítulo 1 se detallan los componentes). La figura 3-2 muestra la estructura interna del módulo nodo de Castalia [72].

En la Figura 3-2 las flechas de línea continua significan que existe un paso de mensajes, mientras las flechas punteadas significan que existe un simple llamado de funciones.

Por ejemplo, la mayoría de módulos llaman a una función del administrador de recursos para señalar que se ha consumido energía.

Módulo de Aplicación: Es el módulo que comúnmente el usuario puede cambiar, usualmente creando un nuevo módulo para implementar un nuevo algoritmo.

Módulos de Comunicación: MAC y Enrutamiento: Usualmente serán modificados por el usuario para implementar un nuevo protocolo.

Módulo Administrador de movilidad: El usuario generalmente modificará este módulo para crear un nuevo patrón de movilidad.

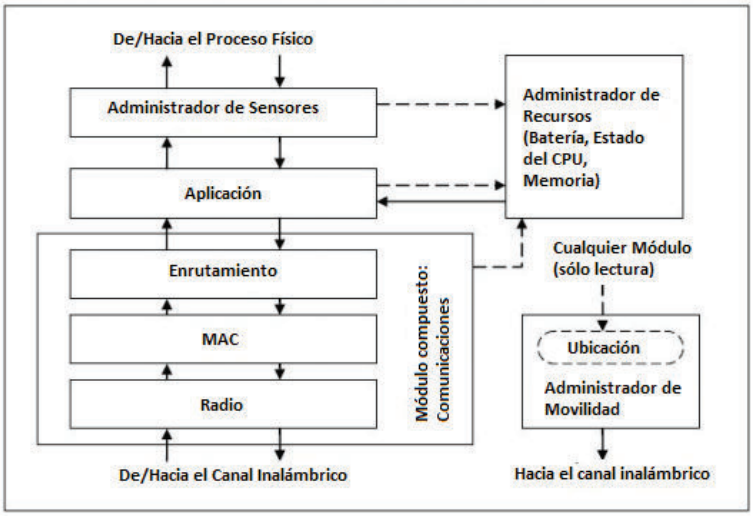


Figura 3-2 Módulos que conforman un nodo en el simulador Castalia, traducido de [71].

Para el proyecto, se realizó modificaciones en el módulo de aplicación, para los módulos ConnectivityMap y ThroughputTest, tal como se explica en 3.1.2.1.2 y 3.1.2.1.4, y se utilizó el módulo de radio para el chip CC2420[39], que utilizan los 3 modelos de mota propuestos. Además, para el resto de módulos se permiten los parámetros por defecto, como por ejemplo BypassMAC[40] y BypassRouting[41]

3.1.2.2.2 Desventajas

Algunas desventajas son que no usa el IDE (*Integrated Development Enviroment* – Entorno de Desarrollo Integrado) de OMNET++, solamente se puede utilizar a través del terminal, por lo que se recomienda la edición del código en las herramientas notepad++ o sublime text para evitar errores de sintaxis, además que cada simulación solamente puede tener una WSN [70].

---

[39] Es un chip sencillo que trabaja en la banda de los 2.4 GHz, utiliza un transceptor compatible con el estándar IEEE 802.15.4 para aplicaciones inalámbricas de baja potencia [89].
[40] Módulo de Castalia que no implementa ninguna tecnología de acceso al medio.
[41] Módulo de Castalia que no implementa ningún protocolo de enrutamiento.

3.1.2.2.3 Ventajas

En su tesis de maestría, Martín Stehlík, utiliza motas TmoteSky para comparar las simulaciones en Castalia, MiXiM[42] y TOSSIM[43] con una WSN real, los resultados que obtuvo Stehlík se resumen en la Tabla 3-4 [73]:

	CASTALIA	MIXIM	TOSSIM
Ensombrecimiento lognormal	Sí	Sí (usando la fórmula de Friis)	Sí (usando una herramienta externa)
Extensibilidad del modelo de canal inalámbrico	Es necesario cambiar el código fuente	Es posible crearlo externamente	Es necesario cambiar el código fuente o crear una herramienta externa
Modelamiento de ruido	Sí	Sí	Sí (usando un enfoque estadístico)
BER y PER	Técnica de modulación y SNR	Técnica de modulación y SNR	Empíricamente medido usando CC2420 y MicaZ
Documentación	Excelente	Pobre	Media
Rapidez de aprendizaje	Rápida	Lenta	Media
Ejemplos provistos	Suficientes	Suficientes	Medio
MAC 802.15.4	Parcialmente	Sí	Parcialmente
Soporte para el chip CC2420	Sí	Sí	Basado en mediciones
Soporte para simulación de consumo energético	Sí	Sí	No
Extensibilidad de su funcionalidad interna	Bueno	Bueno	Confuso
Tiempo de ejecución de la simulación	Rápido	Lento	Medio

Tabla 3-4 Resumen de la comparación entre los simuladores Castalia, MiXiM y TOSSIM (traducido de [73]).

[42] Framework de simulación para redes inalámbricas y móviles usando como motor de simulación OMNeT++ [90].
[43] Es el simulador del sistema operativo para WSN, TinyOS.

Gracias a la documentación en el manual de usuario del simulador [71] y al aporte de la comunidad en los foros de Research Gate [74] y Google Groups [75], se puede modificar el código fuente del simulador sin ser un experto en C++ (como es el caso del autor de esta tesis). Aunque cabe recalcar, que sería necesario un conocimiento más profundo del lenguaje C++ para realizar simulaciones que requieran mayor complejidad y/o personalización.

3.1.3 ESQUEMA GENERAL DE LA SIMULACIÓN

La simulación implementada, se resume en la Figura 3-3:

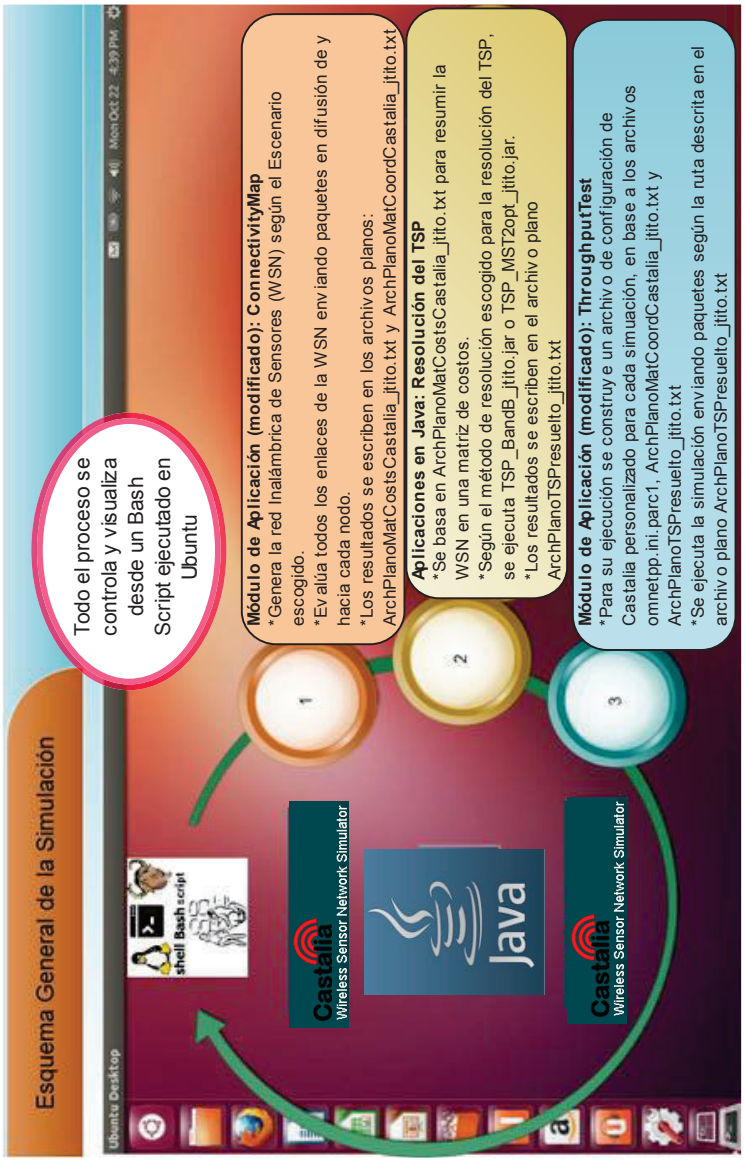


Figura 3-3 Esquema general de la simulación, mostrado a través de cuatro subprocesos.

3.1.3.1 Subprocesos

3.1.3.1.1 Subproceso de control: Shell Bash Script

A través de un *Bash Shell script* se controla la ejecución de cada uno de los subprocesos implementados, de tal manera que se despliega en el Terminal de comandos de Ubuntu los resultados arrojados por cada subproceso gestionado.

Se usó del código publicado en [76] para la creación archivos y directorios con fecha, hora, minuto y segundo, además para el control de flujo se consultó el código publicado en [77].

El script fue diseñado de tal manera que, el usuario solo seleccione el número de la opción deseada y presione la tecla *enter*, previniendo los errores de digitación por parte del usuario y prescindiendo del uso de código adicional para controlar los errores de usuario.

A continuación, se muestran en orden cronológico los menús disponibles al ejecutar el script. Una vez terminada la simulación, el script vuelve a la pantalla inicial y solo termina si el usuario escoge el numeral correspondiente a la opción “salir”.

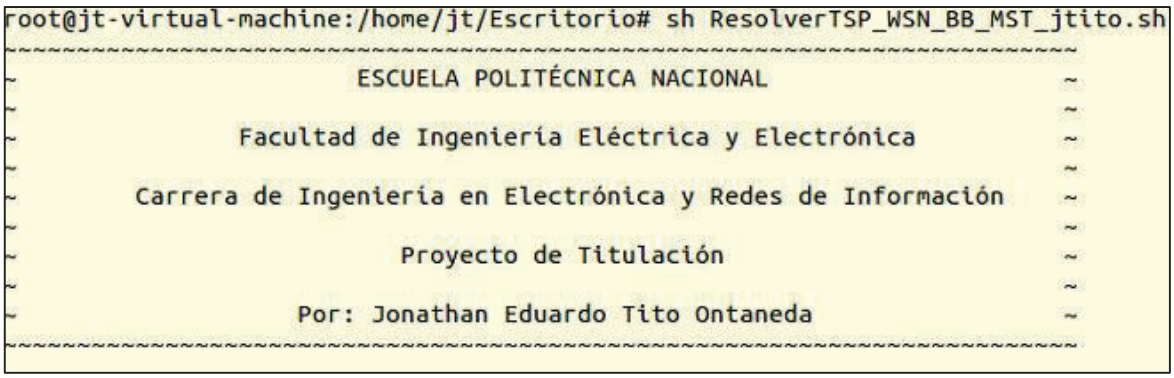


Figura 3-4 Comando para correr el script ResolverTSP_WSN_BB_MST_jtito.sh y pantalla de presentación del script.

En la primera línea de la Figura 3-4 se puede observar el comando con el que se ejecuta el *bash script*, que para el presente trabajo se ha decidido llamarlo “ResolverTSP_WSN_BB_MST_jtito.sh”; con el fin de que tan solo con el nombre del archivo ya se obtenga una idea de lo que se está ejecutando.

```
--------------------------------------------------------------------------------
********************************************************************************
El script ejecutado, resuelve el TSP en una WSN.
Para ello se debe escoger entre,
-Dos Escenarios: Escenario A y Escenario B.
-Tres modelos de mota: TelosB, Imote2 y Zolertia.
-Dos métodos de resolución del TSP: Branch and Bound y Aprox. por MST.
********************************************************************************
--------------------------------------------------------------------------------
```

Figura 3-5 Pantalla de indicaciones para ejecutar la simulación.

```
***OPCIONES DE ESCENARIO***

1) Escenario A: Los nodos son colocados uniformemente y distribuidos de manera aleatoria,
con el nodo fuente ubicado en el extremo izquierdo del escenario de simulación

2) Escenario B: Los nodos son colocados uniformemente y distribuidos de manera aleatoria,
con el nodo fuente ubicado en el centro del escenario de simulación

3) SALIR

Escriba el número de la opción deseada y presione la tecla enter.
(Si se equivoca, volverá al inicio del script).
```

Figura 3-6 Pantalla de elección del Escenario de simulación.

```
Usted ha escogido el Escenario B.
--------------------------------------------------------------------------------
********************************************************************************
El script ejecutado, resuelve el TSP en una WSN.
Para ello se debe escoger entre,
-Dos Escenarios: Escenario A y Escenario B.
-Tres modelos de mota: TelosB, Imote2 y Zolertia.
-Dos métodos de resolución del TSP: Branch and Bound y Aprox. por MST.
********************************************************************************
--------------------------------------------------------------------------------

***OPCIONES DE MOTA***

1) TelosB

2) Imote2

3) Zolertia

4) SALIR

Escriba el número de la opción deseada y presione la tecla enter
(Si se equivoca, volverá al inicio del script).
```

Figura 3-7 Pantalla de elección del modelo de mota a simularse.

A continuación del primer contenido mostrado en pantalla (Véase Figura 3-4) se despliega una descripción más detallada del script, como se puede observar en la

Figura 3-5, mientras las Figuras 3-6 y 3-7 muestran los menús de opciones de escenario y de modelo de mota.

```
Usted ha escogido la mota Zolertia.
En el EscenarioB
--------------------------------------------------------------------------------
********************************************************************************
El script ejecutado, resuelve el TSP en una WSN.
Para ello se debe escoger entre,
-Dos Escenarios: Escenario A y Escenario B.
-Tres modelos de mota: TelosB, Imote2 y Zolertia.
-Dos métodos de resolución del TSP: Branch and Bound y Aprox. por MST.
********************************************************************************
--------------------------------------------------------------------------------

***OPCIONES DE RESOLUCIÓN DEL TSP***

1) Método de Branch and Bound usando el límite inferior de Held - Karp

2) Método de Aproximación por el Minimum Spanning Tree usando 2opt

3) SALIR

Escriba el número de la opción deseada y presione la tecla enter
(Si se equivoca, volverá al inicio del script).
```

Figura 3-8 Pantalla de elección del método de resolución del TSP a simularse.

```
Usted ha escogido el método de resolución por Branch and Bound usando el límite inferior de Held - Karp.
En el EscenarioB
Con la mota Z1
La simulación se ejecutará 10 veces.
1 de 10 repeticiones
Se crea una carpeta con la fecha y hora de la simulación a ejecutarse en /home/jt/Escritorio/ResultadosTesisjtito/EscenarioB/Z1/BB
En la primera parte de la simulación se procede  evaluar los enlaces para construir el grafo
Running configuration 1/1
```

Figura 3-9 Pantalla de simulación para el primer subproceso: ConnectivityMap.

```
Usted ha escogido el método de resolución por Branch and Bound usando el límite inferior de Held - Karp.
En el EscenarioB
Con la mota Z1
La simulación se ejecutará 10 veces.
1 de 10 repeticiones
Se crea una carpeta con la fecha y hora de la simulación a ejecutarse en /home/jt/Escritorio/ResultadosTesisjtito/EscenarioB/Z1/BB
En la primera parte de la simulación se procede  evaluar los enlaces para construir el grafo
Running configuration 1/1
Se resuleve el TSP
El mejor límite inferior hasta el momento es: 46.0
El mejor límite inferior hasta el momento es: 18.0
Tour TSP Resuelto:
ID del nodo/vértice: 0
ID del nodo/vértice: 1
ID del nodo/vértice: 13
ID del nodo/vértice: 11
ID del nodo/vértice: 3
ID del nodo/vértice: 10
ID del nodo/vértice: 4
ID del nodo/vértice: 5
ID del nodo/vértice: 7
ID del nodo/vértice: 8
ID del nodo/vértice: 15
ID del nodo/vértice: 9
ID del nodo/vértice: 16
ID del nodo/vértice: 14
ID del nodo/vértice: 2
ID del nodo/vértice: 17
ID del nodo/vértice: 12
ID del nodo/vértice: 6
ID del nodo/vértice: 0
```

Figura 3-10 Termina el primer subproceso e inicia el segundo subproceso que resuelve el TSP en java.

Como se puede observar en la Figura 3-8, la primera línea muestra información de la opción escogida en el anterior menú al actual.

Además en la Figura 3-9 se muestra información sobre el número de repeticiones de la simulación y sobre la ejecución del primer subproceso de simulación (ConnectivityMap), la Figura 3-10 muestra información sobre el subproceso de resolución del TSP.

```
Usted ha escogido el método de resolución por Branch and Bound usando el límite inferior de Held - Karp.
En el EscenarioB
Con la nota Z1
La simulación se ejecutará 10 veces.
1 de 10 repeticiones
Se crea una carpeta con la fecha y hora de la simulación a ejecutarse en /home/jt/Escritorio/ResultadosTesisJtito/EscenarioB/Z1/BB
En la primera parte de la simulación se procede  evaluar los enlaces para construir el grafo
Running configuration 1/1
Se resuleve el TSP
El mejor límite inferior hasta el momento es: 46.0
El mejor límite inferior hasta el momento es: 58.0
Tour TSP Resuelto:
ID del nodo/vértice: 0
ID del nodo/vértice: 1
ID del nodo/vértice: 13
ID del nodo/vértice: 11
ID del nodo/vértice: 3
ID del nodo/vértice: 10
ID del nodo/vértice: 4
ID del nodo/vértice: 5
ID del nodo/vértice: 7
ID del nodo/vértice: 8
ID del nodo/vértice: 15
ID del nodo/vértice: 9
ID del nodo/vértice: 16
ID del nodo/vértice: 14
ID del nodo/vértice: 2
ID del nodo/vértice: 17
ID del nodo/vértice: 12
ID del nodo/vértice: 6
ID del nodo/vértice: 0
Simula el TSP resuelto en la WSN planteada
Running configuration 1/1
```

Figura 3-11 Termina el segundo subproceso e inicia el tercer y último subproceso, ThroughputTest

```
2 de 10 repeticiones
Se crea una carpeta con la fecha y hora de la simulación a ejecutarse en /home/jt/Escritorio/ResultadosTesisJtito/EscenarioB/Z1/BB
En la primera parte de la simulación se procede  evaluar los enlaces para construir el grafo
Running configuration 1/1
```

Figura 3-12 Al terminar la simulación se vuelven a ejecutar los subprocesos hasta completar 10 repeticiones y vuelve a la pantalla de inicio del script

El proceso de los dos últimos subprocesos de simulación que observa el usuario, se muestran en la Figura 3-11. Así mismo, se tuvo en cuenta el uso de funciones para evitar la redundancia de código en el script y además se controla que cada escenario de simulación se repita diez veces (véase Figura 3-12 y Espacio de Código 3-1), antes de regresar a la pantalla Inicial.

```
168 RepetirSim10Veces () {
169 echo "La simulación se ejecutará 10 veces."
170 x=1
171 while [ $x -le 10 ]
172 do
173         echo "$x de 10 repeticiones"
174         SimularEnCastaliaE1
175         x=$(( $x + 1 ))
176 done
177 }
```

Espacio de Código 3-1 Código de la función RepetirSim10Veces(), para controlar la ejecución de 10 repeticiones de la simulación.

```
52 DespliegaOpcionesEscenario () {
53         echo "***OPCIONES DE ESCENARIO***"
54         echo ""
55         echo "1) Escenario A: Los nodos son colocados uniformemente y distribuidos de manera
   aleatoria,"
56         echo "con el nodo fuente ubicado en el extremo izquierdo del escenario de simulación"
57         echo ""
58         echo "2) Escenario B: Los nodos son colocados uniformemente y distribuidos de manera
   aleatoria,"
59         echo "con el nodo fuente ubicado en el centro del escenario de simulación"
60         echo ""
61         echo "3) SALIR"
62         echo ""
63         echo "Escriba el número de la opción deseada y presione la tecla enter."
64         echo "(Si se equivoca, volverá al inicio del script)."
65         echo  ""
66 }
67 DespliegaOpcionesMota () {
68         echo "***OPCIONES DE MOTA***"
69         echo ""
70         echo "1) TelosB"
71         echo ""
72         echo "2) Imote2"
73         echo ""
74         echo "3) Zolertia"
75         echo ""
76         echo "4) SALIR"
77         echo ""
78         echo "Escriba el número de la opción deseada y presione la tecla enter"
79         echo "(Si se equivoca, volverá al inicio del script)."
80         echo  ""
81 }
```

Espacio de Código 3-2 Funciones DespliegaOpcionesEscenario() y DespliegaOpcionesMota(), como ejemplo de las funciones que despliegan información para el usuario.

En los Espacios de Código 3-2 y 3-3 se muestra parte de la captura, validación y control de las opciones ingresadas por el usuario se lo realiza a través de funciones que implementan control de flujo mediante *case* y llamadas a las funciones pertinentes para la correcta ejecución de la simulación

```
125 EscogerMetodoResolvrTSP () {
126 retMetTSP=""
127 DespliegaResumenScript
128 DespliegaOpcionesReslvrTSP
129 read retMetTSP
130 strRslvrTSP=""
131 case $retMetTSP in
132             1 ) tput reset;
133                 echo "Usted ha escogido el método de resolución por Branch and Bound usando el límite inferior de Held - Karp."
134                 strRslvrTSP="BB"
135                 ImprimirEscenario
136                 ImprimirMota
137                 RepetirSim10Veces
138                 ;;
139             2 ) tput reset;
140                 echo "Usted ha escogido el método de resolución aproximado, usando Minimum Spanning Tree"
141                 strRslvrTSP="MST"
142                 ImprimirEscenario
143                 ImprimirMota
144                 RepetirSim10Veces
145                 ;;
146             3 ) tput reset;
147                 echo "Gracias por correr este script. Adiós."
148                 exit
149                 ;;
150             * ) tput reset;
151                 echo "Opción Incorrecta: Por favor, intente de nuevo."
152                 ;;
153         esac
154 }
```

Espacio de Código 3-3 Función *EscogerMetodoResolvrTSP()*, como ejemplo de las funciones que controlan la interacción con el usuario.

La simulación en Castalia se controla a través de dos métodos, en el cual una primera etapa agrupa el subproceso ConnectivityMap y el subproceso de resolución del TSP. Mientras el subproceso ThroughputTest se lo controla en una segunda etapa.

```
SimularEnCastaliaE1 () {
dirArchPlanosEtapa1="/home/jt/Escritorio/SW-jt/castalia3_2/Castalia-3.2/Simulations/connectivityMap"
dirArchPlanosEtapa2="/home/jt/Escritorio/SW-jt/castalia3_2/Castalia-3.2/Simulations/throughputTest_jtito"
cat /dev/null > "$dirArchPlanosEtapa1"/ArchPlanoMatCostsCastalia_jtito.txt
cat /dev/null > "$dirArchPlanosEtapa1"/ArchPlanoCoordCastalia_jtito.txt
cat /dev/null > "$dirArchPlanosEtapa2"/ArchPlanoTSPresuelto_jtito.txt
echo "Se crea una carpeta con la fecha y hora de la simulación a ejecutarse en /home/jt/Escritorio/ResultadosTesisjtito/
   $strEscenario/$strMota/$strRslvrTSP"
dirResSim=/home/jt/Escritorio/ResultadosTesisjtito/"$strEscenario"/"$strMota"/"$strRslvrTSP"/"$(date "+%d.%m.%Y-%H.%M.%S")"
mkdir "$dirResSim"
echo 'En la primera parte de la simulación se procede  evaluar los enlaces para construir el grafo'
cd /home/jt/Escritorio/SW-jt/castalia3_2/Castalia-3.2/Simulations/connectivityMap/;
tiempoSim=$(date "+%d.%m.%Y-%H.%M.%S")
Castalia -c General,"$strEscenario","$strMota" -o "$dirResSim"/ArchResulCrudosCastaliaGrafo_jtito_"$tiempoSim".txt


CastaliaResults -i "$dirResSim"/ArchResulCrudosCastaliaGrafo_jtito_"$tiempoSim".txt -s "received" -n | CastaliaPlot -o
   "$dirResSim"/rxvedPktWSN_stacked_Grafo_"$tiempoSim".png -s stacked --
   colors=#000000,#ff0000,#00c000,#c8c800,#0080ff,#c000ff,#ffa500,#f03232,#40e0d0,#c0c0c0,#c04000,#7fffd4,#f0c68c,#ee82ee,#801414,#ffc
   --xtitle="Nodos" --ytitle="# de paquetes recibidos exitosamente" -l outside
```

Espacio de Código 3-4 Fragmento del código de la función *SimularEnCastaliaE1()*, que implementa la lógica de los dos primeros subprocesos en una primera etapa de la simulación.

```
SimularEnCastaliaE2 () {
echo "Simula el TSP resuelto en la WSN planteada"
cat /dev/null > "$dirArchPlanosEtapa2"/omnetpp.ini
cat "$dirArchPlanosEtapa2"/omnetpp.ini.parc1 >> "$dirArchPlanosEtapa2"/omnetpp.ini
cat "$dirArchPlanosEtapa1"/ArchPlanoCoordCastalia_jtito.txt >> "$dirArchPlanosEtapa2"/omnetpp.ini
cat "$dirArchPlanosEtapa2"/ArchPlanoTSPresuelto_jtito.txt >> "$dirArchPlanosEtapa2"/omnetpp.ini
cd "$dirArchPlanosEtapa2"
Castalia -c General,ResolverTSP,"$strMota" -o "$dirResSim"/ArchResulCrudosCastaliaTSP_jtito_"$tiempoSim".txt


CastaliaResults -i "$dirResSim"/ArchResulCrudosCastaliaTSP_jtito_"$tiempoSim".txt -s "TXed pkts" -n | CastaliaPlot -o
"$dirResSim"/txedPktWSN_stacked_TSP_"$tiempoSim".png -s stacked --
colors=#000000,#ff0000,#00c000,#c8c800,#0080ff,#c000ff,#ffa500,#f03232,#40e0d0,#c0c0c0,#c04000,#7fffd4,#f0c68c,#ee82ee,#801414,#ffc
--xtitle="Nodos" --ytitle="# de paquetes transmitidos" -l outside

CastaliaResults -i "$dirResSim"/ArchResulCrudosCastaliaTSP_jtito_"$tiempoSim".txt -s "TXed pkts" -n | CastaliaPlot -o
"$dirResSim"/txedPktWSN_linespoints_TSP_"$tiempoSim".png -s linespoints --
colors=#000000,#ff0000,#00c000,#c8c800,#0080ff,#c000ff,#ffa500,#f03232,#40e0d0,#c0c0c0,#c04000,#7fffd4,#f0c68c,#ee82ee,#801414,#ffc
--xtitle="Nodos" --ytitle="# de paquetes transmitidos" -l outside
```

Espacio de Código 3-5 Fragmento del código de la función *SimularEnCastaliaE2()*, que implementa la lógica del tercer subproceso en una segunda etapa de la simulación.

Con el objetivo presentar ordenadamente los resultados de la simulación, se generan archivos y carpetas con la fecha, hora, minuto y segundo de la ejecución de la simulación.

Adicionalmente, la lógica de la programación fue implementada al principio del código de las funciones SimularEnCastaliaE1() y SimularEnCastaliaE2(), como se puede apreciar en el Espacio de Código 3-4 y en el Espacio de Código 3-5. En estos dos espacios de código se puede apreciar los comandos del simulador.

Todos los archivos planos resultantes de la simulación se encuentran bajo carpetas con el nombre del escenario, modelo de mota y método de resolución del TSP

respectivamente, esto se realizó con el objetivo de mantener ordenados los resultados de la simulación, tal como se puede apreciar en las Figuras 3-13 y 3-14:

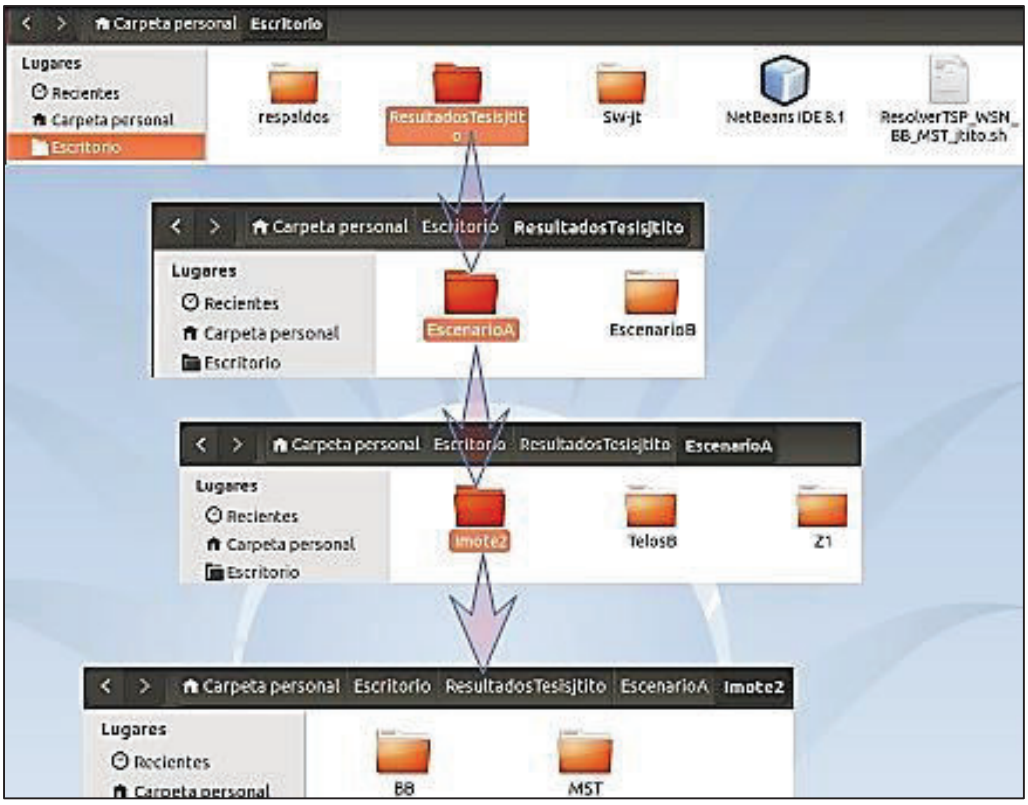


Figura 3-13 Jerarquía de las carpetas previamente creadas para mantener ordenados los archivos de las simulaciones ejecutadas.

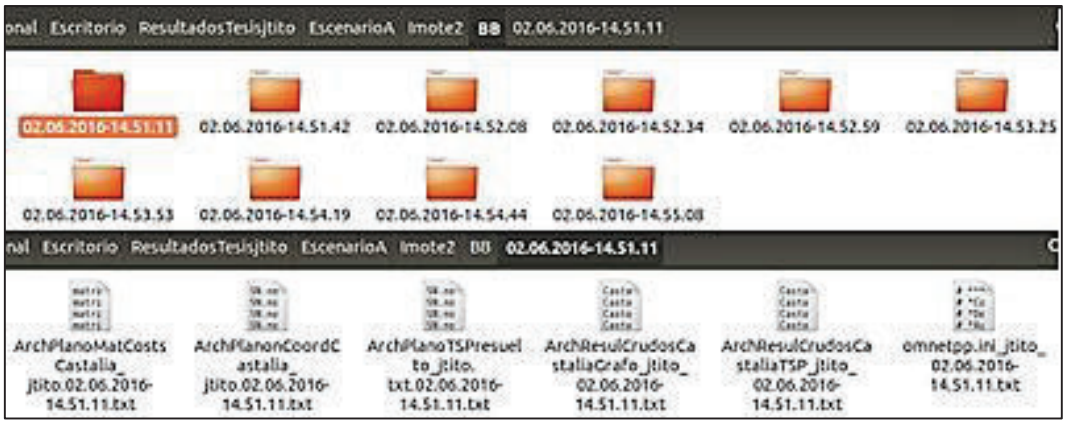


Figura 3-14 Carpetas y archivos de simulación creados en tiempo de ejecución, bajo la carpeta del método de resolución del TSP

Los detalles de la programación del *script* para *Bash Shell* se encuentran en el ANEXO 1, perteneciente al presente proyecto de titulación.

3.1.3.1.2 Primer subproceso de la simulación: ConnectivityMap (modificado),

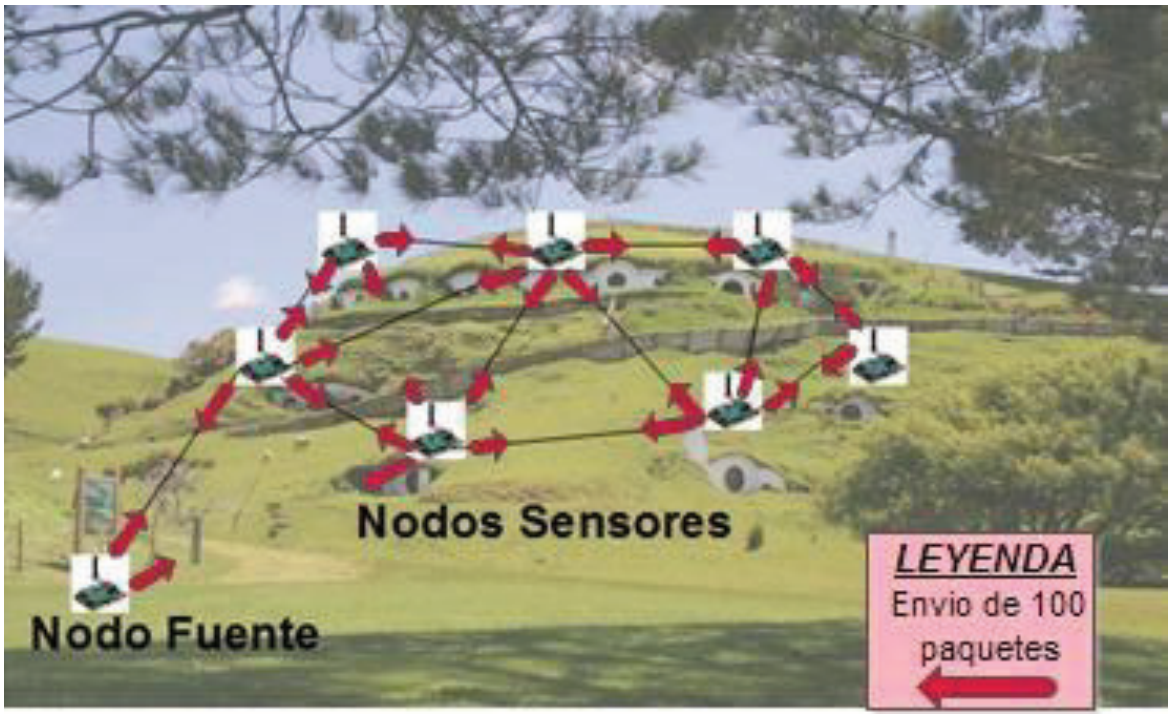


Figura 3-15 Ilustración didáctica de la medición de la calidad del enlace, difundiendo 100 paquetes entre todos los nodos de la WSN.

La aplicación de Castalia ConnectivityMap, está diseñada según el Dr. Boulis [71], para encontrar la calidad de los enlaces entre los nodos de la WSN. Por defecto cada nodo está programado para transmitir 100 paquetes en un único intervalo de tiempo (véase Figura 3-15), de tal manera que no existan colisiones con otros nodos. Un nodo (cuando no está transmitiendo) constantemente escucha los paquetes entrantes, y cuando un paquete es recibido por el nodo en cuestión, este incrementa el contador de los paquetes recibidos identificando el nodo transmisor.

3.1.3.1.2.1 Nota importante

Como se puede observar en la Figura 3-15, existen casos en los que llegan cero paquetes al nodo destino y esto se traduce como un enlace inexistente entre el par de nodos comunicantes. Al definirse el TSP en un grafo completo (como se expuso en el capítulo 1), se debe buscar una WSN con el mínimo número de enlaces inexistentes para que su grafo correspondiente sea completo y el TSP pueda ser resuelto.

Cuando la simulación finaliza, los contadores se escriben como salida de la aplicación usando el índice del nodo transmisor, tal como se puede apreciar en la Figura 3-16:

```
 1 Castalia| what:General,EscenarioB,Imote2 (1)
 2 Castalia| when:2016-06-03 03:48
 3 Castalia| label:General,EscenarioB,Imote2
 4 Castalia|       module:SN.node[0].ResourceManager
 5 Castalia|               simple output name:Consumed Energy
 6 Castalia|                       15213
 7 Castalia|       module:SN.node[0].Communication.Radio
 8 Castalia|               simple output name:RX pkt breakdown
 9 Castalia|                       89 Failed with NO interference
10 Castalia|                       1611 Received with NO interference
11 Castalia|               simple output name:TXed pkts
12 Castalia|                       100 TX pkts
13 Castalia|       module:SN.node[0].Application
14 Castalia|                index:1 simple output name:Packets received
15 Castalia|                       100 Success
16 Castalia|                index:2 simple output name:Packets received
17 Castalia|                       100 Success
18 Castalia|                index:3 simple output name:Packets received
19 Castalia|                       100 Success
20 Castalia|                index:4 simple output name:Packets received
21 Castalia|                       100 Success
22 Castalia|                index:5 simple output name:Packets received
23 Castalia|                       100 Success
24 Castalia|                index:6 simple output name:Packets received
25 Castalia|                       99 Success
26 Castalia|                index:7 simple output name:Packets received
27 Castalia|                       100 Success
28 Castalia|                index:8 simple output name:Packets received
29 Castalia|                       100 Success
30 Castalia|                index:9 simple output name:Packets received
31 Castalia|                       100 Success
32 Castalia|                index:10 simple output name:Packets received
33 Castalia|                       82 Success
34 Castalia|                index:11 simple output name:Packets received
35 Castalia|                       100 Success
36 Castalia|                index:12 simple output name:Packets received
37 Castalia|                       100 Success
38 Castalia|                index:13 simple output name:Packets received
```

Figura 3-16 Fragmento del archivo ArchResulCrudosCastaliaGrafo_jtito_fechayhora.txt, resultante del primer subproceso dela simulación.

Sería dificultoso trabajar con el formato de salida de Castalia, si se necesita extraer información específica de la simulación, como lo es el caso de recibir paquetes exitosamente por el nodo “A” y enviados por el nodo “B”. Con el comando CastaliaResults, que viene incluido en el simulador se puede obtener los resultados solo en pantalla (véase Figura 3-17) y en un formato menos complejo, por lo tanto es difícil tratar de tomar estos resultados desplegados en pantalla y trabajar con ellos

```
jt@jt-virtual-machine:~/Escritorio/ResultadosTesisjtito/EscenarioB/Imote2/BB/03.06.2016-03.48.05$ CastaliaResults -i ArchResulCrudosCa
staliaGrafo_jtito_03.06.2016-03.48.05.txt -s received -n -o 2


Application:Packets received - Success
 | node=0 | node=1 | node=2 | node=3 | node=4 | node=5 | node=6 | node=7 | node=8 | node=9 | node=10 | node=11 | node=12 | node=13 | n
ode=14 | node=15 | node=16 | node=17
index=0 | 0 | 100 | 99 | 99 | 100 | 100 | 100 | 99 | 99 | 100 | 73 | 100 | 100 | 6 | 78 | 99 | 100 | 100
index=1 | 100 | 0 | 100 | 100 | 100 | 100 | 100 | 100 | 99 | 100 | 67 | 99 | 100 | 100 | 100 | 100 | 100 | 100
index=2 | 100 | 100 | 0 | 100 | 97 | 100 | 0 | 100 | 100 | 100 | 100 | 100 | 100 | 52 | 100 | 100 | 100 | 100
index=3 | 100 | 100 | 99 | 0 | 100 | 97 | 100 | 100 | 100 | 100 | 99 | 100 | 100 | 65 | 100 | 100 | 100 | 100
index=4 | 100 | 100 | 100 | 100 | 0 | 100 | 0 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100
index=5 | 100 | 100 | 100 | 99 | 100 | 0 | 76 | 100 | 99 | 100 | 0 | 100 | 0 | 100 | 0 | 96 | 100 | 62
index=6 | 99 | 100 | 0 | 100 | 0 | 8 | 0 | 69 | 100 | 12 | 0 | 100 | 100 | 0 | 0 | 78 | 100 | 100
index=7 | 100 | 100 | 100 | 100 | 100 | 100 | 97 | 0 | 100 | 100 | 0 | 1 | 100 | 4 | 1 | 100 | 100 | 100
index=8 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 0 | 100 | 100 | 1 | 98 | 100 | 72 | 91 | 100 | 100
index=9 | 100 | 100 | 100 | 100 | 100 | 100 | 22 | 100 | 100 | 0 | 95 | 98 | 100 | 100 | 100 | 100 | 100 | 100
index=10 | 82 | 94 | 100 | 100 | 100 | 0 | 0 | 0 | 0 | 100 | 80 | 0 | 0 | 0 | 100 | 97 | 82 | 0 | 0
index=11 | 100 | 98 | 99 | 100 | 100 | 100 | 100 | 30 | 0 | 99 | 0 | 0 | 47 | 100 | 1 | 1 | 100 | 100 | 100
index=12 | 100 | 100 | 100 | 100 | 100 | 4 | 100 | 100 | 79 | 100 | 13 | 51 | 0 | 9 | 95 | 0 | 100 | 100
index=13 | 50 | 100 | 100 | 7 | 100 | 100 | 0 | 1 | 100 | 100 | 100 | 100 | 49 | 0 | 0 | 0 | 100 | 0
index=14 | 83 | 100 | 100 | 100 | 100 | 0 | 0 | 0 | 69 | 100 | 94 | 8 | 100 | 0 | 0 | 100 | 100 | 0
index=15 | 97 | 100 | 100 | 100 | 100 | 67 | 78 | 100 | 100 | 100 | 73 | 100 | 3 | 0 | 100 | 0 | 100 | 100
index=16 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 0 | 100 | 100 | 100 | 100 | 100 | 0 | 100
index=17 | 100 | 100 | 100 | 100 | 100 | 32 | 100 | 100 | 100 | 100 | 0 | 100 | 100 | 0 | 0 | 100 | 100 | 0
```

Figura 3-17 Resultados del primer subproceso de simulación obtenidos con la herramienta CastaliaResults, para detalles de su uso refiérase a [71].

Por los motivos expuestos anteriormente, se ha planteado la creación de los archivos planos:

- ArchPlanoMatCostsCastalia_jtito.txt y
- ArchPlanoCoordCastalia_jtito.txt,

Estos archivos planos se obtienen mediante la modificación del código fuente de Castalia y su re-compilación, tal como se ilustra en los Espacios de Código 3-6, 3-7 y 3.8:

```
105 void ConnectivityMap::finishSpecific()
106 {
107
108 int nodeID = atoi(SELF_NETWORK_ADDRESS);
109         declareOutput("Packets received");
110         for (int i = 0; i < (int)neighborTable.size(); i++)
111         {
112                 collectOutput("Packets received", neighborTable[i].id,
113                                 "Success", neighborTable[i].receivedPackets);
114                 //jt.1.ln Guardo en la matriz de costos el número de paquetes recibidos exitosamente
115                 mc[neighborTable[i].id][nodeID]=neighborTable[i].receivedPackets;
116
117         }
```

Espacio de Código 3-6 Código de ConnectivityMap.cc modificado para almacenar la cantidad de paquetes recibidos por cada nodo en la matriz *mc*

En el Espacio de Código 3-6 se puede observar cómo se declara en la línea 109, que se presentará información sobre los paquetes recibidos como resultado de la simulación. Este resultado se lo puede observar en los archivos ArchResulCrudosCastaliaGrafo_jtito_fechayhora.txt, Castalia-Trace.txt o de manera

más legible en la pantalla del terminal del sistema operativo, haciendo uso de la herramienta CastaliaResults.

```
121                         if(nodeID == 17) //Si la simulación llegó al último nodo de la WSN
122                         {
123                         for (int i = 0; i < 18;i++)
124                                 {
125                                         for (int j = 0; j < 18;j++)
126                                         {
127                                                 ofstream myfile;
128                                                 myfile.open
    ("ArchPlanoMatCostsCastalia_jtito.txt", std::ofstream::out | std::ofstream::app);
129                                                 myfile << "matriz["<<i<<"]["<<j<<"]="<<mc[i]
    [j]<<endl; //Escribo la matriz de costos, con los paqts. rx.
    exitosamente
130                                                 myfile.close();
131                                 }
132
133                 }
134                                                 for(int a=0; a < 18; a++)
135                                                 {
136                                                 //A su vez voy obteniendo las coordenadas de
    los nodos WSN a los que se puede acceder en cada iteración
137                                                 VirtualMobilityManager *nodeMobilityModule =
    check_and_cast<VirtualMobilityManager*>(getParentModule()->getParentModule()->getSubmodule
    ("node",a)->getSubmodule("MobilityManager"));
138                                                 x[a] = nodeMobilityModule->getLocation().x;
139                                                 y[a] = nodeMobilityModule->getLocation().y;
140
141                                                 ofstream myfilec;
142                                                 myfilec.open
    ("ArchPlanoCoordCastalia_jtito.txt", std::ofstream::out | std::ofstream::app);
143                                                 //myfilec << "x["<<a<<"]=" << x[a] <<",y["
    << a<< "]=" << y[a]<<endl;
144                                                 myfilec << "SN.node[" << a << "].xcoord = "
    << x[a] << endl;
145                                                 myfilec << "SN.node[" << a << "].ycoord = "
    << y[a] << endl;
146                                                 myfilec.close();
147
148                                                 }
149                         }
150         //jt.1.end.new
151 }
```

Espacio de Código 3-7 Código de ConnectivityMap.cc modificado para escribir los archivos planos con las coordenadas de los nodos sensores y con los paquetes recibidos satisfactoriamente por cada nodo (parte 1 de 2.)

```
138                                                 x[a] = nodeMobilityModule->getLocation().x;
139                                                 y[a] = nodeMobilityModule->getLocation().y;
140
141                                                 ofstream myfilec;
142                                                 myfilec.open ("ArchPlanonCoordCastalia_jtito.txt", std::ofstream::out |
    std::ofstream::app);
143                                                 //myfilec << "x["<<a<<"]=" << x[a] <<",y[" << a<< "]=" << y[a]<<endl;
144                                                 myfilec << "SN.node[" << a << "].xcoord = " << x[a]     << endl;
145                                                 myfilec << "SN.node[" << a << "].ycoord = " << y[a]     << endl;
146                                                 myfilec.close();
147
148                                                 }
149                         }
```

Espacio de Código 3-8 Código de ConnectivityMap.cc modificado para escribir los archivos planos con las coordenadas de los nodos sensores y con los paquetes recibidos satisfactoriamente por cada nodo (parte 2 de 2).

En los Espacios de Código 3-7 y 3-8 se implementa el código para escribir archivos planos personalizados utilizando C++ en el simulador Castalia. Se realiza la escritura del archivo plano en el mismo formato del archivo omnetpp.ini

Además, es necesario seguir el siguiente proceso de re-compilación del simulador mostrado en Figura 3-18 para que los cambios al código fuente del simulador Castalia surtan efecto:

```
jt@jt-virtual-machine:~/Escritorio/SW-jt/castalia3_2/Castalia-3.2$ ./makemake
Creating Makefile in /home/jt/Escritorio/SW-jt/castalia3_2/Castalia-3.2...
Makefile created, running "make depend" to add dependencies...
Creating dependencies...
jt@jt-virtual-machine:~/Escritorio/SW-jt/castalia3_2/Castalia-3.2$ make
Creating executable: out/gcc-release//CastaliaBin
jt@jt-virtual-machine:~/Escritorio/SW-jt/castalia3_2/Castalia-3.2$
```

Figura 3-18 Proceso de re-compilación del simulador Castalia cuando se ha modificado su código fuente.

A continuación, se muestra el fragmento de código del *script* que ordena la ejecución del primer subproceso de la simulación.

```
211 SimularEnCastaliaE1 () {
212 dirArchPlanosEtapa1="/home/jt/Escritorio/SW-jt/castalia3_2/Castalia-3.2/Simulations/connectivityMap"
213 dirArchPlanosEtapa2="/home/jt/Escritorio/SW-jt/castalia3_2/Castalia-3.2/Simulations/throughputTest_jtito"
214 cat /dev/null > "$dirArchPlanosEtapa1"/ArchPlanoMatCostsCastalia_jtito.txt
215 cat /dev/null > "$dirArchPlanosEtapa1"/ArchPlanoCoordCastalia_jtito.txt
216 cat /dev/null > "$dirArchPlanosEtapa2"/ArchPlanoTSPresuelto_jtito.txt
217 echo "Se crea una carpeta con la fecha y hora de la simulación a ejecutarse en /home/jt/Escritorio/ResultadosTesisjtito/
    $strEscenario/$strMota/$strRslvrTSP"
218 dirResSim=/home/jt/Escritorio/ResultadosTesisjtito/"$strEscenario"/"$strMota"/"$strRslvrTSP"/"$(date "+%d.%m.%Y-%H.%M.%S")"
219 mkdir "$dirResSim"
220 echo 'En la primera parte de la simulación se procede  evaluar los enlaces para construir el grafo'
221 cd /home/jt/Escritorio/SW-jt/castalia3_2/Castalia-3.2/Simulations/connectivityMap/;
222 tiempoSim=$(date "+%d.%m.%Y-%H.%M.%S")
223 Castalia -c General,"$strEscenario","$strMota" -o "$dirResSim"/ArchResulCrudosCastaliaGrafo_jtito_"$tiempoSim".txt
```

Espacio de Código 3-9 Fragmento de código de la función SimularEnCastaliaE1(), en la línea 223 se puede apreciar el comando que ejecuta el primer subproceso de la simulación.

La opción –c (mostrada en la línea 223 del Espacio de Código 3-9) permite escoger la(s) configuración(es) descritas en el archivo de configuración de Castalia.

Por defecto el nombre del archivo corresponde a omnetpp.ini, el cual posee los parámetros iniciales para comenzar la simulación. Cuando se usa el archivo de configuración de la simulación con su nombre por defecto, no hace falta especificar el nombre del archivo.

El archivo de resultados de la simulación se especifica con la opción –o, tal como se puede apreciar en el Espacio de Código 3-9. En el Espacio de Código 3-10, se presenta la estructura del archivo de configuración usado en el primer subproceso de

la simulación. Para mayor detalle del archivo, se recomienda revisar el Anexo 2, pertenecientes al presente proyecto de titulación.

```
#::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::#
#               Configuración General de la Simulación                         #
#::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::#

[General]

# ========================================================#
# Siempre incluir el archivo principal Castalia.ini       #
# ========================================================#
include ../Parameters/Castalia.ini

# ========================================================#
# Duración de la simulación                               #
# ========================================================#
sim-time-limit = 9000000s

# ========================================================#
# Tamaño del área de simulación                           #
# ========================================================#
SN.field_x = 50                         # metros
SN.field_y = 50                     # metros


# ========================================================#
# Número de nodos para la simulación                      #
# ========================================================#
SN.numNodes = 18

# ========================================================#
# Parámetros del canal inalámbrico                        #
# ========================================================#

###===Modelo de la Atenuación (path loss) Promedio usando el modelo de ensombrecimiento lognormal===###
SN.wirelessChannel.PLd0 = 55 #Atenuación en dB a una distancia d0, por defecto
SN.wirelessChannel.d0 = 1 #Distancia d0, por defecto
SN.wirelessChannel.sigma = 4 #Efecto del ensombrecimiento,por defecto

SN.wirelessChannel.pathLossExponent = 2.4 #Exponente de atenuación, por defecto
```

Espacio de Código 3-10 Fragmento del archivo de configuración del primer subproceso de la simulación en Castalia.

3.1.3.1.2.2 Nota Importante

Como consecuencia de lo enunciado en 3.1.2.1.2.1, y en mutuo acuerdo con el director de este proyecto de titulación, se tomó la decisión de cambiar el área de simulación propuesta de 200 metros de largo por 100 metros de alto a 50 metros de largo por 50 metros de ancho (Como se puede observar en el Espacio de Código 3-10, líneas 50 y 51).

El cambio es necesario debido a que, luego de someter a diversas pruebas los programas que implementan los dos métodos propuestos para resolver el TSP, se observó que ambos algoritmos tomaban enlaces inexistentes para completar el tour de mínimo costo. Según lo expuesto en

el capítulo uno del presente proyecto, un enlace inexistente puede ser reemplazado por un enlace de costo infinito (en la matriz de costos), con el objetivo de que el grafo que representa la WSN, sea un grafo completo y lo más conexo posible.

En la Figura 3-19 se muestra un grafo no muy conexo en el que existe una fase en la resolución del TSP para ambos algoritmos, donde se llega a un punto en el que no se puede escoger otro camino más que el enlace infinito, a causa de que se debe cumplir con la condición de que el tour de mínimo costo pase una sola vez por cada nodo de la WSN. Además, se debe cumplir con la condición de pasar por todos los nodos antes de regresar al nodo fuente.

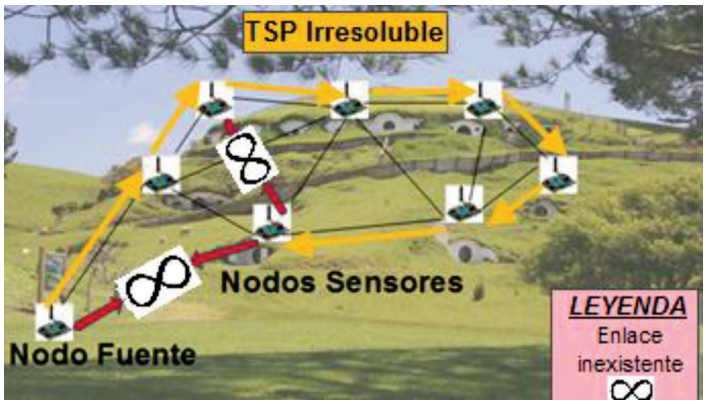


Figura 3-19 Ilustración didáctica de un TSP Irresoluble.

Como resultado del primer subproceso de simulación, se presentan a continuación fragmentos de los archivos planos obtenidos en las Figuras 3-20 y 3-21:

```
 1 matriz[0][0]=0
 2 matriz[0][1]=100
 3 matriz[0][2]=99
 4 matriz[0][3]=99
 5 matriz[0][4]=100
 6 matriz[0][5]=0
 7 matriz[0][6]=100
 8 matriz[0][7]=99
 9 matriz[0][8]=99
10 matriz[0][9]=100
11 matriz[0][10]=98
12 matriz[0][11]=100
13 matriz[0][12]=91
14 matriz[0][13]=0
15 matriz[0][14]=100
```

Figura 3-20 Fragmento del archivo plano ArchPlanoMatCostsCastalia_jtito. (fecha-hora-minuto-segundo).txt que contiene los paquetes recibidos satisfactoriamente por el nodo j desde el nodo i.

```
 1 SN.node[0].xcoord = 27.4407
 2 SN.node[0].ycoord = 29.6422
 3 SN.node[1].xcoord = 42.2133
 4 SN.node[1].ycoord = 30.1382
 5 SN.node[2].xcoord = 27.2442
 6 SN.node[2].ycoord = 42.3626
 7 SN.node[3].xcoord = 31.1782
 8 SN.node[3].ycoord = 32.2947
 9 SN.node[4].xcoord = 21.8794
10 SN.node[4].ycoord = 14.8767
11 SN.node[5].xcoord = 2.83565
12 SN.node[5].ycoord = 48.1831
13 SN.node[6].xcoord = 19.1721
14 SN.node[6].ycoord = 23.8833
15 SN.node[7].xcoord = 40.6084
16 SN.node[7].ycoord = 26.4447
17 SN.node[8].xcoord = 28.4022
18 SN.node[8].ycoord = 19.6392
19 SN.node[9].xcoord = 41.8039
20 SN.node[9].ycoord = 3.5518
21 SN.node[10].xcoord = 4.35646
```

Figura 3-21 Fragmento del archivo plano ArchPlanoCoordCastalia_jtito. (fecha-hora-minuto-segundo).txt en el formato del simulador Castalia, que contiene las coordenadas de cada nodo simulado.

3.1.3.1.3 Segundo subproceso de la simulación: Aplicaciones en Java, resolución del TSP

Ambas aplicaciones toman el archivo plano ArchPlanoMatCostsCastalia_jtito.txt, para convertir el número de paquetes recibidos exitosamente desde el nodo “i” al nodo “j”, en el costo del enlace.

Según el método escogido para resolver el TSP, se ejecuta el archivo TSP_BandB_jtito.jar o el archivo TSP_MST2opt_jtito.jar. El resultado de cualquiera de los dos métodos, se escribe en el archivo plano ArchPlanoTSPresuelto_jtito.txt, como se observa en la Figura 3-22.

ArchPlanoTSPresuelt...06.2016-14.51.11.txt ×

```
 1 SN.node[0].Application.nextRecipient = "17"
 2 SN.node[17].Application.nextRecipient = "2"
 3 SN.node[2].Application.nextRecipient = "1"
 4 SN.node[1].Application.nextRecipient = "5"
 5 SN.node[5].Application.nextRecipient = "16"
 6 SN.node[16].Application.nextRecipient = "13"
 7 SN.node[13].Application.nextRecipient = "11"
 8 SN.node[11].Application.nextRecipient = "3"
 9 SN.node[3].Application.nextRecipient = "15"
10 SN.node[15].Application.nextRecipient = "10"
11 SN.node[10].Application.nextRecipient = "14"
12 SN.node[14].Application.nextRecipient = "9"
13 SN.node[9].Application.nextRecipient = "4"
14 SN.node[4].Application.nextRecipient = "7"
15 SN.node[7].Application.nextRecipient = "12"
16 SN.node[12].Application.nextRecipient = "8"
17 SN.node[8].Application.nextRecipient = "6"
18 SN.node[6].Application.nextRecipient = "0"
```

Figura 3-22 Captura del archivo plano ArchPlanoTSPresuelto_jtito.(fecha-hora-minuto-segundo).txt, resultado de la resolución del TSP por cualquiera de los dos métodos en formato Castalia.

```
249
250
251 CastaliaResults -i "$dirResSim"/ArchResulCrudosCastaliaGrafo_jtito_"$tiempoSim".txt -s "RX" -n -o 2   | cat > "$dirResSim"/
    rxFWSN_excel_Grafo_"$tiempoSim".txt
252
253
254 cp "$dirArchPlanosEtapa1"/ArchPlanoMatCostsCastalia_jtito.txt "$dirResSim"/ArchPlanoMatCostsCastalia_jtito."$tiempoSim".txt
255 cp "$dirArchPlanosEtapa1"/ArchPlanoCoordCastalia_jtito.txt "$dirResSim"/ArchPlanoCoordCastalia_jtito."$tiempoSim".txt
256 dirBBTSP="/home/jt/Escritorio/SW-jt/castalia3_2/Castalia-3.2/Simulations/throughputTest_jtito/CODIGOJAVAFINAL/TSP_BandB_jtito/
    dist/TSP_BandB_jtito.jar"
257 dirMSTTSP="/home/jt/Escritorio/SW-jt/castalia3_2/Castalia-3.2/Simulations/throughputTest_jtito/CODIGOJAVAFINAL/TSP_MST2opt_jtito/
    dist/TSP_MST2opt_jtito.jar"
258 echo 'Se resuleve el TSP'
259
260 if [ $retMetTSP = 1 ];
261 then
262 java -jar "$dirBBTSP"
263 cp "$dirArchPlanosEtapa2"/ArchPlanoTSPresuelto_jtito.txt "$dirResSim"/ArchPlanoTSPresuelto_jtito.txt."$tiempoSim".txt
264 SimularEnCastaliaE2
265 fi
266
267 if [ $retMetTSP = 2 ];
268 then
269 java -jar "$dirMSTTSP"
270 cp "$dirArchPlanosEtapa2"/ArchPlanoTSPresuelto_jtito.txt "$dirResSim"/ArchPlanoTSPresuelto_jtito.txt."$tiempoSim".txt
271 SimularEnCastaliaE2
272 fi
```

Espacio de Código 3-11 Fragmento de código del Bash Shell script que ejecuta el método de resolución del TSP escogido.

En el Espacio de código 3-11, se puede observar las sentencias que ejecutan cada uno de los códigos desarrollados en el lenguaje Java para resolver el TSP, según la opción escogida por teclado.

Detalles del código implementado para ambos métodos de resolución del TSP pueden encontrarse en Anexo 4 y Anexo 5, mientras que una explicación más detallada de la lógica de la programación, se complementa en los ítems 3.3 y 3.5.

3.1.3.1.4 Tercer subproceso de la simulación: ThroughputTest (modificado)

A diferencia del primer módulo de simulación, para ThroughputTest no se modificó el código fuente del simulador, pero si se modifica en tiempo real el archivo de configuración de Castalia, omnetpp.ini. Por tal motivo, se ha decidido crear una carpeta adicional en la carpeta de simulaciones de Castalia, la cual tiene por nombre ThroughputTest_jtito, tal como se puede apreciar en la Figura 3-23:

Figura 3-23 Carpeta ThroughputTest_jtito, creada para albergar los archivos del tercer y último subproceso de la simulación.

La modificación del archivo omnetpp.ini se realiza de tal manera que, se conservan la mayoría de los parámetros establecidos en el archivo omnetpp.ini del primer subproceso de la simulación.

Los parámetros de los escenarios A y B, son eliminados para dar paso a los parámetros de las coordenadas de cada uno de los nodos sensores, obtenidos en el archivo ArchPlanoCoordCastalia_jtito.txt.

Además, se anexa una especie de tabla de enrutamiento estática, mediante el uso de la información del archivo plano ArchPlanoTSPresuelto_jtito.txt.

Para lograr la creación del archivo de configuración de Castalia omnetpp.ini en tiempo real se hace uso de 3 archivos mostrados en la Figura 3-24:

- **ArchPlanoCoordCastalia_jtito.txt***:* Contiene las coordenadas en los ejes “x” y “y”, en el formato de Castalia, según el escenario elegido para la simulación.
- **ArchPlanoTSPresuelto_jtito.txt***:* Contiene una tabla de enrutamiento estática en el formato de Castalia, para enviar paquetes según el orden del TSP resuelto.
- **Omnetpp.ini.parc1**.- Contiene una copia modificada del archivo omnetpp.ini del primer subproceso de la simulación, donde se ha eliminado la configuración de los escenarios A y B, y se deja vacía la configuración ResolverTSP, para luego ser completada con la información contenida en los dos archivos planos anteriormente mencionados. Además, el módulo de aplicación obviamente cambia de ConnectivityMap a ThroughputTest.

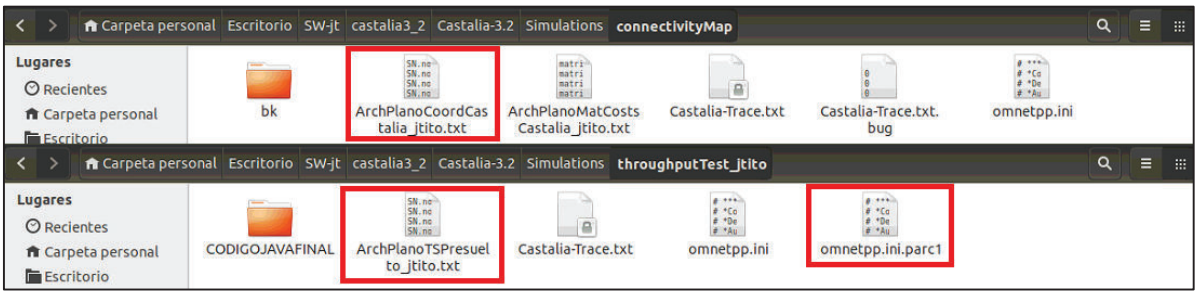


Figura 3-24 Captura de pantalla de los tres archivos planos con los que se crea el archivo omnetpp.ini para el tercer subproceso de la simulación: ThroughputTest

El *bash sell script* se encarga de realizar la unión de los archivos planos mencionados, usando el comando *cat*, propio de las distribuciones GNU/Linux.

```
158 SN.node[*].ResourceManager.initialEnergy = 10142 #Joules (2 baterías AAA)
159
160 [Config ResolverTSP]
161 # ================================================================================#
162 # Parámetros para simular la resolución del TSP                                   #
163 # ================================================================================#
164
165 SN.node[0].xcoord = 27.4407
166 SN.node[0].ycoord = 29.6422
167 SN.node[1].xcoord = 42.2133
168 SN.node[1].ycoord = 30.1382
169 SN.node[2].xcoord = 27.2442
170 SN.node[2].ycoord = 42.3626
171 SN.node[3].xcoord = 31.1782
172 SN.node[3].ycoord = 32.2947
173 SN.node[4].xcoord = 21.8794
174 SN.node[4].ycoord = 14.8767
175 SN.node[5].xcoord = 2.83565
176 SN.node[5].ycoord = 48.1831
```

Espacio de Código 3-12 Fragmentos del archivo omnetpp.ini (final) para el tercer subproceso de la simulación (parte 1 de 2).

```
197 SN.node[16].xcoord = 5.91372
198 SN.node[16].ycoord = 36.0316
199 SN.node[17].xcoord = 29.101
200 SN.node[17].ycoord = 7.16766
201 SN.node[0].Application.nextRecipient = "17"
202 SN.node[17].Application.nextRecipient = "2"
203 SN.node[2].Application.nextRecipient = "1"
204 SN.node[1].Application.nextRecipient = "5"
205 SN.node[5].Application.nextRecipient = "16"
206 SN.node[16].Application.nextRecipient = "13"
207 SN.node[13].Application.nextRecipient = "11"
208 SN.node[11].Application.nextRecipient = "3"
209 SN.node[3].Application.nextRecipient = "15"
210 SN.node[15].Application.nextRecipient = "10"
211 SN.node[10].Application.nextRecipient = "14"
```

Espacio de Código 3-13 Fragmentos del archivo omnetpp.ini (final) para el tercer subproceso de la simulación (parte 2 de 2).

Los archivos planos mencionados, pueden observarse a detalle en Anexo 7 y Anexo 8. Siendo el archivo de configuración (véase los Espacios de Código 3-12 y 3-13), el encargado de describir el escenario de la simulación y los detalles de cada uno de los módulos, al modificar dinámicamente el archivo omnetpp.ini se está modificando la simulación dinámicamente.

Gracias a estas modificaciones dinámicas se podría simular escenarios en los que se posea los datos del movimiento de las motas con anterioridad, de tal manera que en cada iteración se modificarían las coordenadas de las motas.

3.2 IMPLEMENTACION DE LA RESOLUCIÓN DEL TSP POR EL MÉTODO DE APROXIMACIÓN POR EL MST EN UNA WSN

La implementación del método de resolución del TSP por el algoritmo de *Branch and Bound* aplicando el límite inferior de Held-Karp, se resume en el siguiente diagrama de clases UML[44], conseguido a través del *plugin* EasyUML para Netbeans 8.1.

El desarrollo se basa en el código libre publicado en el foro de desarrolladores *Stack Overflow* [64] y en *Google Code* [63]. Para una mejor comprensión del código implementado se ha utilizado nombres descriptivos para todas las variables y métodos del desarrollo. A continuación, se detallan las partes más representativas de la aplicación y para un mayor detalle por favor revisar el Anexo 5.

```
/**
 * SimpTSPSolBBHK
 * Copyright (C) 2016 Jonathan Eduardo Tito Ontaneda
 *
 * SimpTSPSolBBHK is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the
 * Free Software Foundation, either version 3 of the License, or (at your option)
 * any later version.
 * SimpTSPSolBBHK is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
 * details.
 * You should have received a copy of the GNU General Public License along
 * with SimpTSPSolBBHK. If not, see http://www.gnu.org/licenses/
```

Espacio de Código 3-14 Aplicación SimpTSPSolBBHK: Al utilizarse código abierto, se realizan modificaciones sobre el código y se lo libera bajo los mismos términos.

```
57      * Declaración de variables.
58      */
59         //número de vértices/nodos WSN
60         private static int n;
61         //número de paquetes enviados por cada vértices/nodo WSN
62         private static int nPaqEnv = 100;
63         //matriz de costos
64         private static double[][] matCostos;
65         //matriz de costos que toma en cuenta el peso del vértices/nodo WSN
66         //según Held -Karp
67         private double[][] matCostosConPesosVertcs;
68         //matriz de costos del archivo plano generado por Castalia
69         private static double[][] matAuxCastalia;
70         //línea de texto leida del archivo plano generado por Castalia
71         static String cadnCastalia;
72         //índice del signo igual en el archivo plano generado por Castalia
73         static int indcSignIgual = 0;
74         //índice del corchete abierto en el archivo plano generado por Castalia
75         static int indcSignCorhtAbrt = 0;
76         //valor de la matriz de paquetes recibidos, según Castalia
77         static String val_mat="";
78         //valor del índice de la fila en el archivo plano generado por Castalia
79         static String val_fil="";
80         //valor del índice de la columna en el archivo plano generado por Castalia
81         static String val_col="";
82         //valor de la fila y la columna (juntos)
83         //en el archivo plano generado por Castalia
84         static String fil_col="";
85         //resultado de la resolución del TSP
86         static List<String> lstTSPresuelto = new ArrayList<>();
```

Espacio de Código 3-15 Aplicación SimpTSPSolBBHK: Declaración de variables globales.

[44] El Lenguaje de Modelamiento Unificado (UML - Unified Modeling Language) es un lenguaje gráfico utilizado para diseñar y desarrollar software [92].

Las modificaciones realizadas deberán liberarse de acuerdo a los términos de la licencia GPL tal como se enuncia en el Espacio de Código 3-14. Como se puede observar en el Espacio de Código 3-15, se ha comentado todo el código para que el lector lo pueda entender de mejor manera y de ser el caso, lo modifique a su conveniencia.

```
99      public static void main(String[] args)
100     {
101         //se obtiene la matriz de costos de Castalia
102         matCostos = obtnMatCostCastalia();
103
104         //Se instancia de la clase principal
105         TSPBBHK tsp_BBHK = new TSPBBHK();
106         //Se resuelve el TSP usando B&B y el límite inferior de H-K
107         tsp_BBHK.resolverTSP();
108         //Se muestran los resultados de la resolución del TSP
109         tsp_BBHK.mostrarResultadoTSP();
110     }
```

Espacio de Código 3-16 Aplicación SimpTSPSolBBHK: Método main con el que inicia la ejecución del programa.

En el Espacio de Código 3-16 se muestra el método *main* donde se realiza un llamado al método obtnMatCostCastalia(), que lee el archivo plano ArchPlanoMatCostsCastalia_jtito.txt (véase Espacio de Código 3-17). Además, convierte los valores leídos como cantidad de paquetes recibidos exitosamente al valor para el costo del enlace (si no llegan paquetes, el enlace se toma por “enlace infinito”), y finalmente devuelve la matriz de costos para almacenarla en la variable matCostos.

```
137             try (Scanner scnArchCast = new Scanner(archCastalia))
138             {
139                 while (scnArchCast.hasNextLine())
140                 {
141                     //Se obtiene la información necesaria de cada línea
142                     //del archivo plano generado por Castalia
143                     cadnCastalia = scnArchCast.nextLine();
144                     indcSignIgual =  cadnCastalia.indexOf("=");
145                     val_mat = cadnCastalia.substring(indcSignIgual+1);
146                     indcSignCorhtAbrt =  cadnCastalia.indexOf("[");
147                     fil_col = cadnCastalia.substring(indcSignCorhtAbrt+1,
148                             indcSignIgual-1);
149                     val_fil = fil_col.substring(0,fil_col.indexOf("]"));
150                     val_col = fil_col.substring(fil_col.indexOf("[")+1);
151                     int fil = Integer.parseInt(val_fil);
152                     int col = Integer.parseInt(val_col);
153                     int val = Integer.parseInt(val_mat);
154
155                     //val contiene el número de paquetes exitosos que llegaron
156                     //del nodo [fil] al nodo [col], y sus valores
157                     //son inversamente proporcionales al costo del enlace,
158                     //por lo tanto se realiza una conversión.
```

Espacio de Código 3-17 Aplicación SimpTSPSolBBHK: Fragmento del código del método obtnMatCostCastalia() *(parte 1 de 2)*

En la línea 172 del Espacio de Código 3-18 se puede apreciar que al no existir la definición de infinito en el lenguaje de programación Java, se ha optado por usar el máximo valor posible para las variables de tipo Double, con el objetivo de expresar un enlace inexistente entre un par de nodos.

```
159                     if (val != 0)
160                     {
161                         //Se invierte la escala: a más paquetes recibidos
162                         //menor es el costo del enlace
163                         int valTmpEscala = convNuevaEscala(1,
164                                 nPaqEnv,nPaqEnv,1,val);
165                         val = valTmpEscala;
166                     }
167                     else
168                     {
169                         //Al no llegar paquetes, el costo del enlace
170                         //debe ser el más alto posible,para que el
171                         //grafo se torne lo más completo posible
172                         Double valTmpCero = Double.MAX_VALUE;
173                         val = valTmpCero.intValue();
174                     }
175
176                     matAuxCastalia[fil][col]= val;
177
178                 }
179             }
180         }
181         catch (Exception e)
182         {
183             System.out.println(Arrays.toString(e.getStackTrace()));
184         }
185
186         return matAuxCastalia;
187     }
```

Espacio de Código 3-18 Aplicación SimpTSPSolBBHK: Fragmento del código del método obtnMatCostCastalia() (parte 2 de 2)

La subclase SolParcBB mostrada en el Espacio de Código 3-19, representa la contenedora de los datos auxiliares para implementar el método de B&B.

```
class SolParcBB {
public boolean[][] matrNlacsExcldsSolParc;
//variables útiles para el límite inferior de Held - Karp
public double[] vctrPesosVrtcs;
public double limtInferior;
public int[] vtorGrdsVrtcs;
public int[] vtorDeValAIndc;
  }
```

Espacio de Código 3-19 Aplicación SimpTSPSolBBHK: Definición de la subclase SolParcBB, que representa las soluciones parciales generadas por el método de B&B

También se utiliza el máximo valor de las variables de tipo Double al inicio de la implementación del algoritmo, tal como se menciona en 2.2.1.3.1.

```
278     public void resolverTSP()
279     {
280         //Al principio del algoritmo, el mejor límite inferior es el valor
281         //más alto posible para poder comenzar a trabajar
282         mejorSolParcBB.limtInferior = Double.MAX_VALUE; //Ver 2.2.1.3.1 en [1]
283         //Se instancia un nodo B&B para evaluar soluciones viables del TSP,
284         //sin asignar variables por el momento
285         SolParcBB solParclBBevaluada = new SolParcBB();
286         //Matriz de enlaces WSN excluidos en la solución evaluada
287         solParclBBevaluada.matrNlacsExcldsSolParc = new boolean[n][n];
288         //Inicializo la matriz de costos que incluye el peso de los vértices/nodos
289         matCostosConPesosVertcs = new double[n][n];
290         //Se obtiene el límite inferior usando el método de Held - Karp
291         //para el nodo B&B evaluado al momento
292         resolverHK(solParclBBevaluada);
```

Espacio de Código 3-20 Aplicación SimpTSPSolBBHK: Fragmento del código del método resolverTSP(), donde se inicializa la clase de Soluciones Parciales B&B y se llama al método resolverHK().

El método ResolverTSP() es el siguiente invocado en el método main(), desde una instancia de la clase principal, tal como se puede apreciar en el Espacio de Código 3-16 (línea 107 del código). A su vez, el método en cuestión realizará llamadas a otros métodos que le servirán para obtener el TSP resuelto, como en el Espacio de Código 3-20 (línea 292), donde se invoca al método que implementa el algoritmo de Held-Karp.

```
418   private void resolverHK(SolParcBB nodoBBparaHK) {
419     nodoBBparaHK.vctrPesosVrtcs = new double[n];
420     nodoBBparaHK.limtInferior = Double.MIN_VALUE;
421     nodoBBparaHK.vtorGrdsVrtcs = new int[n];
422     nodoBBparaHK.vtorDeValAIndc = new int[n];
423     //Para comprender los cálculos a continuación
424     //se debe revisar la ecuación 2-19 en [1]
425     //y su explicación
426     double lambda = 0.1;
427     while (lambda > 1e-06) {
428       double previousLowerBound = nodoBBparaHK.limtInferior;
429       resolverOneTree(nodoBBparaHK);
430       if (!(nodoBBparaHK.limtInferior < mejorSolParcBB.limtInferior)) return;
431       if (!(nodoBBparaHK.limtInferior < previousLowerBound)) lambda *= 0.9;
432       int denom = 0;
433       for (int i = 1; i < n; i++) {
434         int dMenos2 = nodoBBparaHK.vtorGrdsVrtcs[i] - 2;
435         denom += Math.pow(2,dMenos2);
436       }
437       if (denom == 0) return;
438       double t = lambda * nodoBBparaHK.limtInferior / denom;
439       for (int i = 1; i < n; i++){
440           nodoBBparaHK.vctrPesosVrtcs[i] +=
441                   t * (nodoBBparaHK.vtorGrdsVrtcs[i] - 2);
442       }
443
444     }
445   }
```

Espacio de Código 3-21 Aplicación SimpTSPSolBBHK: Método que implementa el algoritmo de Held – Karp para obtener el límite inferior mínimo.

En el Espacio de Código 3-21 se implementa la Ecuación 2-25, explicada en el capítulo 2 del presente proyecto de titulación. De dicho proceso, se desprende la obtención del 1-tree que se logra a través del llamado al método resolverOneTree(), tal como se observa en el Espacio de Código 3-22.

```
447     private void resolverOneTree(SolParcBB hojaOT)
448     {
449         // Se calcula la matriz de costos dandole costos a los vértices/nodos
450         hojaOT.limtInferior = 0.0;
451         Arrays.fill(hojaOT.vtorGrdsVrtcs, 0);
452         for (int fil = 0; fil < n; fil++)
453         {
454             for (int col = 0; col < n; col++)
455             {
456                 matCostosConPesosVertcs[fil][col] =
457                     hojaOT.matrNlacsExcldsSolParc[fil][col] ? Double.MAX_VALUE :
458                     matCostos[fil][col] +
459                     hojaOT.vctrPesosVrtcs[fil] +
460                     hojaOT.vctrPesosVrtcs[col];
461             }           //mayor  =    (x>y)      ?        x         :          y;
462         }
```

Espacio de Código 3-22 Aplicación SimpTSPSolBBHK: Fragmento del método resolverOneTree() que implementa la lógica para obtener el 1-tree.

```
292             resolverHK(solParclBBevaluada);
293             //Comparo los limites inferiores de los nodos B&B para
294             //ordenarlos descendentemente, según el valor de H-K obtenido
295             PriorityQueue<SolParcBB> pilaDescLimInfSolParclsBB =
                        new PriorityQueue<SolParcBB>(11, new ComparadorSolParcBB());
297             //Mientras el nodo B&B evaluado no sea una solución completa
298             //y su limite inferior obtenido con H-K sea al menos tan bueno como
299             //el mejor límite inferior
300             do {
301                 //Mientras el límite inferior más bajo del nodo B&B evaluado sea
302                 //menor que el mejor límite inferior hallado
303                 do
304                 {
305                     //Vértice WSN auxiliar (se numeran del 0 al "n")
306                     int vrtcAux = -1;
307                     //Se recorre los vértices/nodos WSN
308                     for (int vrtcEvldo = 0; vrtcEvldo < n; vrtcEvldo++)
```

Espacio de Código 3-23 Aplicación SimpTSPSolBBHK: Fragmento del método resolverTSP(), luego de obtener el límite inferior de Held-Karp se evalúan las soluciones parciales en una cola de prioridades (PriorityQueue).

En el Espacio de Código 3-22 se observa el proceso de exclusión de enlaces para crear soluciones parciales producto de la implementación de *Branch and Bound*. De nuevo se utiliza el máximo valor de *Double* para expresar un costo infinito.

Posteriormente, como se observa en el Espacio de Código 3-23 se obtiene el mejor límite inferior hasta el momento y se lo imprime en cada iteración hasta el momento en que se halla el menor posible o hasta que se determina que el TSP es irresoluble.

```
329                     if (solParclBBevaluada.limtInferior
330                             < mejorSolParcBB.limtInferior)
331                     {
332                         //De ser así se actualiza el valor del mejor límite
333                         //inferior obtenido hasta el momento.
334                         mejorSolParcBB = solParclBBevaluada;
335                         //Se imprime el mejor límite inferior.
336                         System.out.println("El mejor límite inferior hasta"
337                                 + " el momento es: "
338                         + mejorSolParcBB.limtInferior);
339
340                         //Si el mejor límite inferior es "infinito"
341                         //el grafo no tuvo suficientes caminos para
342                         //resolver el TSP, y el programa se vió
343                         //obligado a optar por un camino de costo "infinito"
344                         //para completar el TSP.
345                         int infAux = Integer.MAX_VALUE;
346                         if(mejorSolParcBB.limtInferior >= infAux)
347                         {
348                             //Por lo tanto se lanza una alerta
349                             //System.out.println("NO TIENE SOLUCIÓN"
350                             JOptionPane.showMessageDialog(null,
351                                     "LIMITE INFERIOR INFINITO",
352                                     "TSP IRRESOLUBLE",
353                                     JOptionPane.WARNING_MESSAGE);
354                             //Y por último se termina el programa
355                             System.exit(0);
```

Espacio de Código 3-24 Aplicación SimpTSPSolBBHK: Fragmento de código del método resolverTSP(), que busca el mejor límite inferior y a su vez valora si el TSP tiene solución.

```
238     public void mostrarResultadoTSP()
239     {
240         try
241         {
242             //Se guarda la solución del TSP en un archivo plano, separando con
243             //comas los ID de los vértices/nodos WSN debidamente ordenados.
244
245             File archTSPresult =
246                     new File("/home/jt/Escritorio/SW-jt/castalia3_2/Castalia-3.2/Simulations"
247                             + "/throughputTest_jtito/ArchPlanoTSPresuelto_jtito.txt");
248             //File archTSPresult =
249                     //new File("D:\\TESIS TEMPO\\UN MES\\tspbbjavafinal"
250                             //+ "\\ArchPlanoTSPresuelto_jtito.txt");
251
252             try (PrintWriter prntWrtrEscrbrResult =
253                     new PrintWriter(archTSPresult))
254             {
255                 /*
256                 lstTSPresuelto.stream().forEach((item) ->
257                     prntWrtrEscrbrResult.append(item+",");
258                 });
259                 */
260                 int i = 0;
261                 while(i < lstTSPresuelto.size())
262                 {
263                     if(i > 0)
264                     {
265                         prntWrtrEscrbrResult.println("SN.node["+lstTSPresuelto.get(i-1)+"].Application.nextRecipient =
266                     }
267                     i++;
```

Espacio de Código 3-25 Aplicación SimpTSPSolBBHK: Fragmento de código del método mostrarResultadoTSP(), que imprime el TSP en el archivo plano ArchPlanoTSPresuelto_jtito.txt usando el formato del simulador.

En el Espacio de Código 3-24 y en el Espacio de Código 3-25, respectivamente, se observa la evaluación de las posibles resoluciones del TSP, y en el caso de encontrarse la mejor factible, se la imprime en un archivo plano.

3.3 SIMULACIÓN DE LA RESOLUCIÓN DEL TSP POR EL MÉTODO DE APROXIMACIÓN POR EL MST EN UNA WSN

La implementación del método de resolución del TSP en cuestión, se resume en el diagrama de clases UML mostrado en la Figura 3-25, obtenido a través del *plugin* EasyUML para Netbeans 8.1.

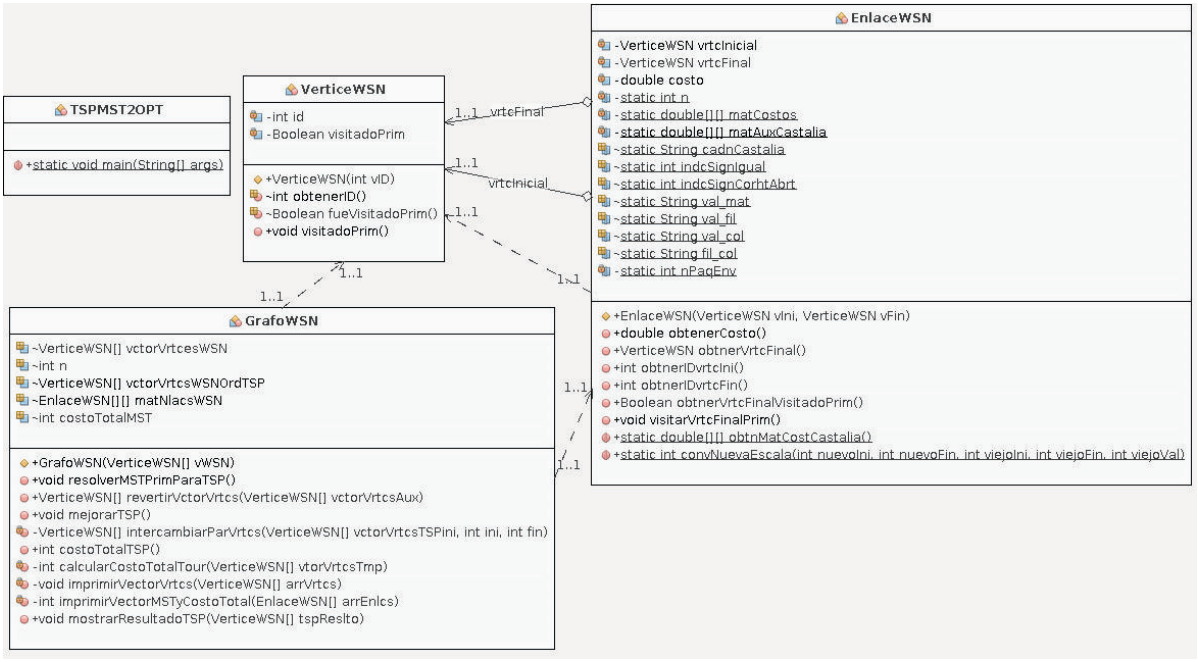
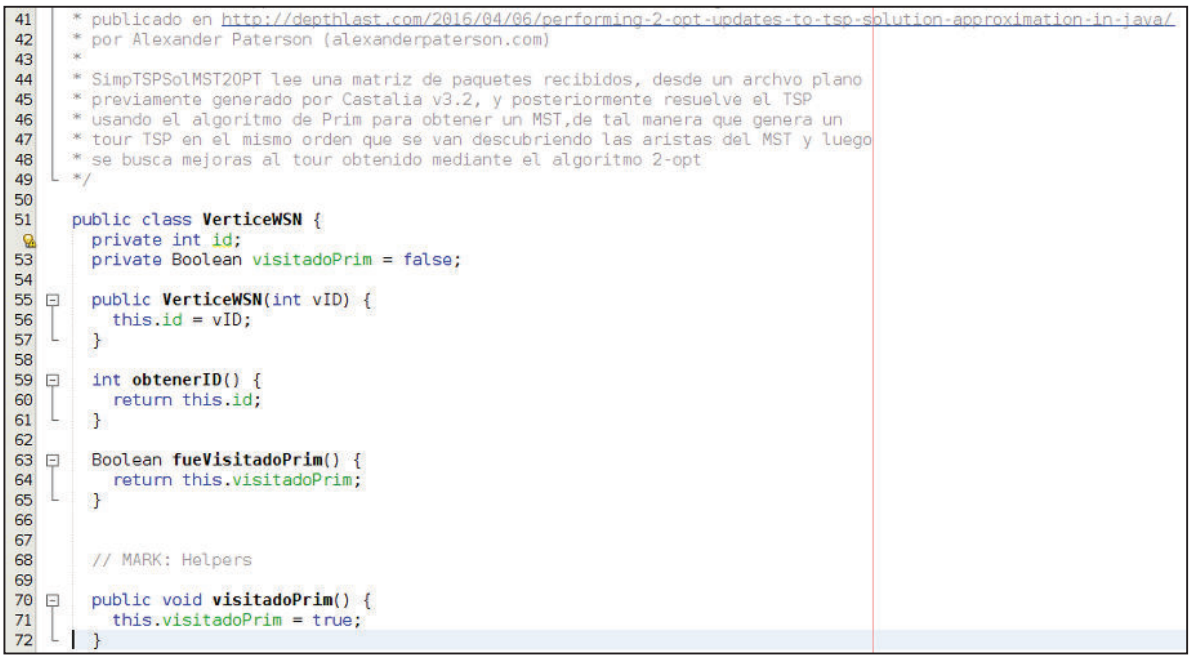


Figura 3-25 Diagrama UML de la aplicación en JAVA para resolver el TSP usando el método aproximado por Minimum Spanning Tree.

El desarrollo se basa en el código [53] [55] publicado por Alexander Paterson [78] en su página *Depth Last*. Para una mejor comprensión del código implementado, se ha utilizado nombres descriptivos para todas las variables y métodos del desarrollo.

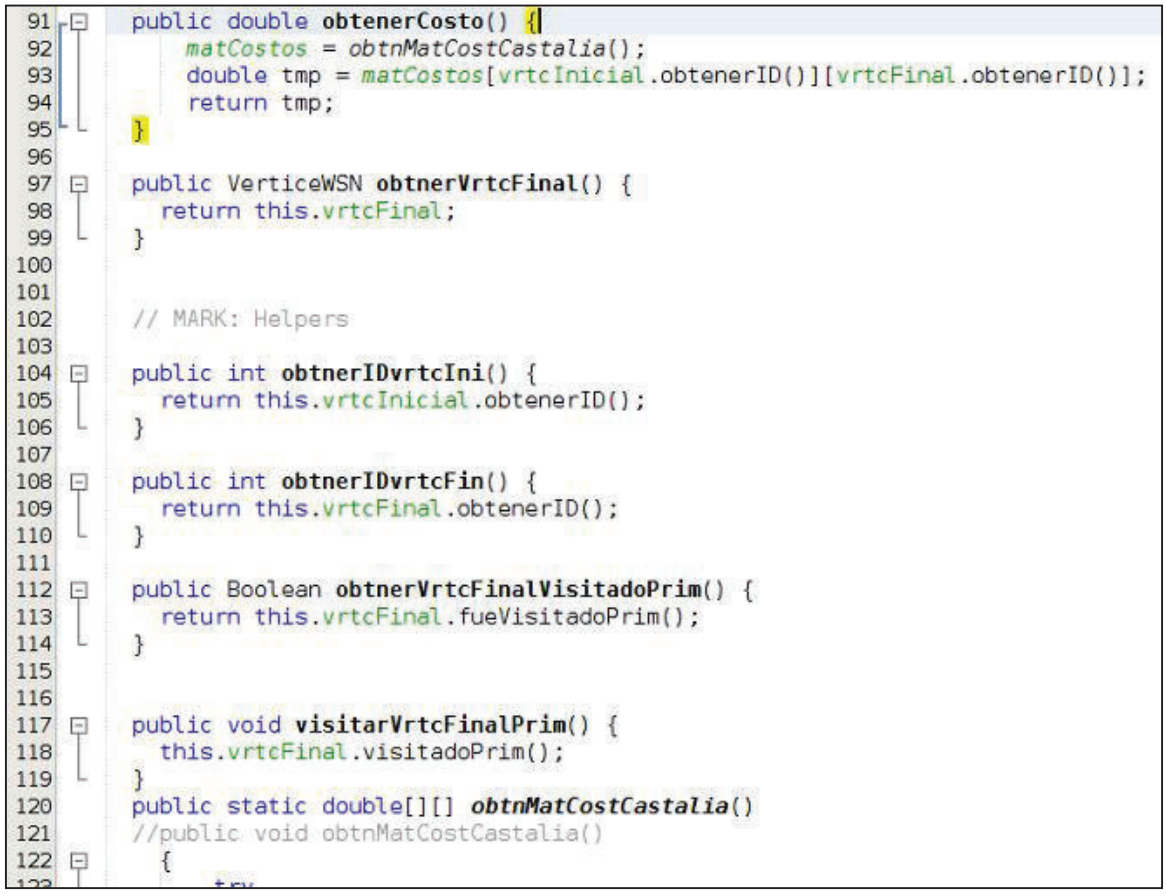
La clase más sencilla utilizada corresponde a VerticeWSN, cuyo código se muestra en el Espacio de Código 3-26. Sus miembros otorgan información sobre el identificador del nodo WSN (numerados del 0 al 17) e información sobre el algoritmo de Prim (para obtener el MST) aplicado al vértice.

Adicionalmente, se hace uso de *getters* y *setters* [79] traducidos al español para que la implementación sea más comprensible, en el caso de que otras personas desarrollen código basadas en el código presentado para este trabajo.



```
41   * publicado en http://depthlast.com/2016/04/06/performing-2-opt-updates-to-tsp-solution-approximation-in-java/
42   * por Alexander Paterson (alexanderpaterson.com)
43   *
44   * SimpTSPSolMST2OPT lee una matriz de paquetes recibidos, desde un archvo plano
45   * previamente generado por Castalia v3.2, y posteriormente resuelve el TSP
46   * usando el algoritmo de Prim para obtener un MST,de tal manera que genera un
47   * tour TSP en el mismo orden que se van descubriendo las aristas del MST y luego
48   * se busca mejoras al tour obtenido mediante el algoritmo 2-opt
49   */
50
51  public class VerticeWSN {
      private int id;
53    private Boolean visitadoPrim = false;
54
55    public VerticeWSN(int vID) {
56      this.id = vID;
57    }
58
59    int obtenerID() {
60      return this.id;
61    }
62
63    Boolean fueVisitadoPrim() {
64      return this.visitadoPrim;
65    }
66
67
68    // MARK: Helpers
69
70    public void visitadoPrim() {
71      this.visitadoPrim = true;
72    }
```

Espacio de Código 3-26 Aplicación SimpTSPSolMST2OPT: Código completo de la clase VerticeWSN.



```
91    public double obtenerCosto() {
92        matCostos = obtnMatCostCastalia();
93        double tmp = matCostos[vrtcInicial.obtenerID()][vrtcFinal.obtenerID()];
94        return tmp;
95    }
96
97    public VerticeWSN obtnerVrtcFinal() {
98      return this.vrtcFinal;
99    }
100
101
102   // MARK: Helpers
103
104   public int obtnerIDvrtcIni() {
105     return this.vrtcInicial.obtenerID();
106   }
107
108   public int obtnerIDvrtcFin() {
109     return this.vrtcFinal.obtenerID();
110   }
111
112   public Boolean obtnerVrtcFinalVisitadoPrim() {
113     return this.vrtcFinal.fueVisitadoPrim();
114   }
115
116
117   public void visitarVrtcFinalPrim() {
118     this.vrtcFinal.visitadoPrim();
119   }
120   public static double[][] obtnMatCostCastalia()
121   //public void obtnMatCostCastalia()
122     {
```

Espacio de Código 3-27 Aplicación SimpTSPSolMST2OPT: Fragmento de código de la clase EnlaceWSN, se muestran todos sus métodos.

Como se observa en los Espacios de Código 3-26 y 3-27, los parámetros tomados en cuenta para definir la clase EnlaceWSN son el vértice inicial, el vértice final (ambas instancias de la clase VerticeWSN) y el costo del enlace. El costo del enlace, es obtenido reutilizando el método obtnMatCostCastalia(), desarrollado para la aplicación SimpTSPSolBBHK, cuya explicación se encuentra en el ítem 3.2 del presente trabajo.

Además, se implementan métodos para obtener información sobre los vértices que conforman el enlace instanciado, como por ejemplo la identificación de sus nodos WSN (numerados del 0 al 17) e información referente a la implementación del algoritmo de Prim para la obtención del MST.

```
40  * Se usan code snippets del foro Stack Overflow, además del código abierto
41  * publicado en http://depthlast.com/2016/04/06/performing-2-opt-updates-to-tsp-solution-approximation-in-java/
42  * por Alexander Paterson (alexanderpaterson.com)
43  *
44  * SimpTSPSolMST2OPT lee una matriz de paquetes recibidos, desde un archvo plano
45  * previamente generado por Castalia v3.2, y posteriormente resuelve el TSP
46  * usando el algoritmo de Prim para obtener un MST,de tal manera que genera un
47  * tour TSP en el mismo orden que se van descubriendo las aristas del MST y luego
48  * se busca mejoras al tour obtenido mediante el algoritmo 2-opt
49  */
50 public class TSPMST2OPT {
51     public static void main(String[] args) {
52
53         //Número de nodos/vértices de la WSN
54         int n = 18;
55
56         // Se crea un vector de nodos/vértices WSN
57         VerticeWSN[] vctorVrtcs = new VerticeWSN[n];
58
59         // Se inicializa cada instancia de VerticeWSN
60         //con el id de cada nodo/vértice de la WSN
61         for(int i = 0; i<n;i++)
62         {
63             vctorVrtcs[i]=new VerticeWSN(i);
64         }
65
66         //Se instancia la clase principal GrafoWSN
67         //se pasa como argumento el vector de nodos/vértices WSN
68         GrafoWSN gWSN = new GrafoWSN(vctorVrtcs);
69
70         //se llama al método que resuelve el TSP en la WSN
71         gWSN.resolverMSTPrimParaTSP();
```

Espacio de Código 3-28 Aplicación SimpTSPSolMST2OPT: Fragmento de código de la clase TSPMST2OPT y su método main().

En el Espacio de Código 3-28 se muestra la clase TSPMST2OPT, esta es la clase principal y se encarga de controlar la ejecución del código implementado, alberga el método *main* del código de la aplicación SimpTSPSolMST2OPT.

Primero, se crea 18 instancias de la clase VerticeWSN, y a continuación se instancia la clase GrafoWSN pasándole como argumento a su constructor, las 18 instancias de

VerticeWSN mencionadas. Después a través de la instancia de GrafoWSN, se invoca a su método resolverMSTPrimParaTSP().

```
64      public GrafoWSN(VerticeWSN[] vWSN) {
65          //Toma como argumento el véctor de vértices/nodos WSN
66          this.vctorVrtcesWSN = vWSN;
67          //Inicializo "n" como la longitud del véctor de vértices/nodos WSN
68          this.n = vWSN.length;
69          //Vector que almacena los vértices ordenados para el TSP
70          this.vctorVrtcsWSNOrdTSP = new VerticeWSN[n + 1];
71          //Matriz equivalente a lamatriz de costos
72          this.matNlacsWSN = new EnlaceWSN[this.n][this.n];
73          //Se asigna valores en la matriz de enlaces de la WSN
74          for (int i = 0; i < this.n; i++) {
75              for (int j = 0; j < this.n; j++) {
76                  this.matNlacsWSN[i][j] = new EnlaceWSN(this.vctorVrtcesWSN[i],
77                          this.vctorVrtcesWSN[j]);
78              }
79          }
80      }
```

Espacio de Código 3-29 Aplicación SimpTSPSolMST2OPT: Fragmento de código del constructor de la clase GrfoWSN.

En el constructor de la clase GrafoWSN (véase Espacio de Código 3-29) se toman las instancias de la clase VerticeWSN y se instancia cada enlace de la WSN usando la clase EnlaceWSN.

```
95      int nMST = n+1;
96      EnlaceWSN[] vctorNlcsWSNalMST = new EnlaceWSN[nMST];
97
98      // Mientras existan vértices WSN por visitarse
99      while ((contAuxPrim+1) < this.n) {
100         EnlaceWSN menorNlaceAlMST = null;
101         EnlaceWSN nlacActualWSN;
102     // Encuentra todos los posibles enlaces WSN para formar el MST
103         for (VerticeWSN vrtcPrim : vctorVrtcsVistdsPrim) {
104         //se evita la excepción de puntero a nulo
105         if(vrtcPrim != null) {
106             for (int j = 0; j < this.n; j++) {
107                 nlacActualWSN = this.matNlacsWSN[vrtcPrim.obtenerID()][j];
108                 //Si el vértice WSN final en enlace no ha sido visitado por
109                 //el algoritmo de Prim
110                 if (nlacActualWSN.obtnerVrtcFinalVisitadoPrim() == false) {
111                     //Si es el primer vértice WSN no visitado por el alg de Prim
112                     if (menorNlaceAlMST == null) {
113                         //inicializo el menor enlace encontrado para el MST
114                         //con el enlace actual de la WSN
115                         menorNlaceAlMST = nlacActualWSN;
116                     }
117                     //caso contrario, si el costo del enlace actual de la WSN
118                     //es menor al costo del enlace de menor costo encontrado
119                     //hasta ahora
120                     else if (nlacActualWSN.obtenerCosto() <
121                             menorNlaceAlMST.obtenerCosto()) {
122                         //Se actualiza el enlace de menor costo
123                         menorNlaceAlMST = nlacActualWSN;
124                     }
125                 }
126             }
```

Espacio de Código 3-30 Aplicación SimpTSPSolMST2OPT: Fragmento de código del método resolverMSTPrimParaTSP(), donde se implementa la resolución del MST usando Prim.

En el Espacio de Código 3-30 se observa el bucle donde se calcula el MST utilizando el algoritmo de Prim, además el uso de la programación orientada a objetos facilita la obtención e inicialización de los atributos de un objeto.

```
140             vctorVrtcsVistdsPrim[contAuxPrim]= menorNlaceAlMST.obtnerVrtcFinal();
141             vctorNlcsWSNalMST[contAuxPrim] = menorNlaceAlMST;
142         }
143
144         // Para completar el Tour, se regresa al punto de partida
145         vctorVrtcsVistdsPrim[0]=vctorVrtcesWSN[0];
146
147         // Inicializo el costo total del MST
148         costoTotalMST = imprimirVectorMSTyCostoTotal(vctorNlcsWSNalMST);
149
150         System.out.println("Costo total del MST: "+costoTotalMST);
151         System.out.println("Vértices WSN después de resolver el MST con el "
152                 + "Algoritmo de Prim: ");
153         imprimirVectorVrtcs(vctorVrtcsVistdsPrim);
154         // Se ordena el vector obtenido al resolver el algoritmo de Prim
155         //para usarse con el TSP
156         for(int i=0;i<n+1;i++)
157         {
158             if(i != n)
159             {
160                 vctorVrtcsWSNOrdTSP[i]=vctorVrtcsVistdsPrim[i];
161             }
162             else
163             {
164                 vctorVrtcsWSNOrdTSP[n]=vctorVrtcsVistdsPrim[0];
165             }
166         }
167
168         vctorVrtcsWSNOrdTSP=revertirVctorVrtcs(vctorVrtcsWSNOrdTSP);
169
170         System.out.println("Vértices WSN después de ordenarlos para el TSP: ");
171         imprimirVectorVrtcs(vctorVrtcsWSNOrdTSP);
172     }
```

Espacio de Código 3-31 Aplicación SimpTSPSolMST2OPT: Fragmento de código del método resolverMSTPrimParaTSP(), donde se imprime el MST resuelto y se lo ordena para el MST.

Conforme se resuelve el MST usando el algoritmo de Prim, los vértices visitados son marcados y almacenados en el vector vctorVrtcsVistdsPrim[]. Igualmente, se almacenan los enlaces pertenecientes al MST a medida que se los va descubriendo, en el vector vctorNlacesWSNalMST[].

Tal como se observa en el Espacio de Código 3-31, también se calcula el costo total del MST resuelto, se lo almacena e imprime ya que es usado como límite inferior natural del TSP (Véase el capítulo 2 del presente trabajo).

El método de resolución recomienda que se visiten los vértices del grafo en el mismo orden en el que se fue construyendo el MST, y para ello se procede a ordenar e imprimir los vértices.

El algoritmo trabaja perfectamente hasta esta etapa con grafos pequeños. Sin embargo, cuando el número de nodos aumenta, se tiene inconsistencias en los resultados o inclusive el TSP se torna irresoluble, ya que se toma algún enlace infinito para completar el tour (Véase 3.1.2.1.2.2).

Por lo descrito en el anterior párrafo, se programan iteraciones (véase el Espacio de Código 3-32 a continuación) que parten del tour construido a partir del orden en que se fue descubriendo el MST del grafo, y se realizan permutaciones basadas en el método 2opt, de tal manera que se evalúa el costo total del tour hasta que este sea inferior o igual al doble del costo del MST [55].

```
184         int costoTotalActualTour = calcularCostoTotalTour(vctorVrtcsWSNOrdTSP);
185         System.out.println("Costo del TSP original (solo MST): "+
186                 costoTotalActualTour);
187         // Para cada par de vértices la WSN
188         for (int i = 0; i < n + 1; i++) {
189           for (int j = 0; j < n + 1; j++) {
190             // Intercambiar el par de vértices actuales
191             VerticeWSN[] vctorVrtcsIntrcmbds = intercambiarParVrtcs(
192                     vctorVrtcsWSNOrdTSP, i, j);
193             // Obtener el nuevo costo total del Tour
194             int nuevoCostoTotalTour = calcularCostoTotalTour(vctorVrtcsIntrcmbds);
195             System.out.println("Costo del TSP modificado (MST y 2-opt): "+
196                     nuevoCostoTotalTour);
197             //Se obtiene la primera mejor aproximación al costo total
198             //del TSP resuelto óptimamente y se da por terminado el programa
199             /**/
200             if(nuevoCostoTotalTour <= 2*costoTotalMST &&
201                     //para que sea un TSP válido debe empezar y terminar en el
202                     //mismo nodo/vértice WSN
203                     vctorVrtcsIntrcmbds[0].obtenerID() == 0 &&
204                     vctorVrtcsIntrcmbds[n].obtenerID() == 0)
205             {
206                 System.out.println("El primer mejor costo aprox. del "
207                         + "TSP modificado (MST y 2-opt): "+
208                     nuevoCostoTotalTour);
209                 mostrarResultadoTSP(vctorVrtcsIntrcmbds);
210                 System.exit(0);
211             }
212             /**/
213             // Si es que existe una mejora en el costo total, actualizar valores
214             if (nuevoCostoTotalTour < costoTotalActualTour) {
215               this.vctorVrtcsWSNOrdTSP = vctorVrtcsIntrcmbds;
216               costoTotalActualTour = nuevoCostoTotalTour;
```

Espacio de Código 3-32 Aplicación SimpTSPSolMST2OPT: Fragmento de código del método mejorarTSP(), donde se toma el primer tour que cumpla la condición de la línea 200.

Cómo se observa en el Espacio de Código 3-33 las iteraciones se controlan desde la clase principal TSPMST2OPT, donde se deja como condición auxiliar, el control propuesto por Alexander Paterson, la cual verifica si las iteraciones presente y la anterior han obtenido el mismo costo total del TSP para parar la búsqueda del mejor tour.

```
332    public void mostrarResultadoTSP(VerticeWSN[] tspReslto)
333        {
334            try
335            {
336                //Se guarda la solución del TSP en un archivo plano, separando con
337                //comas los ID de los vértices/nodos WSN debidamente ordenados.
338
339                File archTSPresult =
340                        new File("/home/jt/Escritorio/SW-jt/castalia3_2/Castalia-3.2/Simulations/"
341                                + "throughputTest_jtito/ArchPlanoTSPresuelto_jtito.txt");
342                //File archTSPresult =
343                        //new File("D:\\TESIS TEMPO\\UN MES\\tspbbjavafinal"
344                                //+ "\\ArchPlanoTSPresuelto_jtito.txt");
345
346                try (PrintWriter prntWrtrEscrbrResult =
347                        new PrintWriter(archTSPresult))
348                {
349                    /*
350                    lstTSPresuelto.stream().forEach((item) -> {
351                        prntWrtrEscrbrResult.append(item+",");
352                    });
353                    */
354                    int i = 0;
355                    while(i < tspReslto.length)
356                    {
357                        if(i > 0)
358                        {
359                            prntWrtrEscrbrResult.println("SN.node["+tspReslto[i-1].obtenerID()+"].Application.nextRecipient
360                        }
361                        i++;
```

Espacio de Código 3-33 Aplicación SimpTSPSolMST2OPT: Fragmento de código del método mostrarResultadoTSP(), donde se despliega el TSP resuelto y se lo escribe en el archivo plano ArchPlanoTSPresuelto_jtito.txt

Luego de haber encontrado el TSP que mejor se aproxima a la solución óptima, se procede a mostrar en pantalla el tour resultante y se escribe en el archivo plano ArchPlanoTSPresuelto_jtito.txt, el TSP obtenido en el formato del simulador Castalia.

3.4 ANÁLISIS DE RESULTADOS PARA LOS MÉTODOS DE RESOLUCIÓN DEL TSP

3.4.1 ESTADÍSTICAS POR CADA CASO DE WSN

Para todos los gráficos a continuación se ha utilizado la herramienta CastaliaPlot. Además, para todos los casos de WSN se tienen los siguientes datos en común.

En total circulan por la red (100 paquetes por cada nodo x 18 nodos) 1800 paquetes en la primera etapa de la simulación. La segunda etapa de la simulación corresponde a la resolución del TSP por alguno de los dos métodos planteados en este proyecto.

En total circulan por la red alrededor de 4000 paquetes x18 nodos =72000 paquetes en la tercera etapa de la simulación.

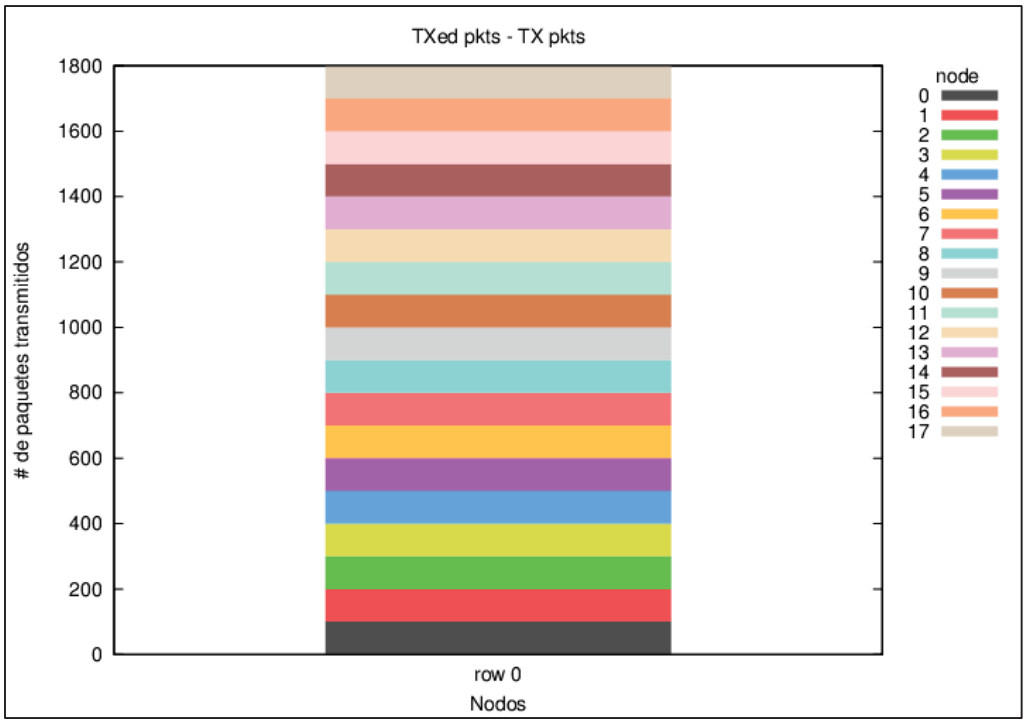


Figura 3-26 Primera etapa de la simulación (ConnectivityMap): todos los nodos transmiten 100 paquetes.

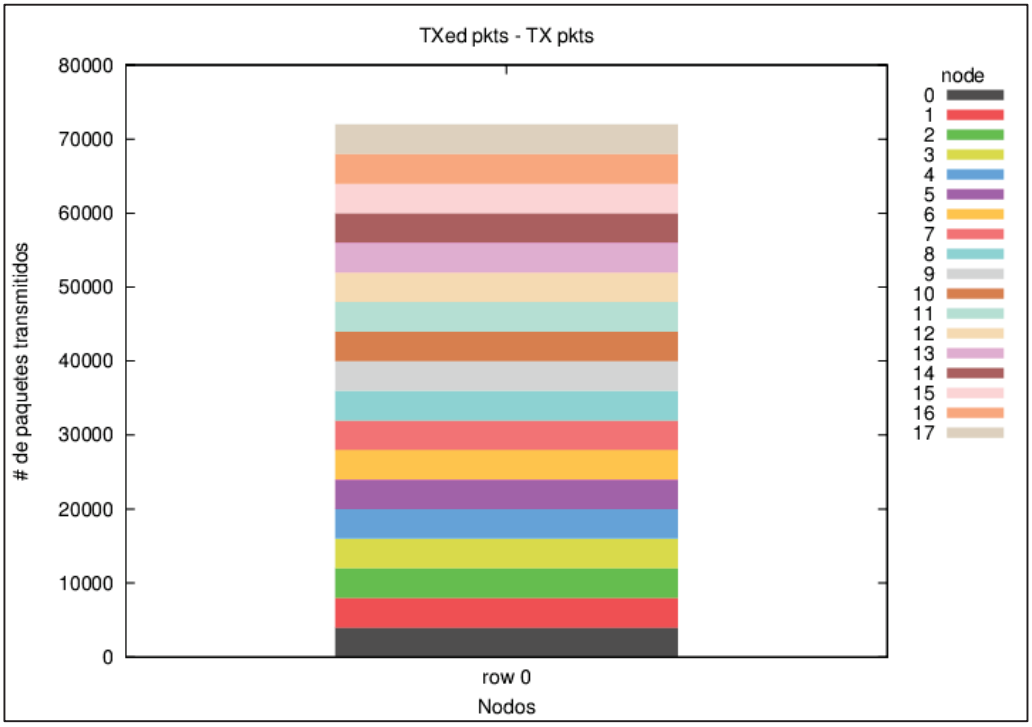


Figura 3-27 Tercera etapa de la simulación (ThroughputTest): todos los nodos transmiten 4000 paquetes.

Failed with NO interference	Failed, below sensitivity	Received with NO interference	Failed, non RX state	Failed with interference
Paq. Fallidos: Ruido Térmico	Paq. Fallidos: RSSI inferior a -95dBm	Paq. Recibidos (sin Interferencia)	Paq. Fallidos: Nodo en estado diferente a RX.	Paq. Fallidos: Interferencia (debida a los nodos cercanos)

Tabla 3-5 Explicación de los parámetros para los paquetes fallidos desplegados por CastaliaPlot [75].

Para las gráficas de paquetes fallidos en la red de sensores inalámbricos, CastaliaPlot por defecto despliega un subconjunto de los parámetros mostrados en la Tabla 3-5.

En todos los casos de redes inalámbricas de sensores (WSN – *Wireless Sensor Network*) simuladas que serán presentadas a continuación, CastaliaPlot toma como referencia el total de paquetes circulando en la WSN respecto a cada nodo de la WSN.

Como consecuencia se observa en cada gráfica concerniente a la cuantificación de paquetes, en el eje vertical los valores máximos coinciden con los mostrados en la Figura 3-26 o Figura 3-27.

Entonces para interpretar correctamente los gráficos obtenidos con CastaliaPlot, se debe tener en cuenta que cada nodo “escuchará” lo que acontece con todos los paquetes circulantes por la WSN (según la etapa de la simulación que se esté ejecutando), sin tener en cuenta los paquetes que el nodo transmite.

Por lo tanto, en la primera etapa de la simulación el nodo escuchará (1800 paquetes circulantes por toda la WSN – 100 paquetes transmitidos por el nodo en cuestión) 1700 paquetes, de manera similar para la segunda etapa de la simulación cada nodo escuchará (72000 paquetes circulantes por toda la WSN – aproximadamente 4000 paquetes transmitidos por el nodo en cuestión) 68000 paquetes aproximadamente.

De los subprocesos de simulación descritos en el ítem 3.1.3.1, existen dos subprocesos que se ejecutan a través del simulador Castalia, las simulaciones son ConnectivityMap y ThroughputTest. Por lo tanto existen dos resultados que deben ser consolidados (Véase 3.4.2).

3.4.1.1 Escenario A

3.4.1.1.1 WSN conformada por nodos sensores TelosB

3.4.1.1.1.1 Branch and Bound

3.4.1.1.1.1.1 Energía Consumida

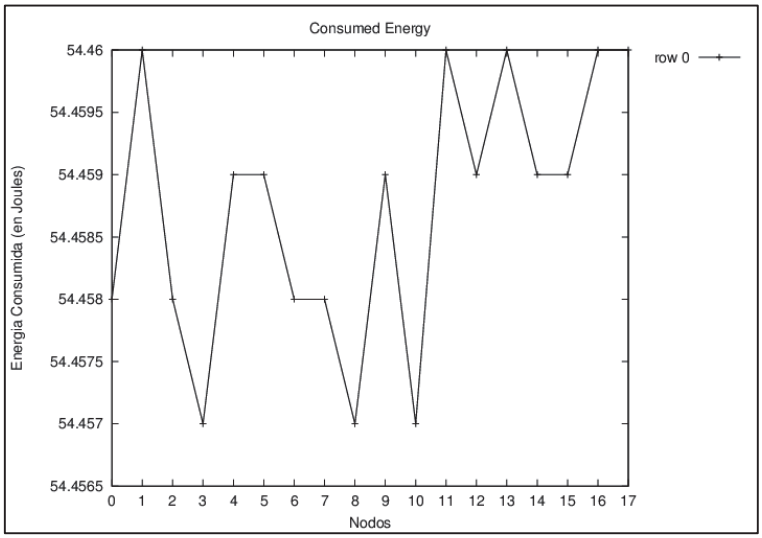


Figura 3-28 Primera etapa de la simulación (ConnectivityMap): Energía consumida por nodo.

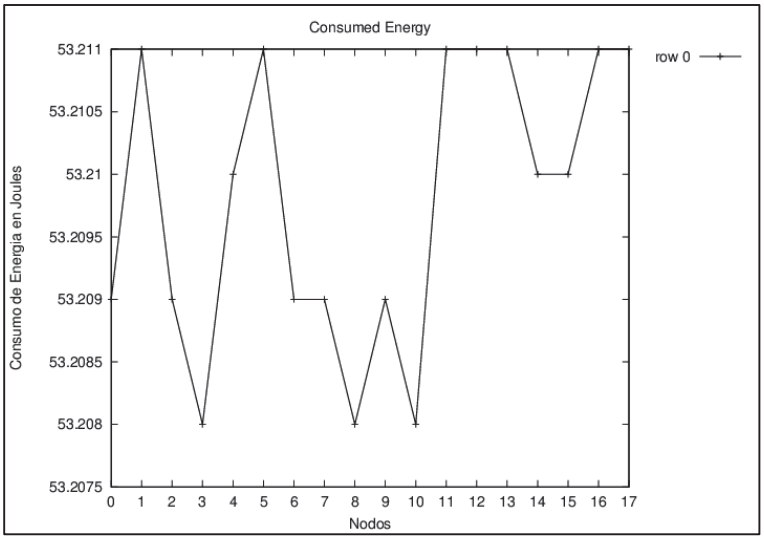


Figura 3-29 Segunda etapa de la simulación (ThroughputTest): Energía consumida por nodo.

En las Figuras 3-28 y 3-29 se observa el consumo de energía por nodo debido al primer y segundo subprocesos de simulación, respectivamente. La WSN consta de motas TelosB desplegadas en el escenario “A” y el TSP se resuelve mediante el método de B&B.

3.4.1.1.1.1.2 Paquetes Fallidos

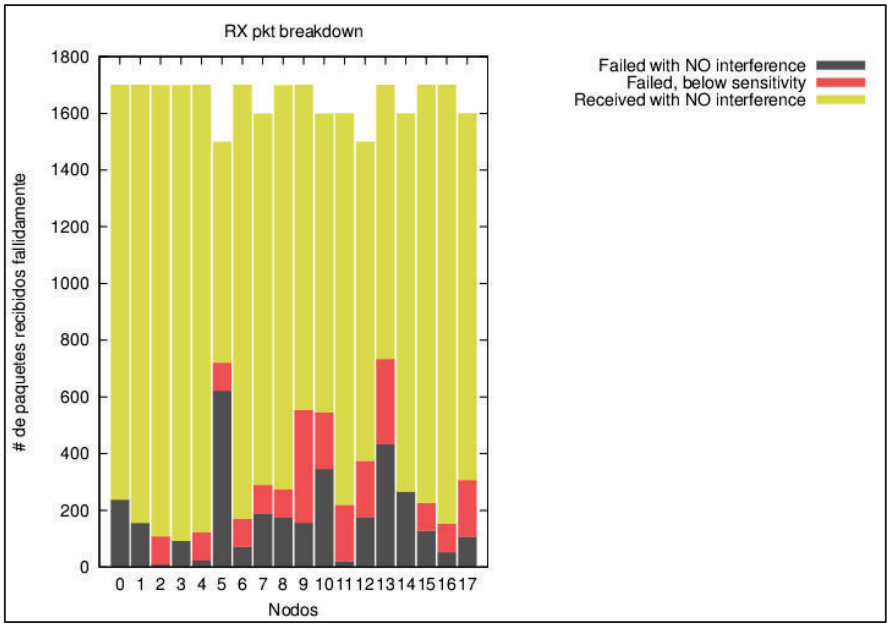


Figura 3-30 Primera etapa de la simulación (ConnectivityMap): Paquetes fallidos.

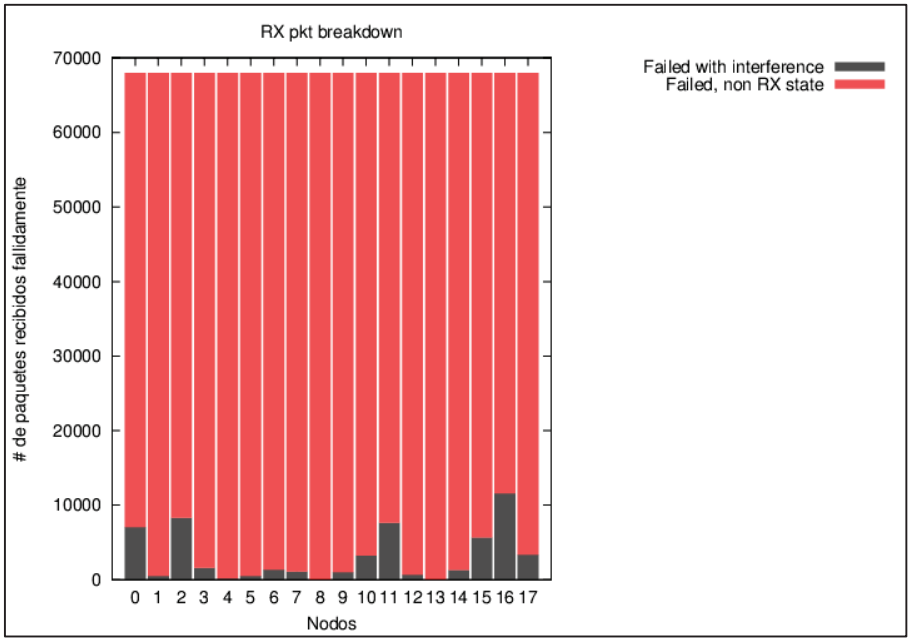


Figura 3-31 Tercera etapa de la simulación (ThroughputTest): Paquetes Fallidos.

En las Figura 3-30 y 3-31 se observa los paquetes fallidos por nodo debido al primer y segundo subprocesos de simulación, respectivamente. En la Figura 3-31 el número de paquetes fallidos en rojo es debido a que no se especifica el control de acceso al medio. La WSN consta de motas TelosB desplegadas en el escenario “A” y el TSP se resuelve mediante el método de B&B.

3.4.1.1.1.2 Aproximación por MST

3.4.1.1.1.2.1 Energía Consumida

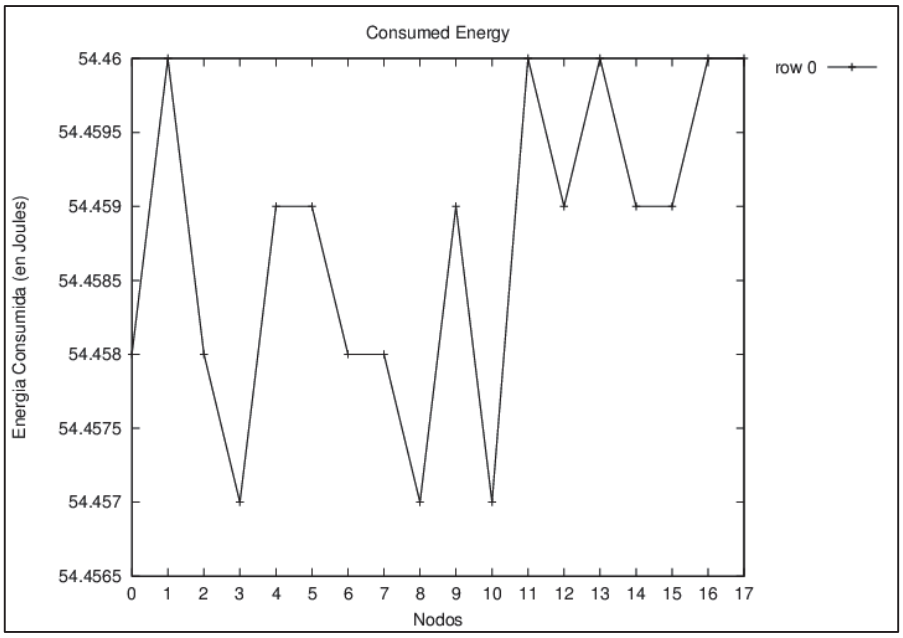


Figura 3-32 Primera etapa de la simulación (ConnectivityMap): Energía consumida por nodo.

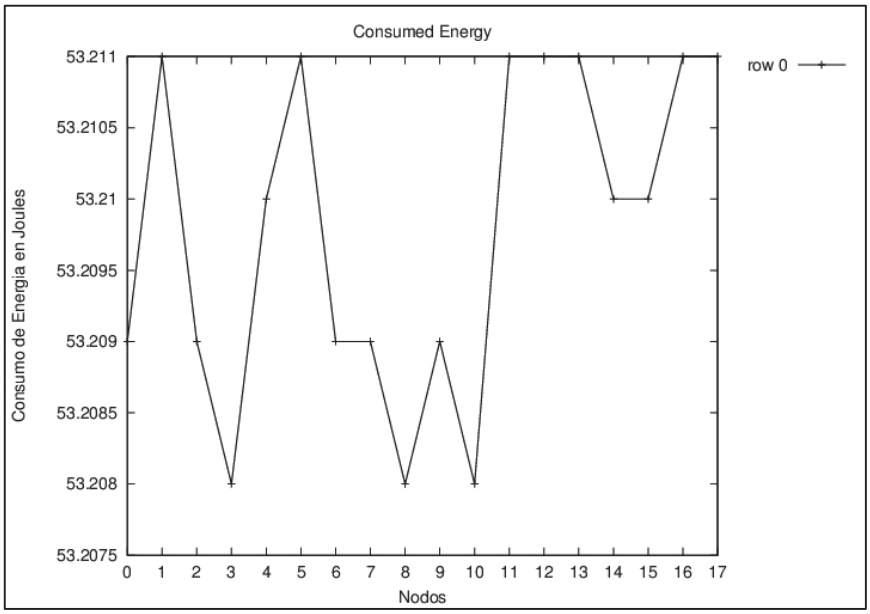


Figura 3-33 Segunda etapa de la simulación (ThroughputTest): Energía consumida por nodo.

En las Figuras 3-32 y 3-33 se observa el consumo de energía por nodo debido al primer y segundo subprocesos de simulación, respectivamente. La WSN consta de motas TelosB desplegadas en el escenario “A” y el TSP se resuelve mediante el método de aproximación por MST.

3.4.1.1.1.2.2 Paquetes Fallidos

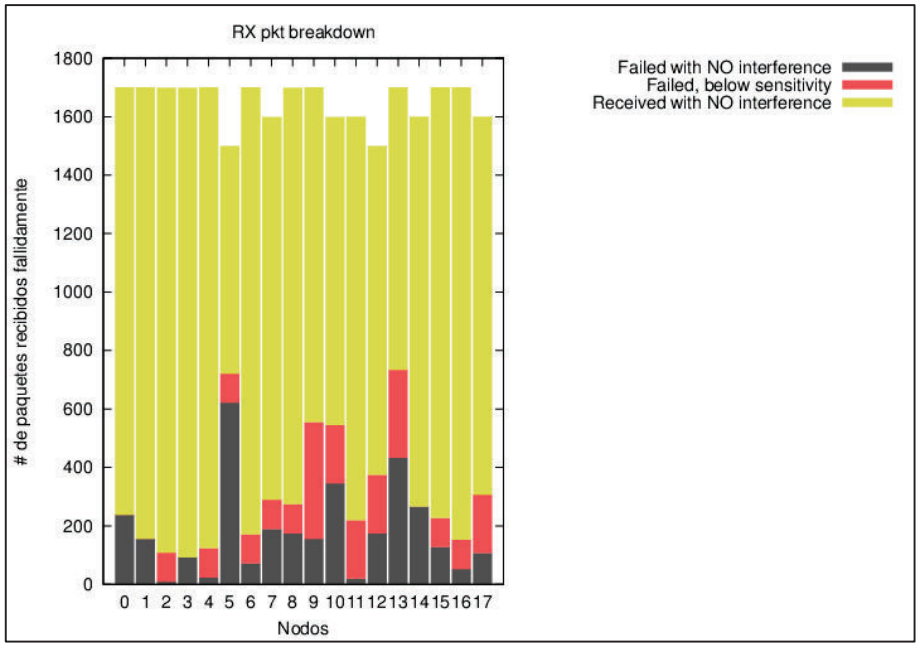


Figura 3-34 Primera etapa de la simulación (ConnectivityMap): Paquetes fallidos.

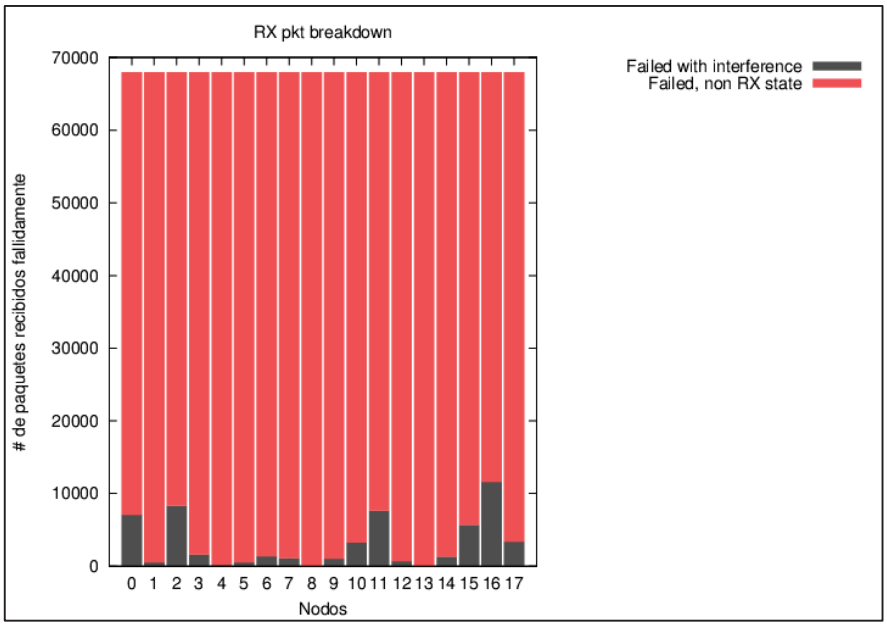


Figura 3-35 Tercera etapa de la simulación (ThroughputTest): Paquetes Fallidos.

En las Figuras 3-34 y 3-35 se observa los paquetes fallidos por nodo debido al primer y segundo subprocesos de simulación, respectivamente. En la Figura 3-35 el número de paquetes fallidos en rojo es debido a que no se especifica el control de acceso al medio. La WSN consta de motas TelosB desplegadas en el escenario “A” y el TSP se resuelve mediante el método de aproximación por MST.

3.4.1.1.2 WSN conformada por nodos sensores Imote2

3.4.1.1.2.1 Branch and Bound

3.4.1.1.2.1.1 Energía Consumida

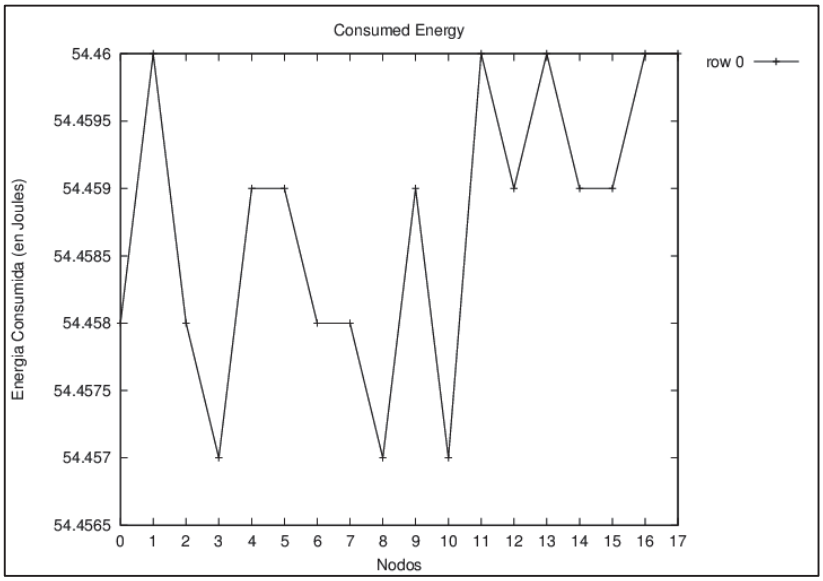


Figura 3-36 Primera etapa de la simulación (ConnectivityMap): Energía consumida por nodo.

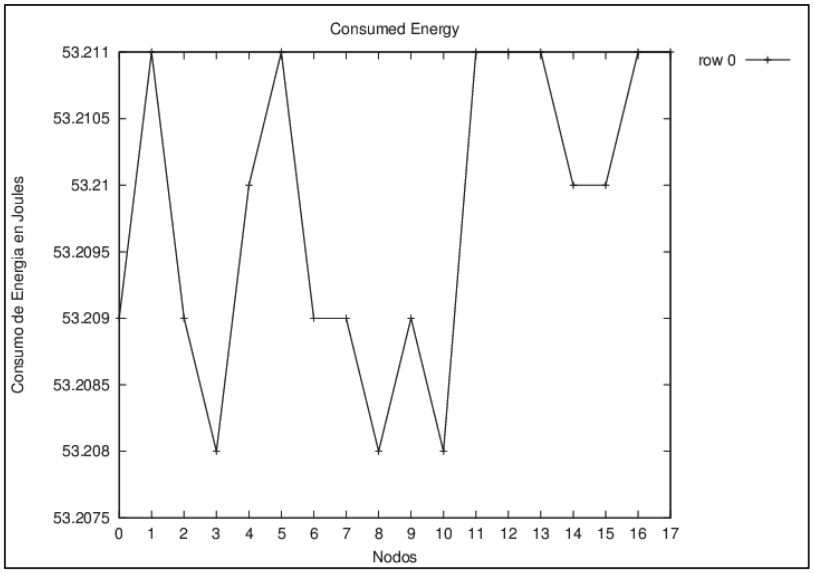


Figura 3-37 Segunda etapa de la simulación (ThroughputTest): Energía consumida por nodo.

En las Figuras 3-36 y 3-37 se observa el consumo de energía por nodo debido al primer y segundo subprocesos de simulación, respectivamente. La WSN consta de motas Imote2 desplegadas en el escenario “A” y el TSP se resuelve mediante el método de B&B.

3.4.1.1.2.1.2 Paquetes Fallidos

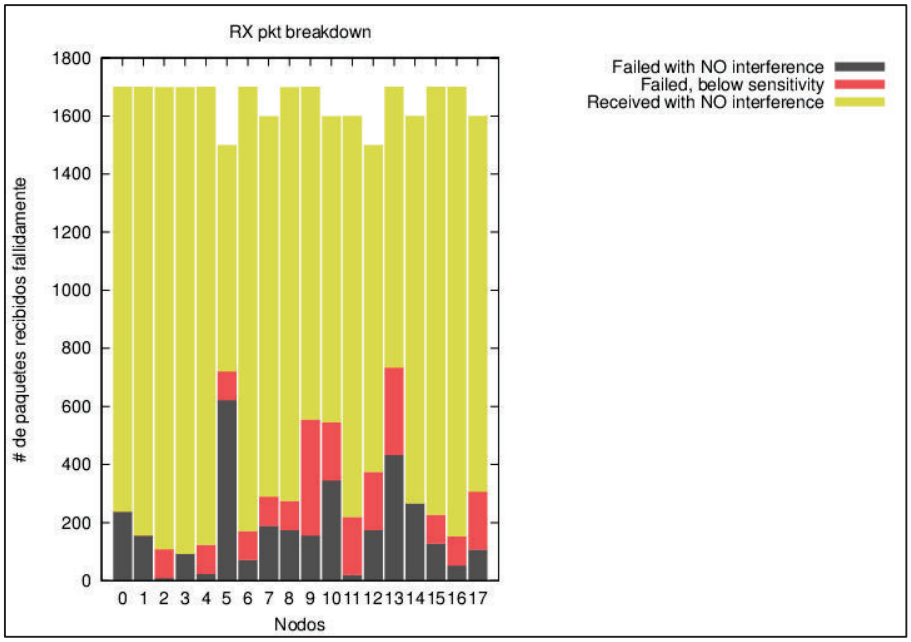


Figura 3-38 Primera etapa de la simulación (ConnectivityMap): Paquetes fallidos.

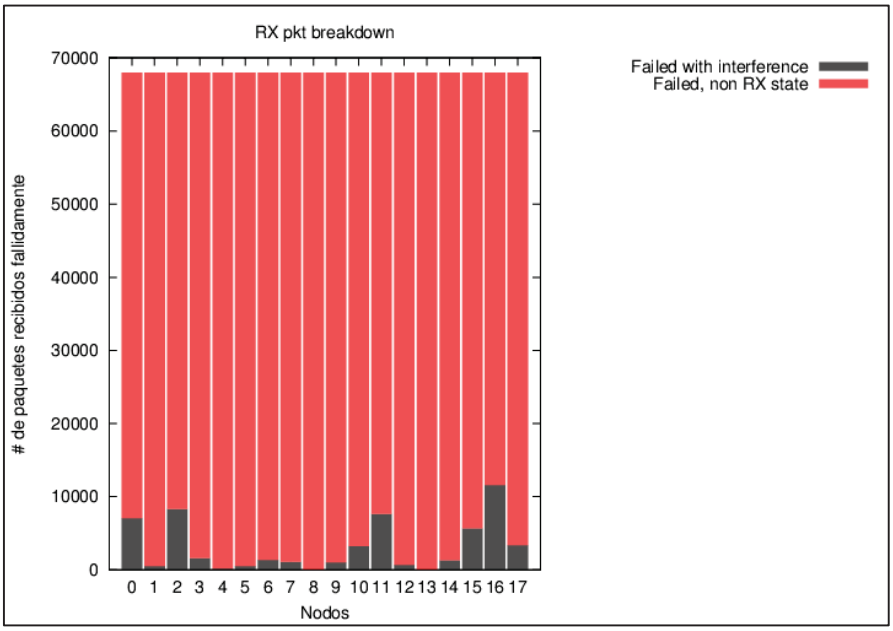


Figura 3-39 Tercera etapa de la simulación (ThroughputTest): Paquetes Fallidos.

En las Figuras 3-38 y Figura 3-39 se observa los paquetes fallidos por nodo debido al primer y segundo subprocesos de simulación, respectivamente. En la Figura 3-39 el número de paquetes fallidos en rojo es debido a que no se especifica el control de acceso al medio. La WSN consta de motas Imote2 desplegadas en el escenario “A” y el TSP se resuelve mediante el método de B&B.

3.4.1.1.2.2 Aproximación por MST

3.4.1.1.2.2.1 Energía Consumida

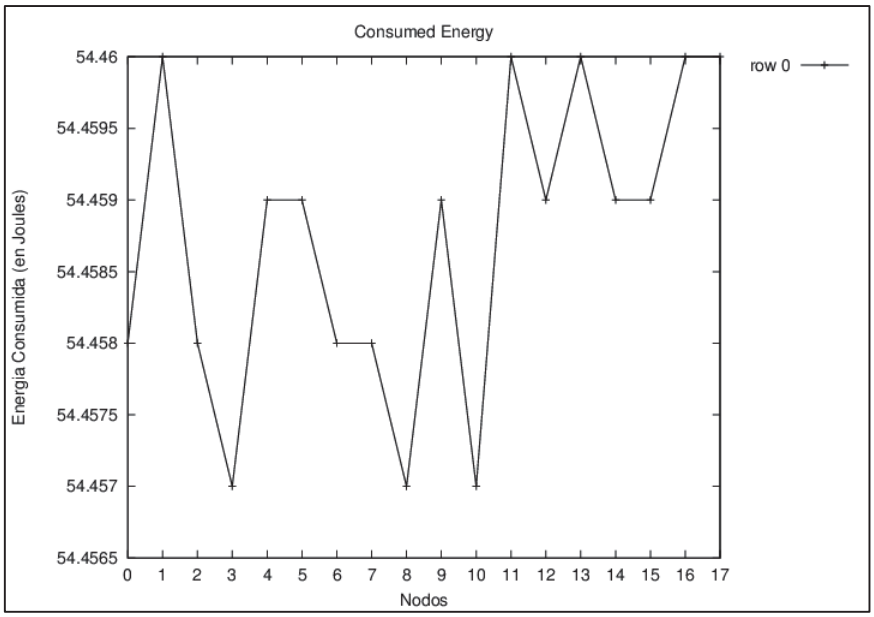


Figura 3-40 Primera etapa de la simulación (ConnectivityMap): Energía consumida por nodo.

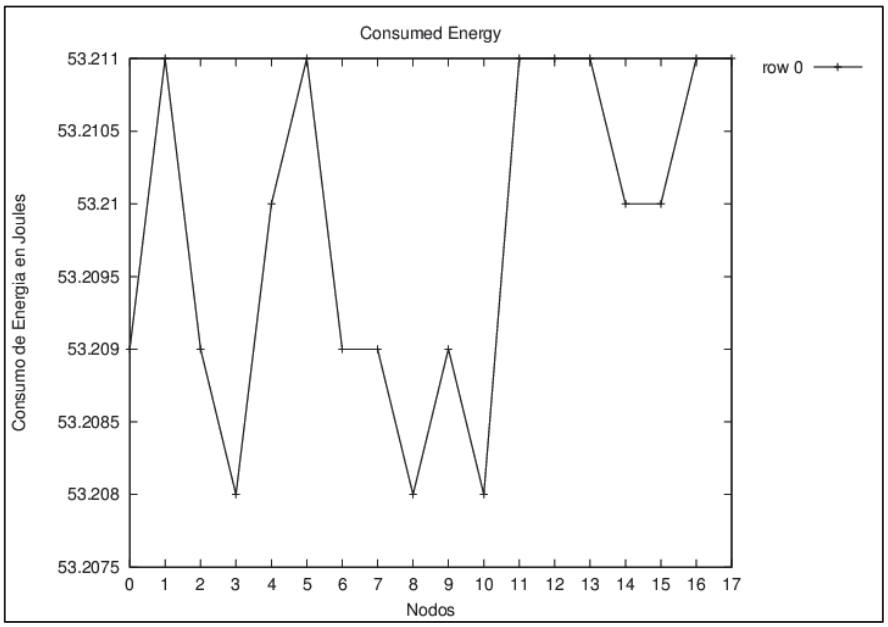


Figura 3-41 Segunda etapa de la simulación (ThroughputTest): Energía consumida por nodo.

En las Figuras 3-40 y Figura 3-41 se observa el consumo de energía por nodo debido al primer y segundo subprocesos de simulación, respectivamente. La WSN consta de motas Imote2 desplegadas en el escenario “A” y el TSP se resuelve mediante el método de aproximación por MST.

3.4.1.1.2.2.2 Paquetes Fallidos

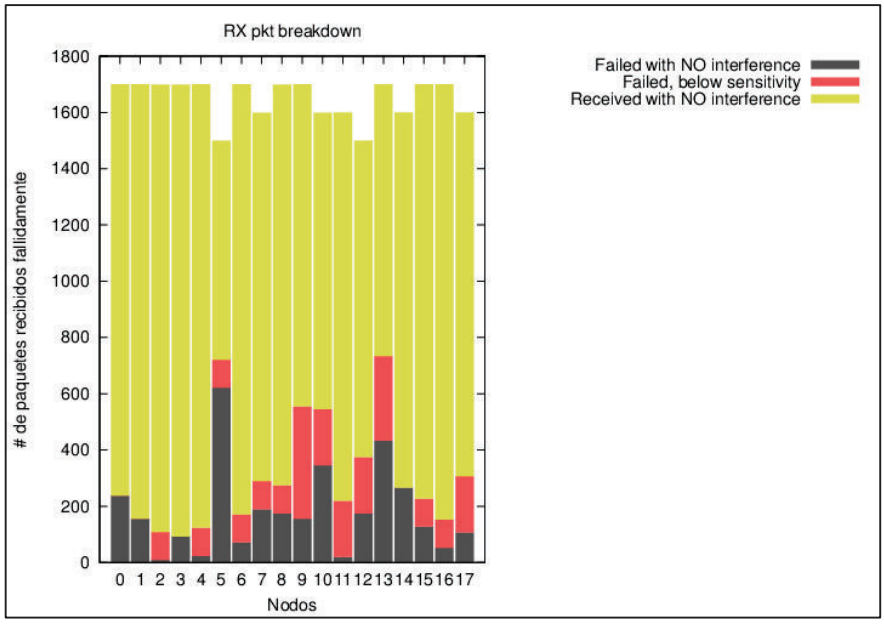


Figura 3-42 Primera etapa de la simulación (ConnectivityMap): Paquetes fallidos.

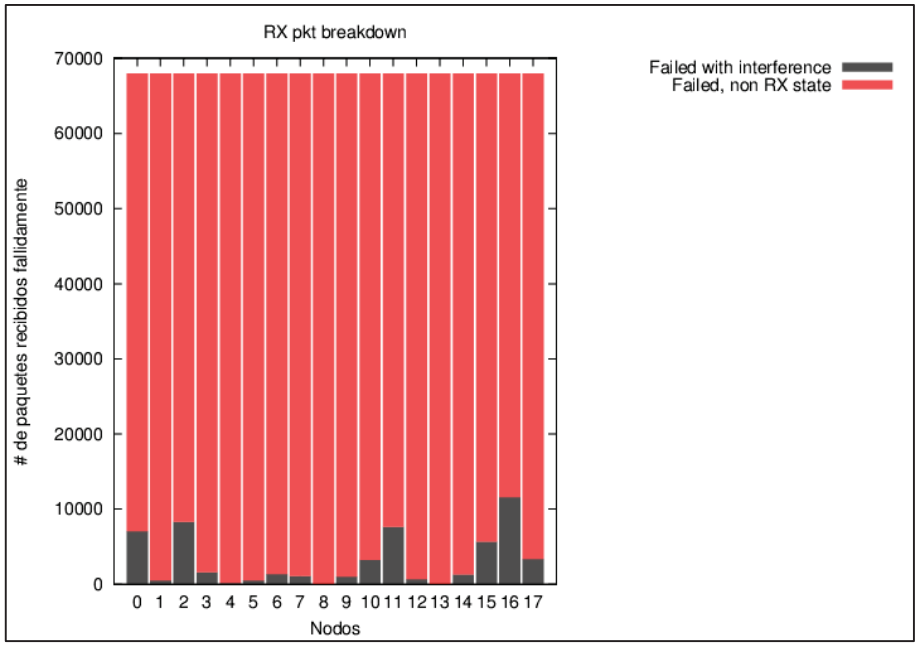


Figura 3-43 Tercera etapa de la simulación (ThroughputTest): Paquetes Fallidos.

En las Figuras 3-42 y 3-43 se observa los paquetes fallidos por nodo debido al primer y segundo subprocesos de simulación, respectivamente. En la Figura 3-43 el número de paquetes fallidos en rojo es debido a que no se especifica el control de acceso al medio. La WSN consta de motas Imote2 desplegadas en el escenario “A” y el TSP se resuelve mediante el método de aproximación por MST.

3.4.1.1.3 WSN conformada por nodos sensores Zolertia

3.4.1.1.3.1 Branch and Bound

3.4.1.1.3.1.1 Energía Consumida

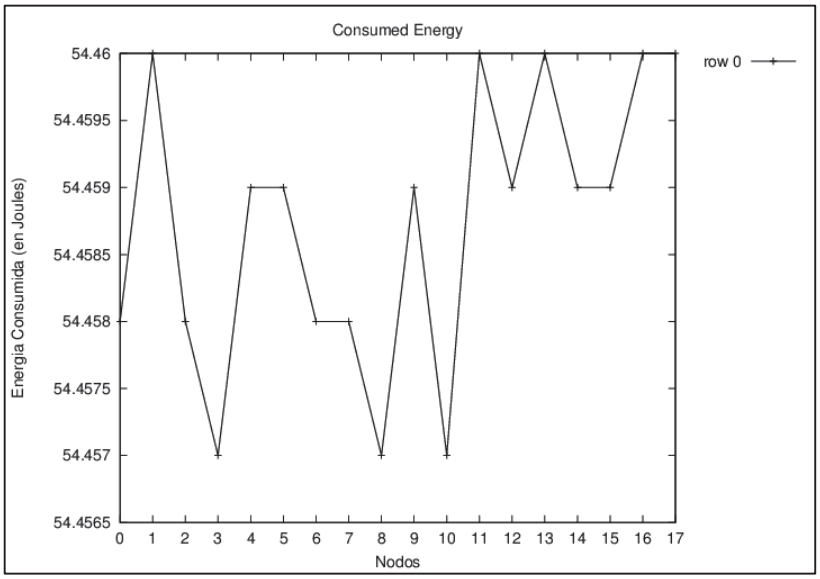


Figura 3-44 Primera etapa de la simulación (ConnectivityMap): Energía consumida por nodo.

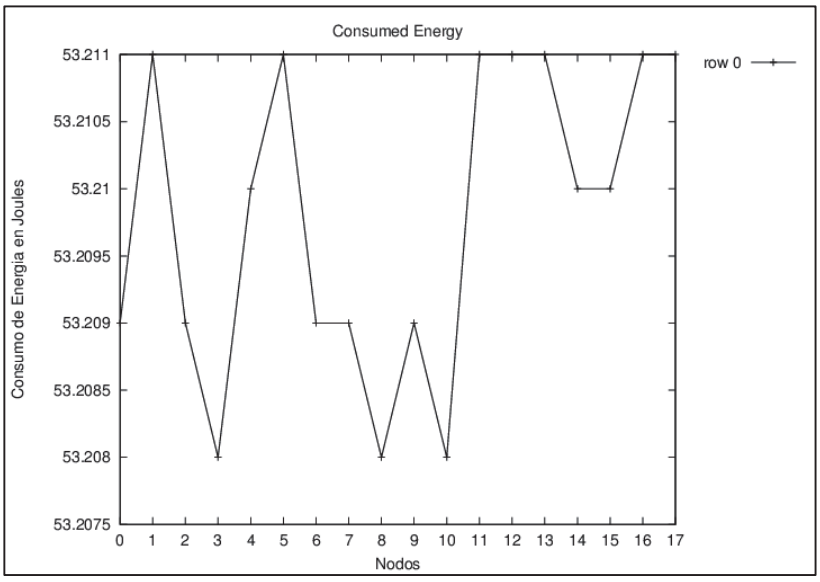


Figura 3-45 Segunda etapa de la simulación (ThroughputTest): Energía consumida por nodo.

En las Figuras 3-44 y 3-45 se observa el consumo de energía por nodo debido al primer y segundo subprocesos de simulación, respectivamente. La WSN consta de motas Zolertia desplegadas en el escenario “A” y el TSP se resuelve mediante el método de B&B.

3.4.1.1.3.1.2 Paquetes Fallidos

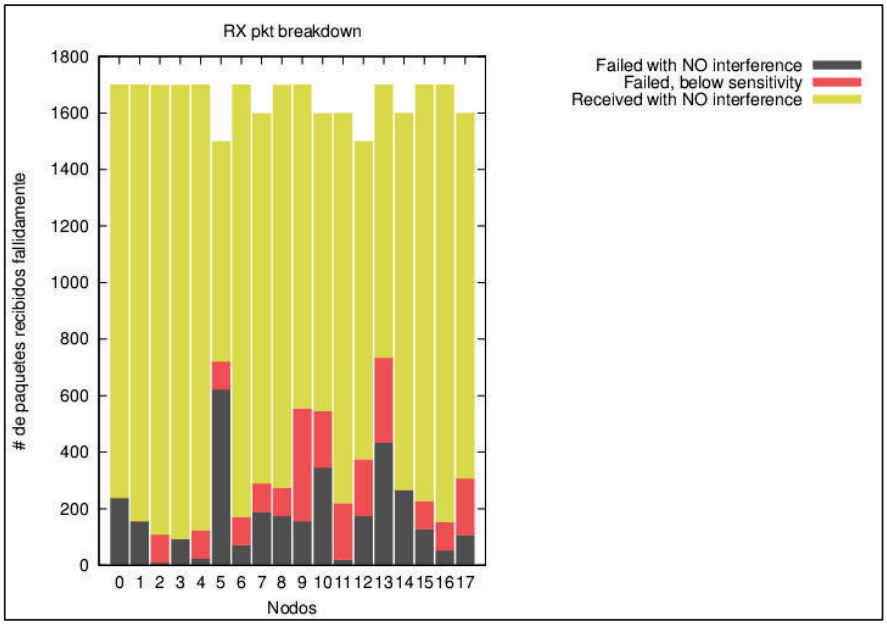


Figura 3-46 Primera etapa de la simulación (ConnectivityMap): Paquetes fallidos.

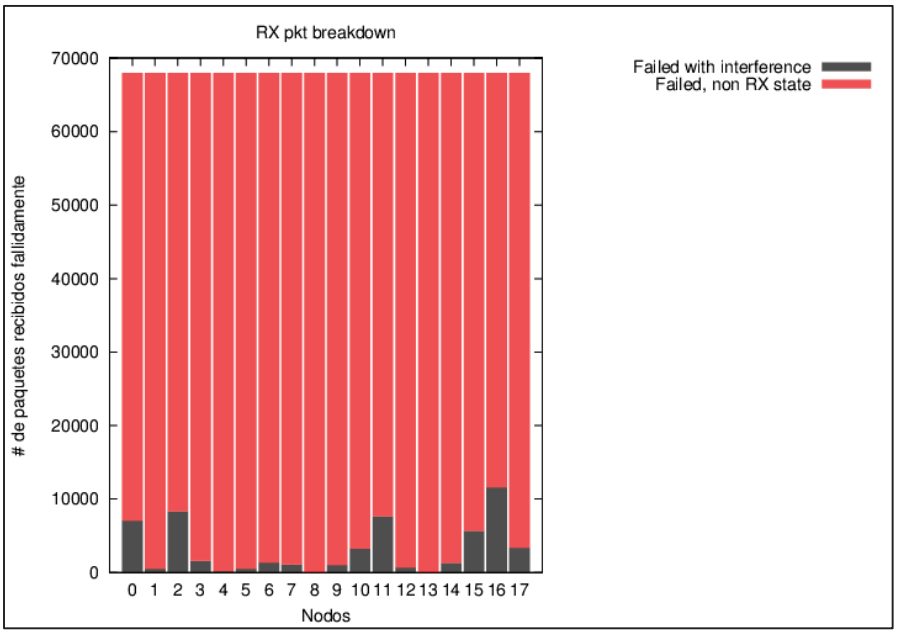


Figura 3-47 Tercera etapa de la simulación (ThroughputTest): Paquetes Fallidos.

En las Figuras 3-46 y 3-47 se observa los paquetes fallidos por nodo debido al primer y segundo subprocesos de simulación, respectivamente. En la Figura 3-47 el número de paquetes fallidos en rojo es debido a que no se especifica el control de acceso al medio. La WSN consta de motas Zolertia desplegadas en el escenario “A” y el TSP se resuelve mediante el método B&B.

3.4.1.1.3.2 Aproximación por MST

3.4.1.1.3.2.1 Energía Consumida

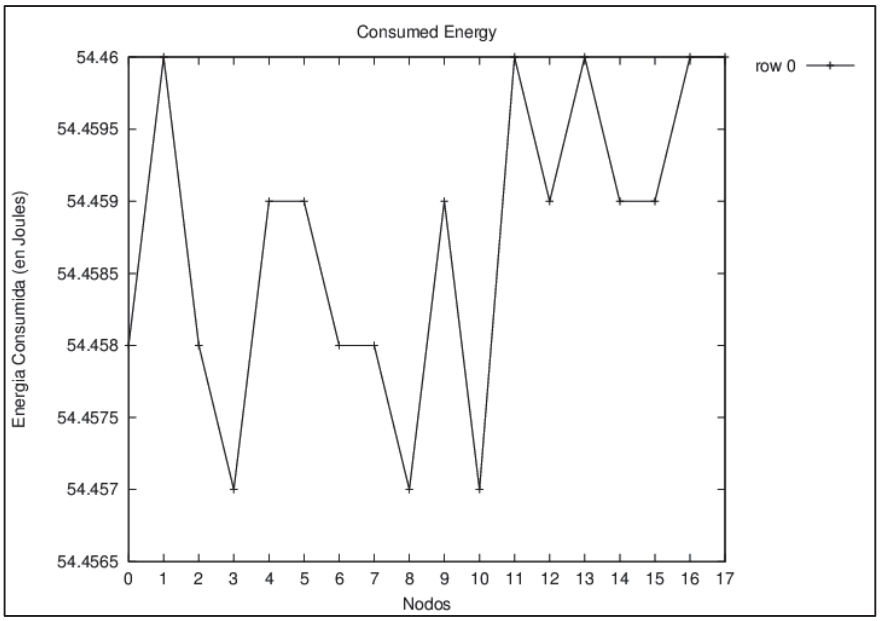


Figura 3-48 Primera etapa de la simulación (ConnectivityMap): Energía consumida por nodo.

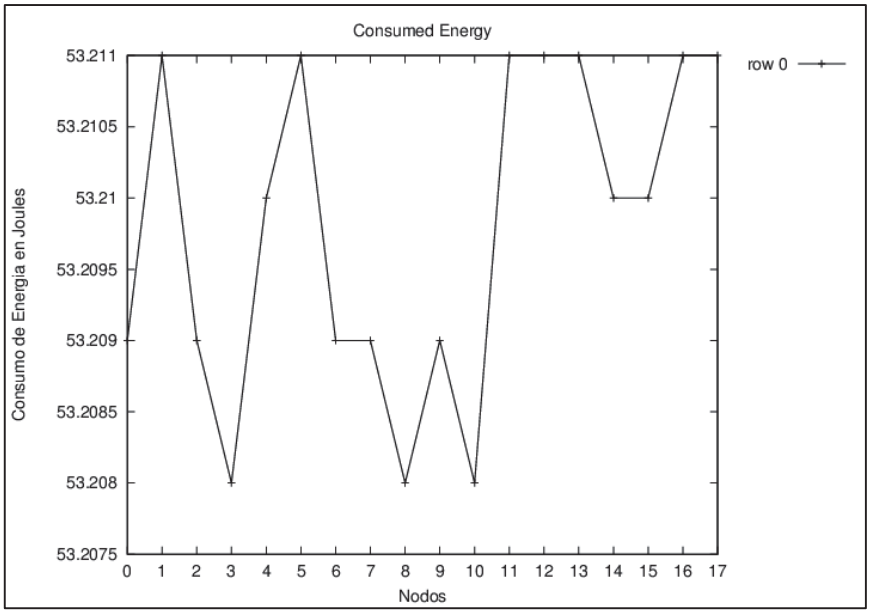


Figura 3-49 Segunda etapa de la simulación (ThroughputTest): Energía consumida por nodo.

En las Figuras 3-48 y 3-49 se observa el consumo de energía por nodo debido al primer y segundo subprocesos de simulación, respectivamente. La WSN consta de motas Zolertia desplegadas en el escenario “A” y el TSP se resuelve mediante el método de aproximación por MST.

3.4.1.1.3.2.2 Paquetes Fallidos

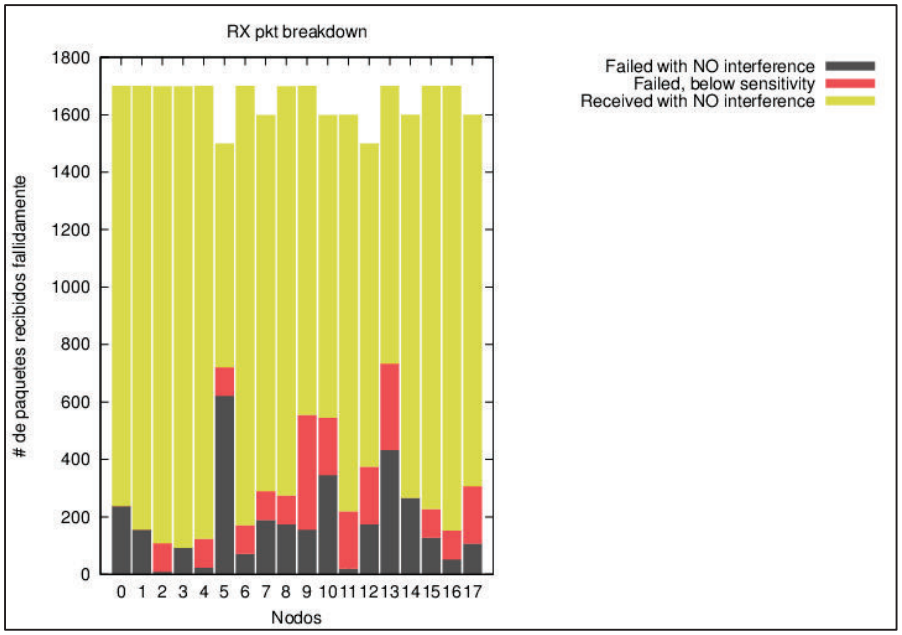


Figura 3-50 Primera etapa de la simulación (ConnectivityMap): Paquetes fallidos.

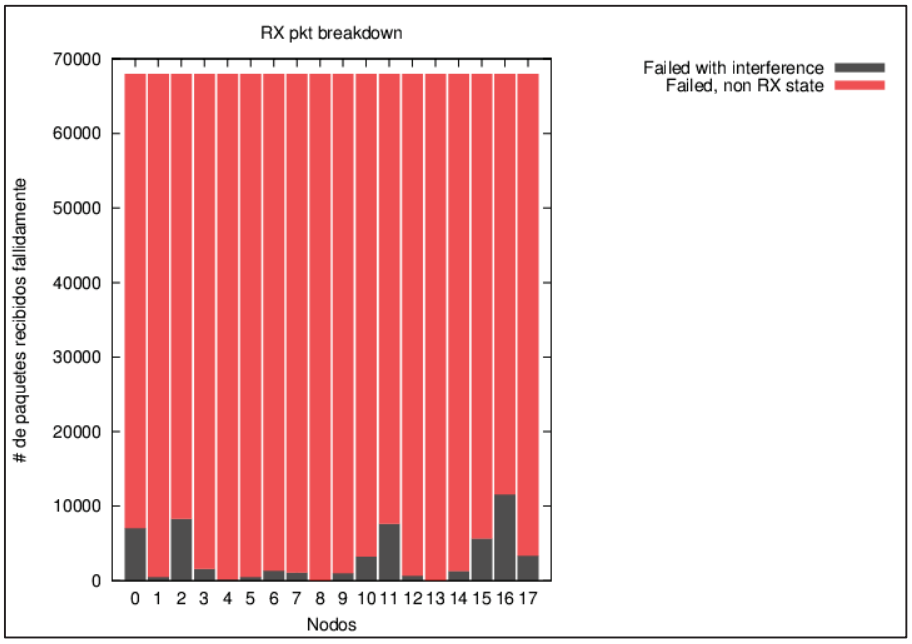


Figura 3-51 Tercera etapa de la simulación (ThroughputTest): Paquetes Fallidos.

En las Figura 3-50 y Figura 3-51 se observa los paquetes fallidos por nodo debido al primer y segundo subprocesos de simulación, respectivamente. En la Figura 3-51 el número de paquetes fallidos en rojo es debido a que no se especifica el control de acceso al medio. La WSN consta de motas Zolertia desplegadas en el escenario “A” y el TSP se resuelve mediante el método de aproximación por MST.

3.4.1.2 Escenario B

3.4.1.2.1 WSN conformada por nodos sensores TelosB

3.4.1.2.1.1 Branch and Bound

3.4.1.2.1.1.1 Energía Consumida

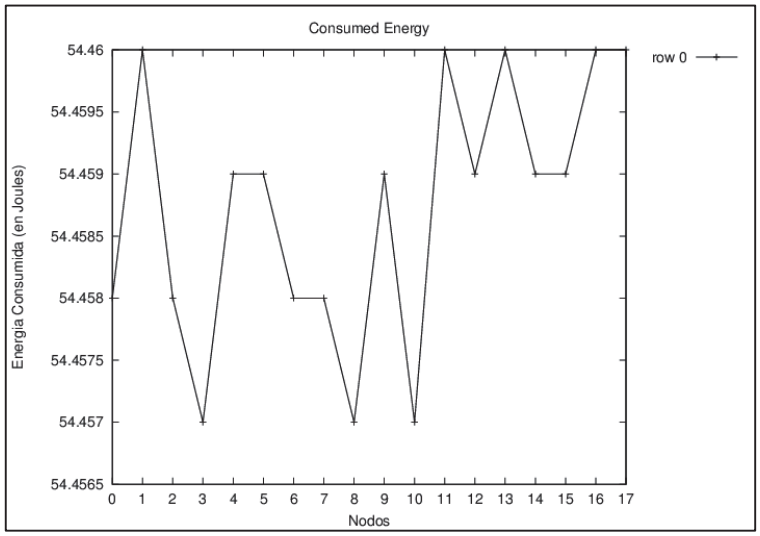


Figura 3-52 Primera etapa de la simulación (ConnectivityMap): Energía consumida por nodo.

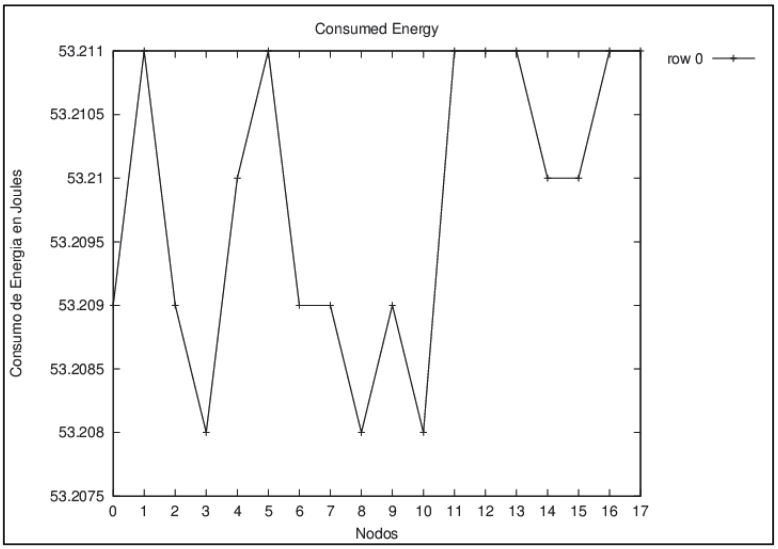


Figura 3-53 Segunda etapa de la simulación (ThroughputTest): Energía consumida por nodo.

En las Figuras 3-52 y Figura 3-53 se observa el consumo de energía por nodo debido al primer y segundo subprocesos de simulación, respectivamente. La WSN consta de motas TelosB desplegadas en el escenario “B” y el TSP se resuelve mediante el método de B&B.

3.4.1.2.1.1.2 Paquetes Fallidos

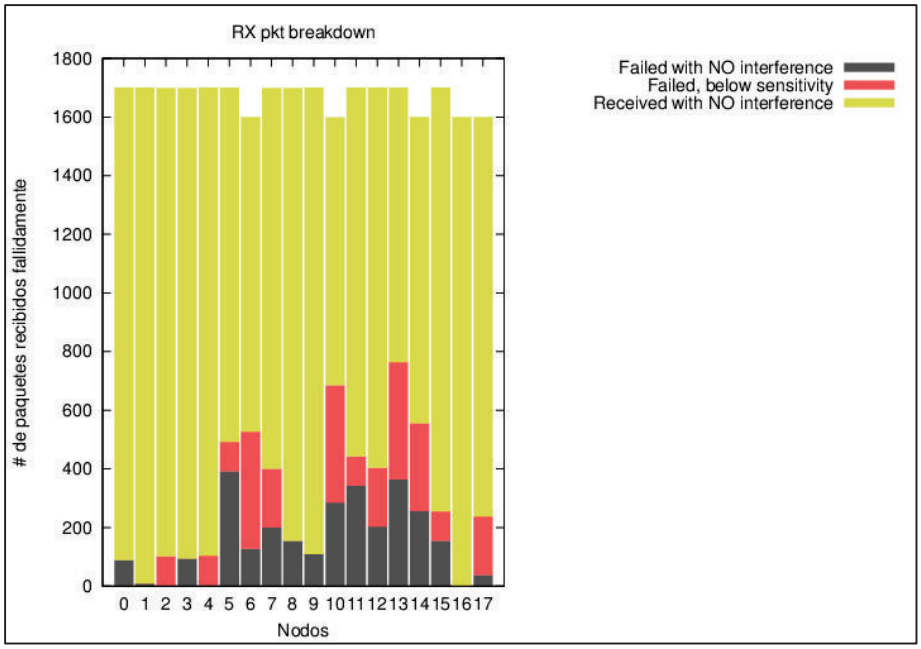


Figura 3-54 Primera etapa de la simulación (ConnectivityMap): Paquetes fallidos.

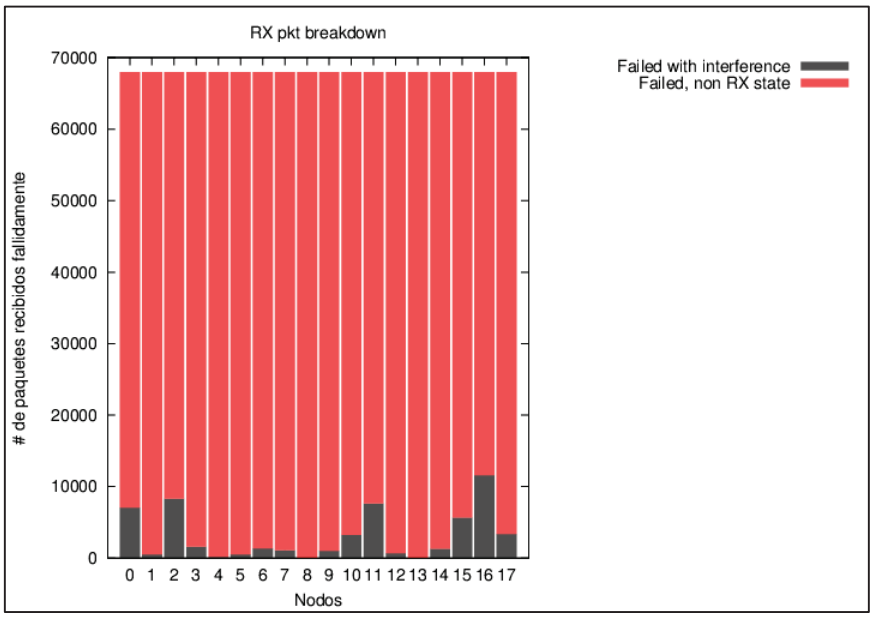


Figura 3-55 Tercera etapa de la simulación (ThroughputTest): Paquetes Fallidos.

En las Figuras 3-54 y 3-55 se observa los paquetes fallidos por nodo debido al primer y segundo subprocesos de simulación, respectivamente. En la Figura 3-55 el número de paquetes fallidos en rojo es debido a que no se especifica el control de acceso al medio. La WSN consta de motas TelosB desplegadas en el escenario “B” y el TSP se resuelve mediante el método de B&B.

3.4.1.2.1.2 Aproximación por MST

3.4.1.2.1.2.1 Energía Consumida

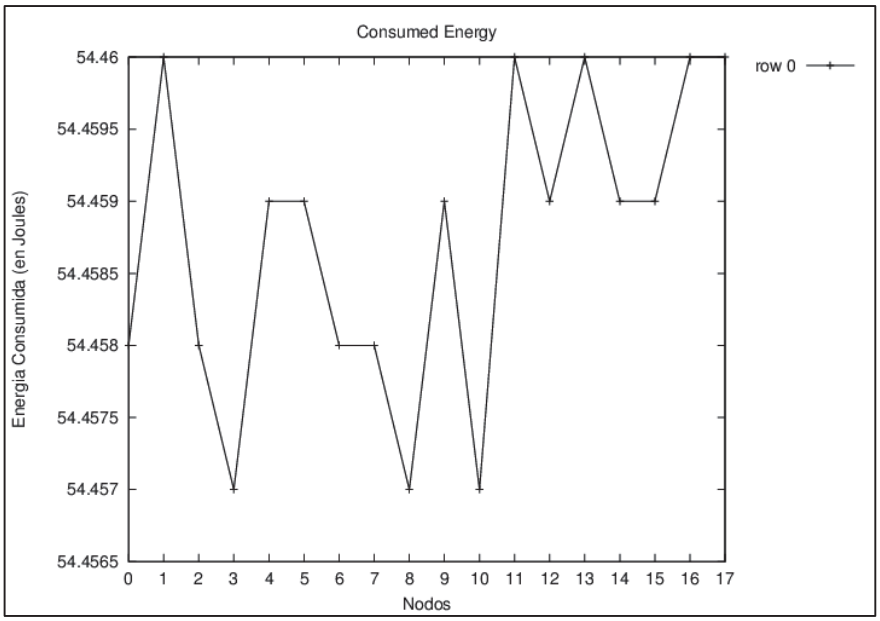


Figura 3-56 Primera etapa de la simulación (ConnectivityMap): Energía consumida por nodo.

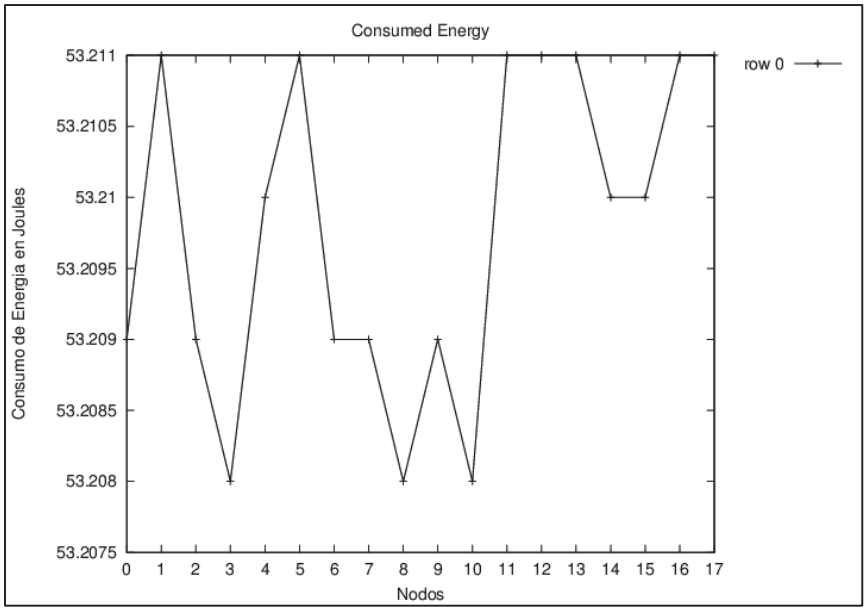


Figura 3-57 Segunda etapa de la simulación (ThroughputTest): Energía consumida por nodo.

En las Figuras 3-56 y 3-57 se observa el consumo de energía por nodo debido al primer y segundo subprocesos de simulación, respectivamente. La WSN consta de motas TelosB desplegadas en el escenario “B” y el TSP se resuelve mediante el método de aproximación por MST.

3.4.1.2.1.2.2 Paquetes Fallidos

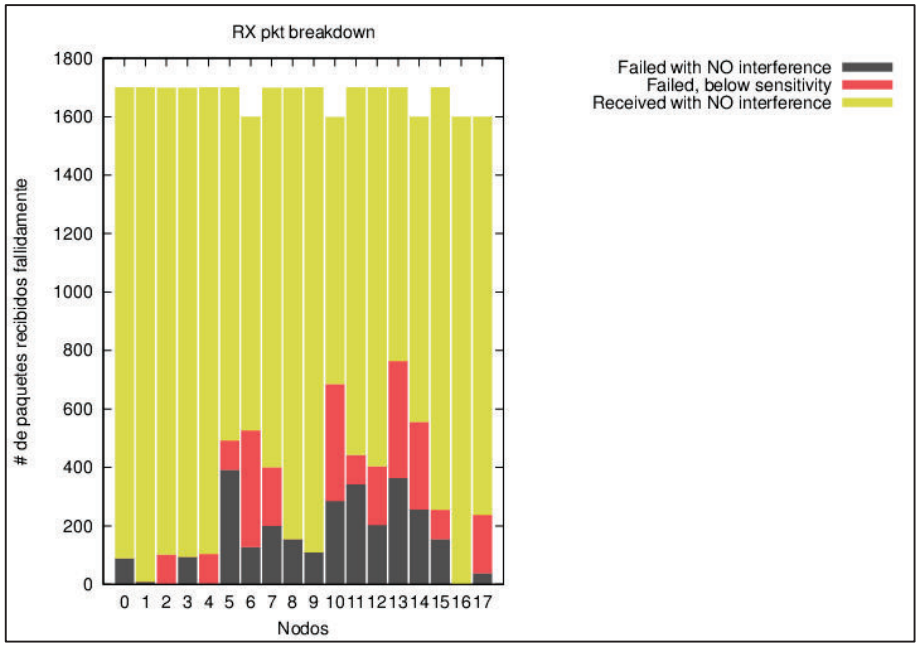


Figura 3-58 Primera etapa de la simulación (ConnectivityMap): Paquetes fallidos.

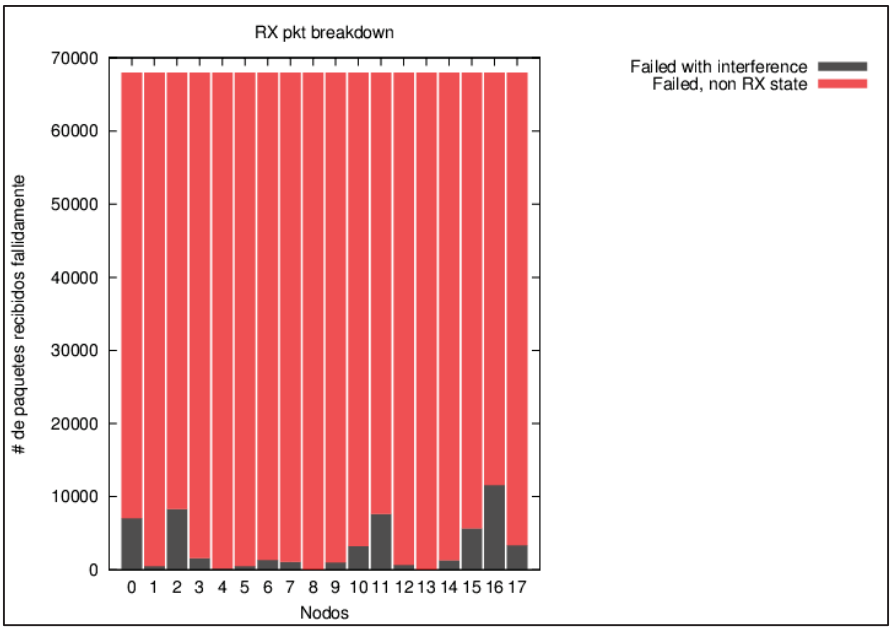


Figura 3-59 Tercera etapa de la simulación (ThroughputTest): Paquetes Fallidos.

En las Figuras 3-58 y 3-59 se observa los paquetes fallidos por nodo debido al primer y segundo subprocesos de simulación, respectivamente. En la Figura 3-59 el número de paquetes fallidos en rojo es debido a que no se especifica el control de acceso al medio. La WSN consta de motas TelosB desplegadas en el escenario “B” y el TSP se resuelve mediante el método de aproximación por MST.

3.4.1.2.2 WSN conformada por nodos sensores Imote2

3.4.1.2.2.1 Branch and Bound

3.4.1.2.2.1.1 Energía Consumida

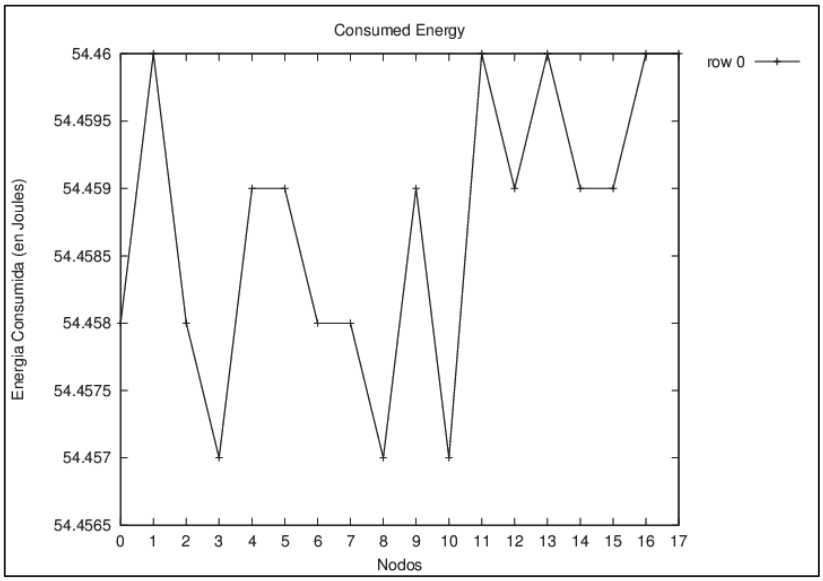


Figura 3-60 Primera etapa de la simulación (ConnectivityMap): Energía consumida por nodo.

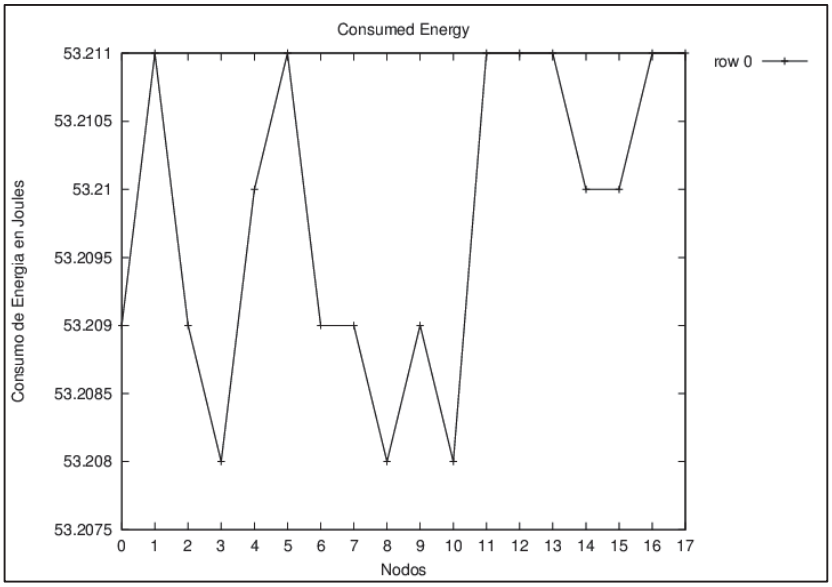


Figura 3-61 Segunda etapa de la simulación (ThroughputTest): Energía consumida por nodo.

En las Figuras 3-60 y 3-61 se observa el consumo de energía por nodo debido al primer y segundo subprocesos de simulación, respectivamente. La WSN consta de motas Imote2 desplegadas en el escenario “B” y el TSP se resuelve mediante el método de B&B.

3.4.1.2.2.1.2 Paquetes Fallidos

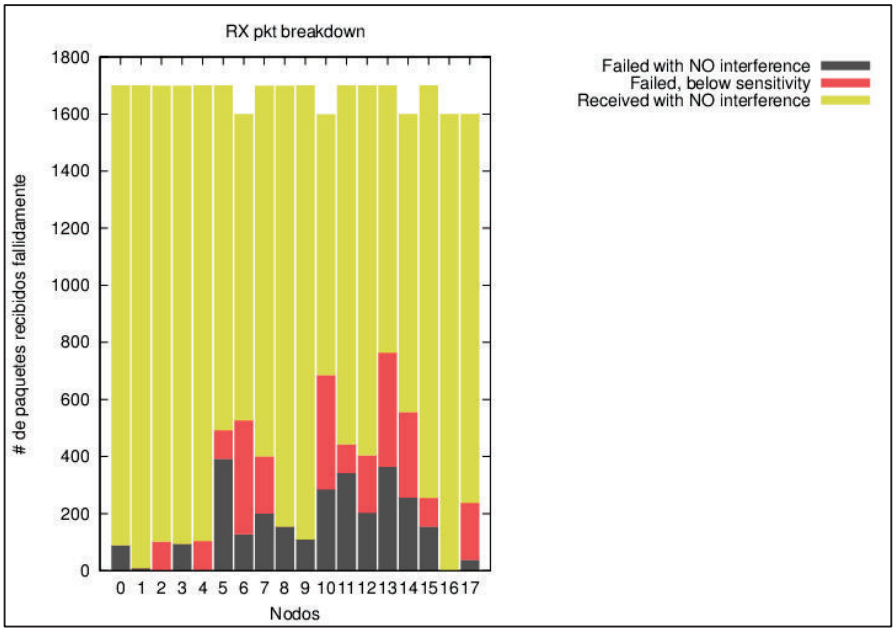


Figura 3-62 Primera etapa de la simulación (ConnectivityMap): Paquetes fallidos.

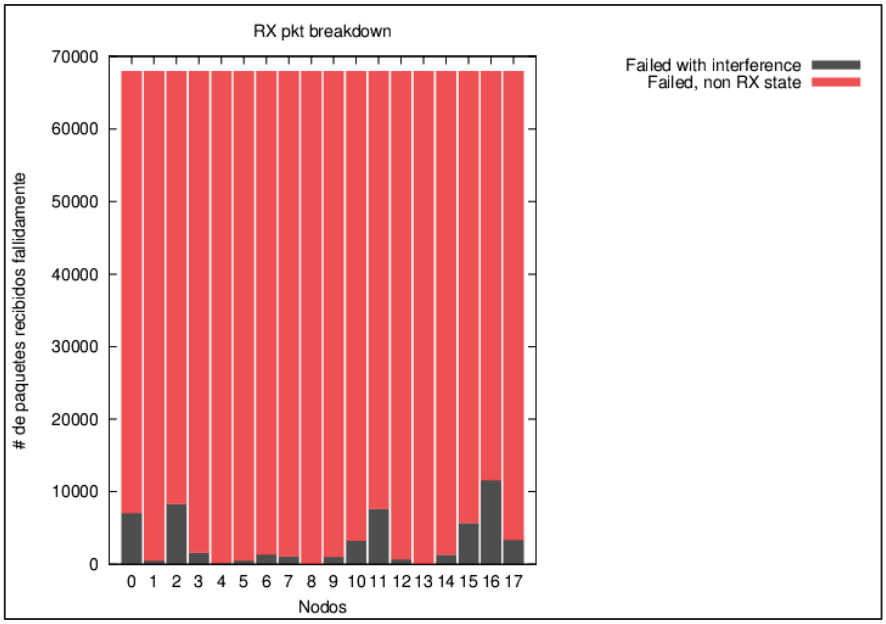


Figura 3-63 Tercera etapa de la simulación (ThroughputTest): Paquetes Fallidos.

En las Figuras 3-62 y 3-63 se observa los paquetes fallidos por nodo debido al primer y segundo subprocesos de simulación, respectivamente. En la Figura 3-63 el número de paquetes fallidos en rojo es debido a que no se especifica el control de acceso al medio. La WSN consta de motas Imote2 desplegadas en el escenario “B” y el TSP se resuelve mediante el método de B&B.

3.4.1.2.2.2 Aproximación por MST

3.4.1.2.2.2.1 Energía Consumida

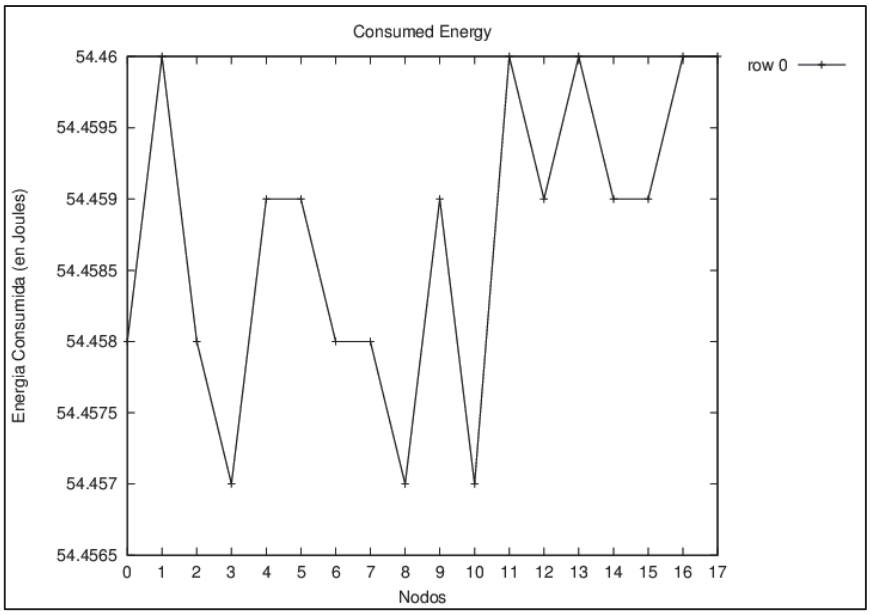


Figura 3-64 Primera etapa de la simulación (ConnectivityMap): Energía consumida por nodo.

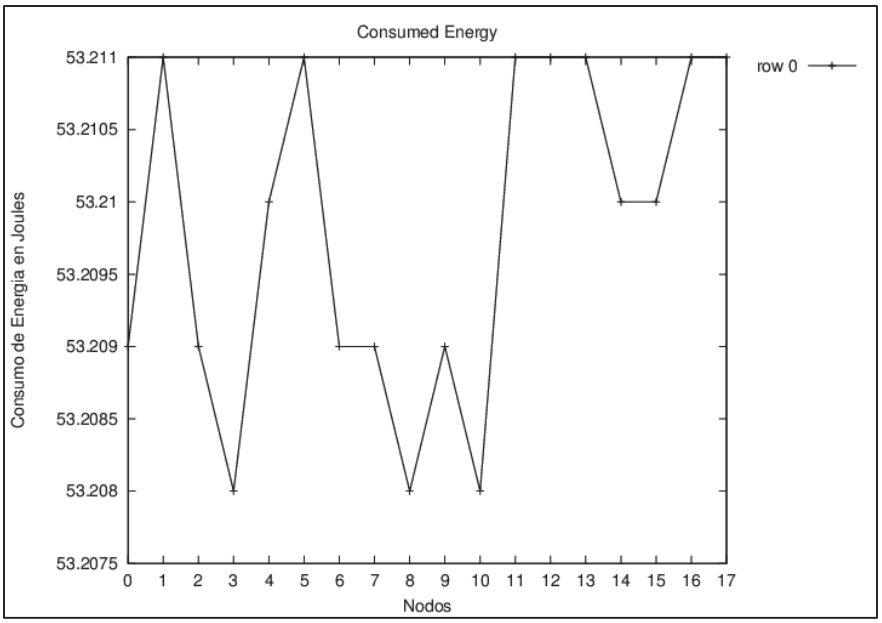


Figura 3-65 Segunda etapa de la simulación (ThroughputTest): Energía consumida por nodo.

En las Figuras 3-64 y 3-65 se observa el consumo de energía por nodo debido al primer y segundo subprocesos de simulación, respectivamente. La WSN consta de motas Imote2 desplegadas en el escenario “B” y el TSP se resuelve mediante el método de aproximación por MST.

3.4.1.2.2.2.2 Paquetes Fallidos

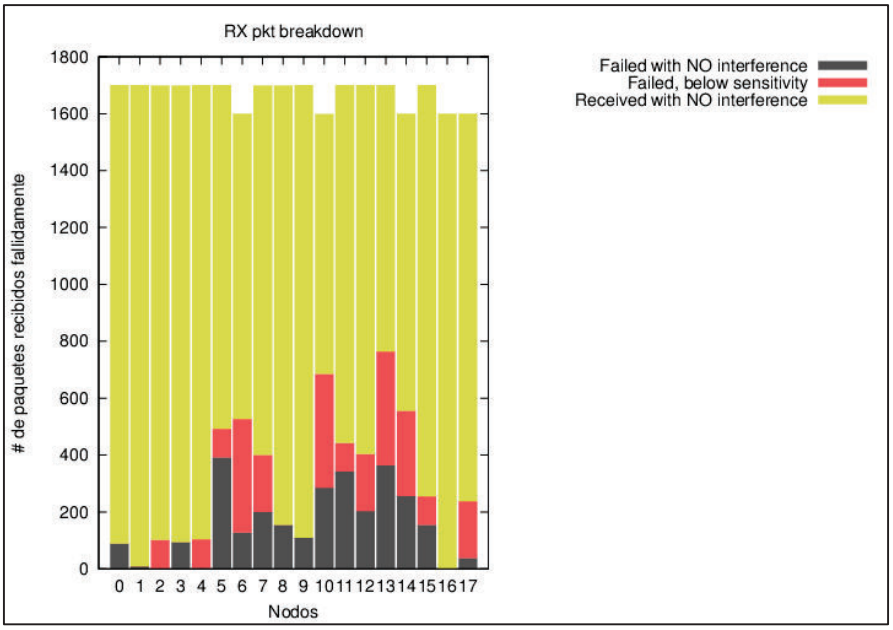


Figura 3-66 Primera etapa de la simulación (ConnectivityMap): Paquetes fallidos.

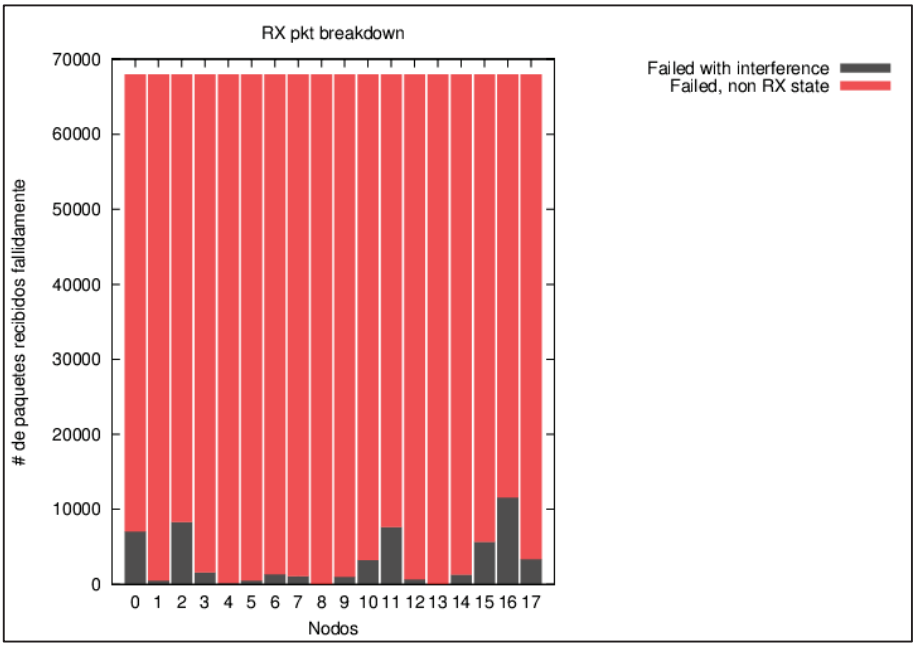


Figura 3-67 Tercera etapa de la simulación (ThroughputTest): Paquetes Fallidos.

En las Figuras 3-66 y 3-67 se observa los paquetes fallidos por nodo debido al primer y segundo subprocesos de simulación, respectivamente. En la Figura 3-67 el número de paquetes fallidos en rojo es debido a que no se especifica el control de acceso al medio. La WSN consta de motas Imote2 desplegadas en el escenario “B” y el TSP se resuelve mediante el método de aproximación por MST.

3.4.1.2.3 WSN conformada por nodos sensores Zolertia

3.4.1.2.3.1 Branch and Bound

3.4.1.2.3.1.1 Energía Consumida

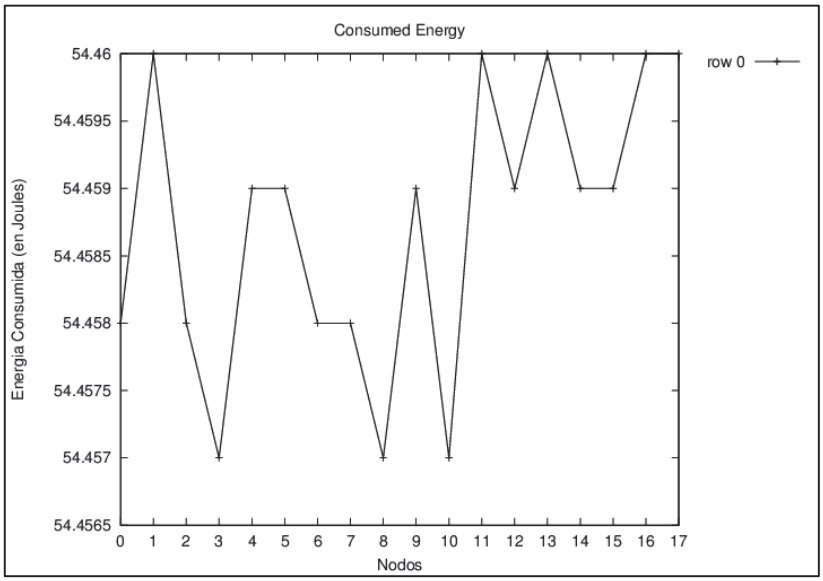


Figura 3-68 Primera etapa de la simulación (ConnectivityMap): Energía consumida por nodo.

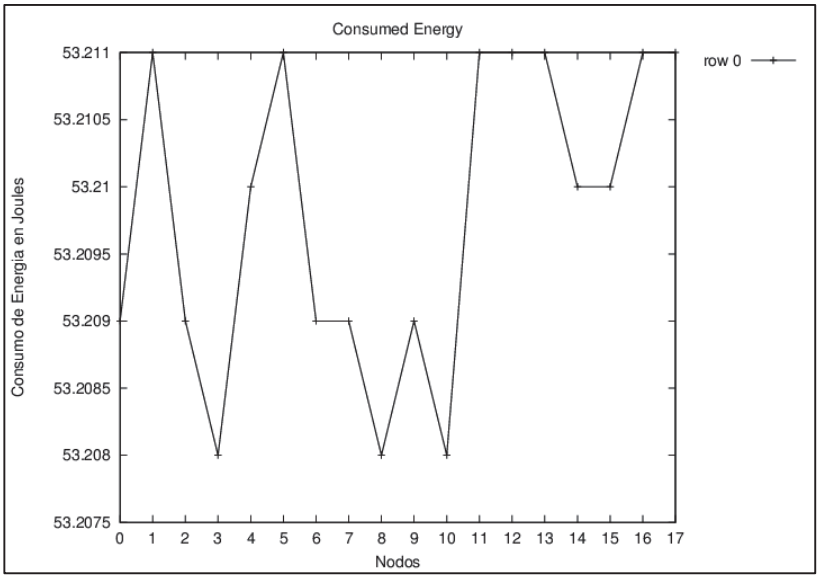


Figura 3-69 Segunda etapa de la simulación (ThroughputTest): Energía consumida por nodo.

En las Figuras 3-68 y 3-69 se observa el consumo de energía por nodo debido al primer y segundo subprocesos de simulación, respectivamente. La WSN consta de motas Zolertia desplegadas en el escenario “B” y el TSP se resuelve mediante el método de B&B.

3.4.1.2.3.1.2 Paquetes Fallidos

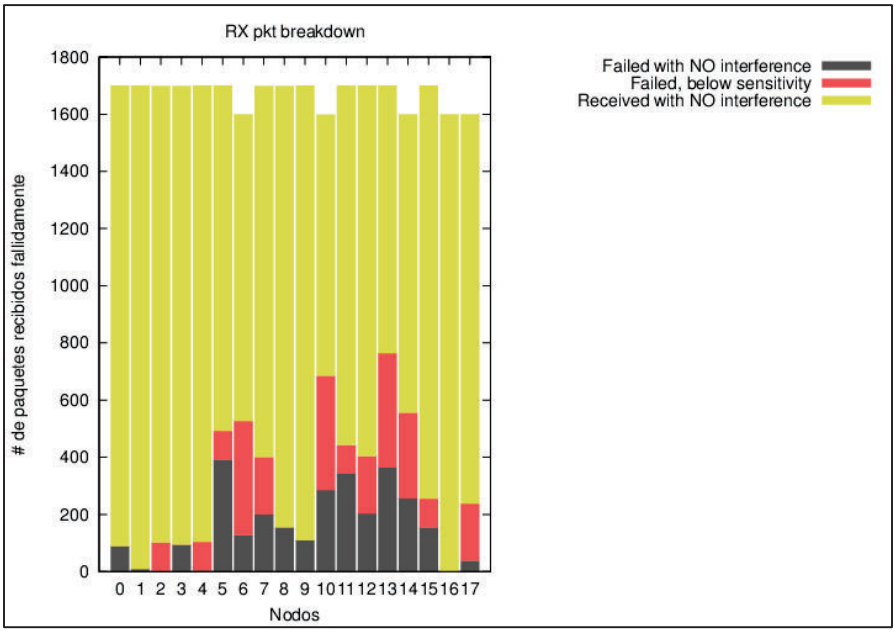


Figura 3-70 Primera etapa de la simulación (ConnectivityMap): Paquetes fallidos.

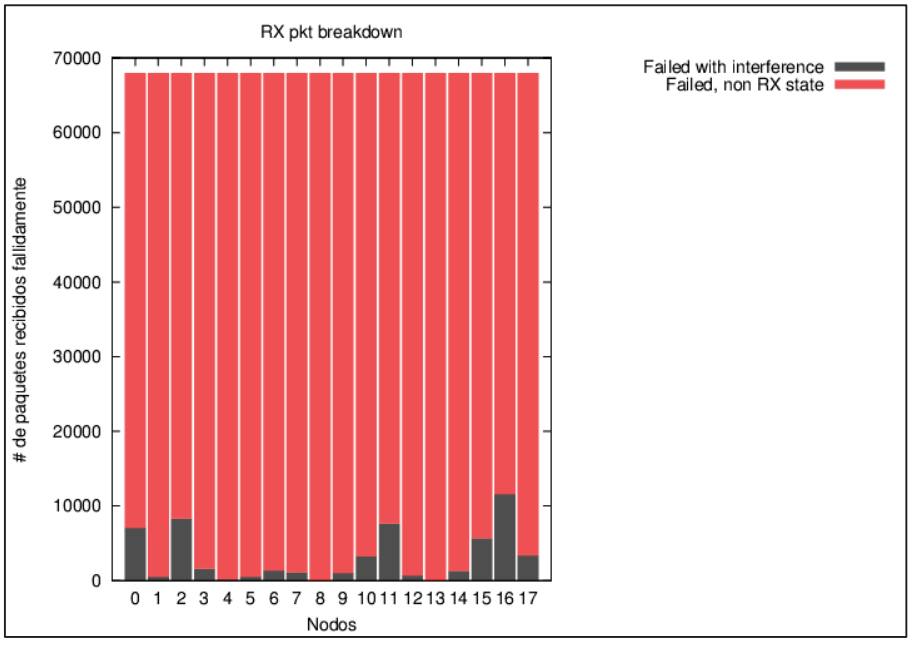


Figura 3-71 Tercera etapa de la simulación (ThroughputTest): Paquetes Fallidos.

En las Figuras 3-70 y 3-71 se observa los paquetes fallidos por nodo debido al primer y segundo subprocesos de simulación, respectivamente. En la Figura 3-71 el número de paquetes fallidos en rojo es debido a que no se especifica el control de acceso al medio. La WSN consta de motas Zolertia desplegadas en el escenario “B” y el TSP se resuelve mediante el método B&B.

3.4.1.2.3.2 Aproximación por MST

3.4.1.2.3.2.1 Energía Consumida

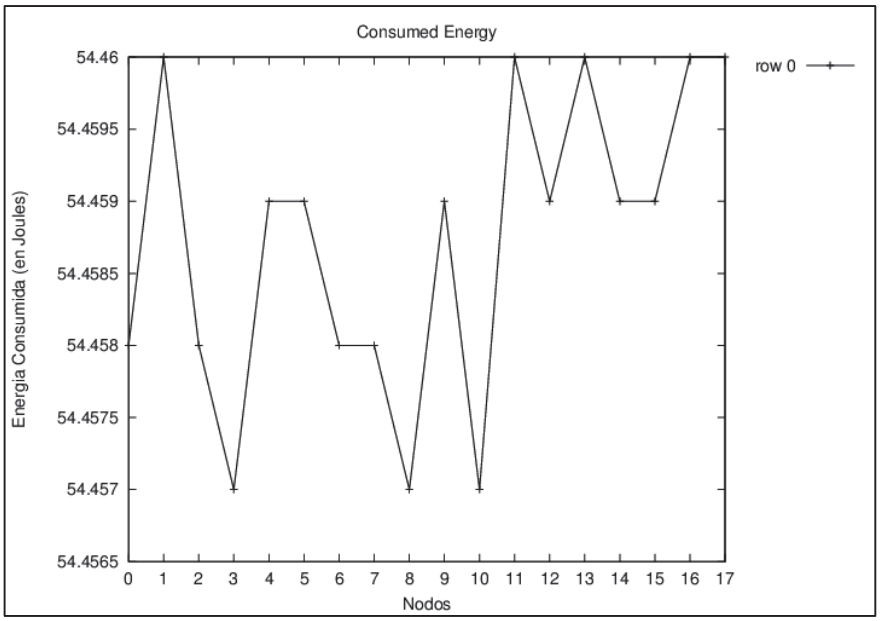


Figura 3-72 Primera etapa de la simulación (ConnectivityMap): Energía consumida por nodo.

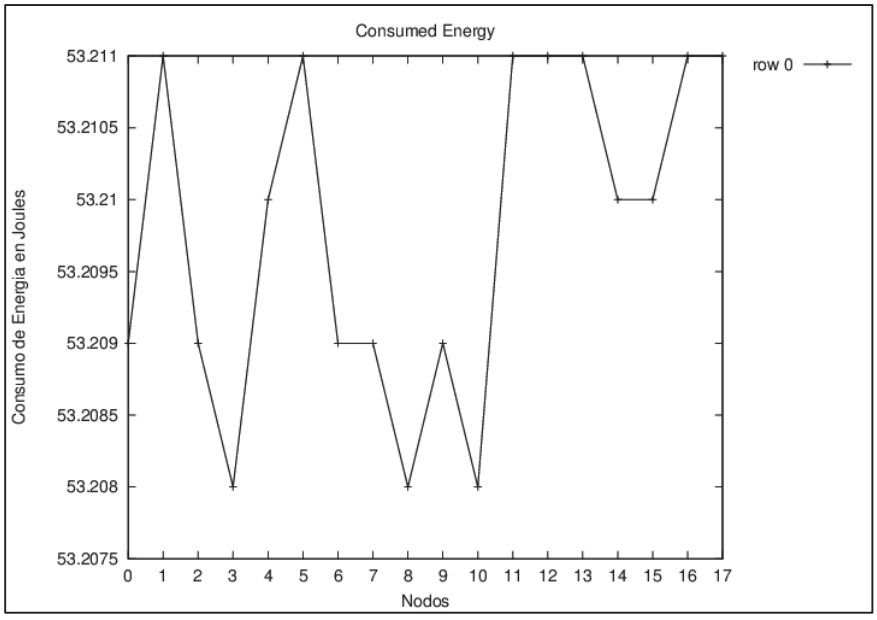


Figura 3-73 Segunda etapa de la simulación (ThroughputTest): Energía consumida por nodo.

En las Figuras 3-72 y 3-73 se observa el consumo de energía por nodo debido al primer y segundo subprocesos de simulación, respectivamente. La WSN consta de motas Zolertia desplegadas en el escenario “B” y el TSP se resuelve mediante el método de aproximación por MST.

3.4.1.2.3.2.2 Paquetes Fallidos

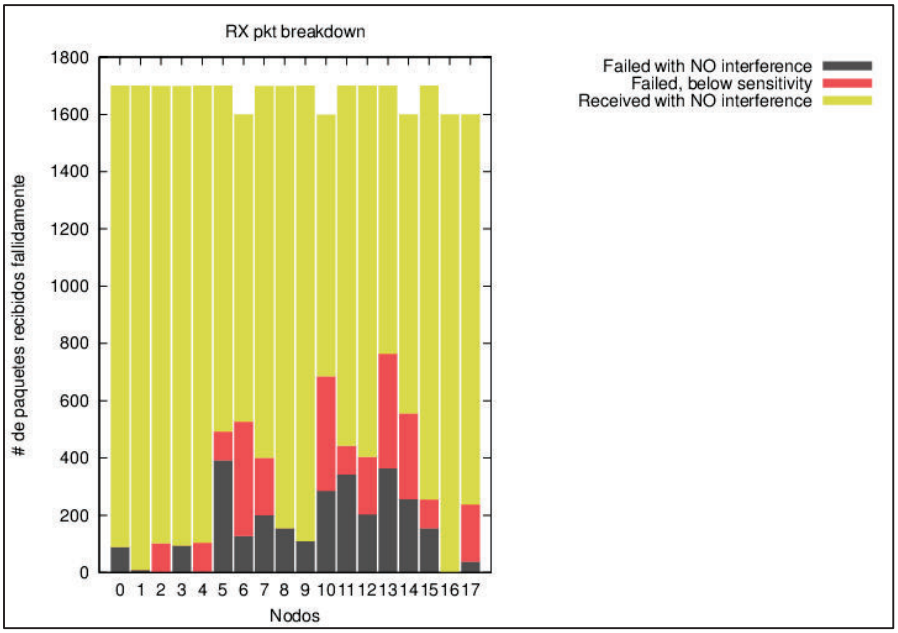


Figura 3-74 Primera etapa de la simulación (ConnectivityMap): Paquetes fallidos.

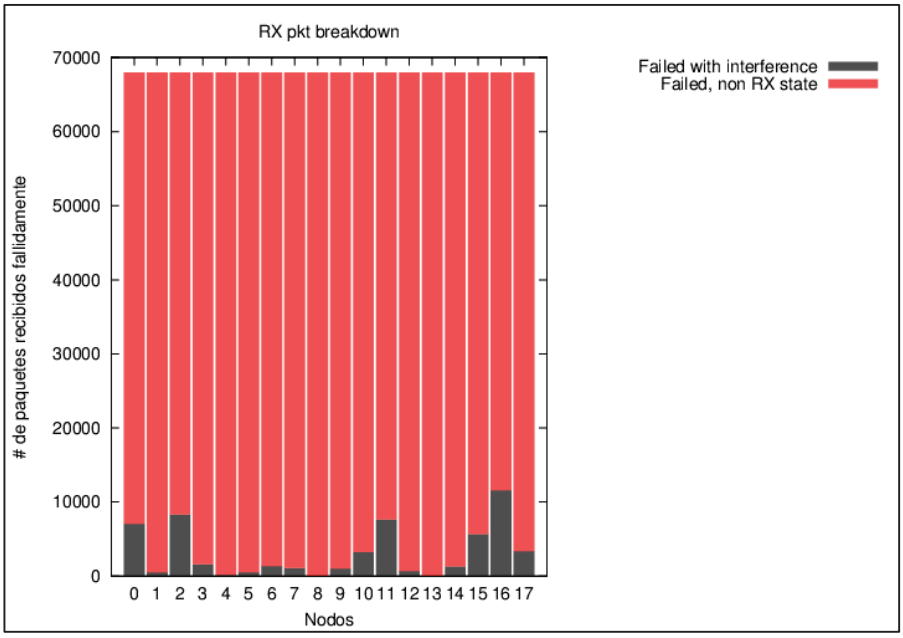


Figura 3-75 Tercera etapa de la simulación (ThroughputTest): Paquetes Fallidos.

En las Figuras 3-74 y 3-75 se observa los paquetes fallidos por nodo debido al primer y segundo subprocesos de simulación, respectivamente. En la Figura 3-75 el número de paquetes fallidos en rojo es debido a que no se especifica el control de acceso al medio. La WSN consta de motas Zolertia desplegadas en el escenario “B” y el TSP se resuelve mediante el método de aproximación por MST.

3.4.2 ESTADÍSTICAS GLOBALES

Al comparar las estadísticas obtenidas, se las ha agrupado para obtener una mejor comprensión de los resultados de las simulaciones. Las cantidades totales presentadas a continuación son la suma de las cantidades obtenidas para cada WSN estudiada. Se ha tomado en cuenta una de las 10 repeticiones de cada caso simulado ya que los datos obtenidos en cada grupo de diez simulaciones son idénticos.

3.4.2.1 Consumo de Energía

3.4.2.1.1 Por Escenario

	Energía Total Consumida [Joules]	Suma de Promedios [Joules]	Suma de Modas [Joules]
Escenario A	11628.198	646.011	646.02
Escenario B	11628.198	646.011	646.02

Tabla 3-6 Consumo de Energía total por Escenario de simulación.

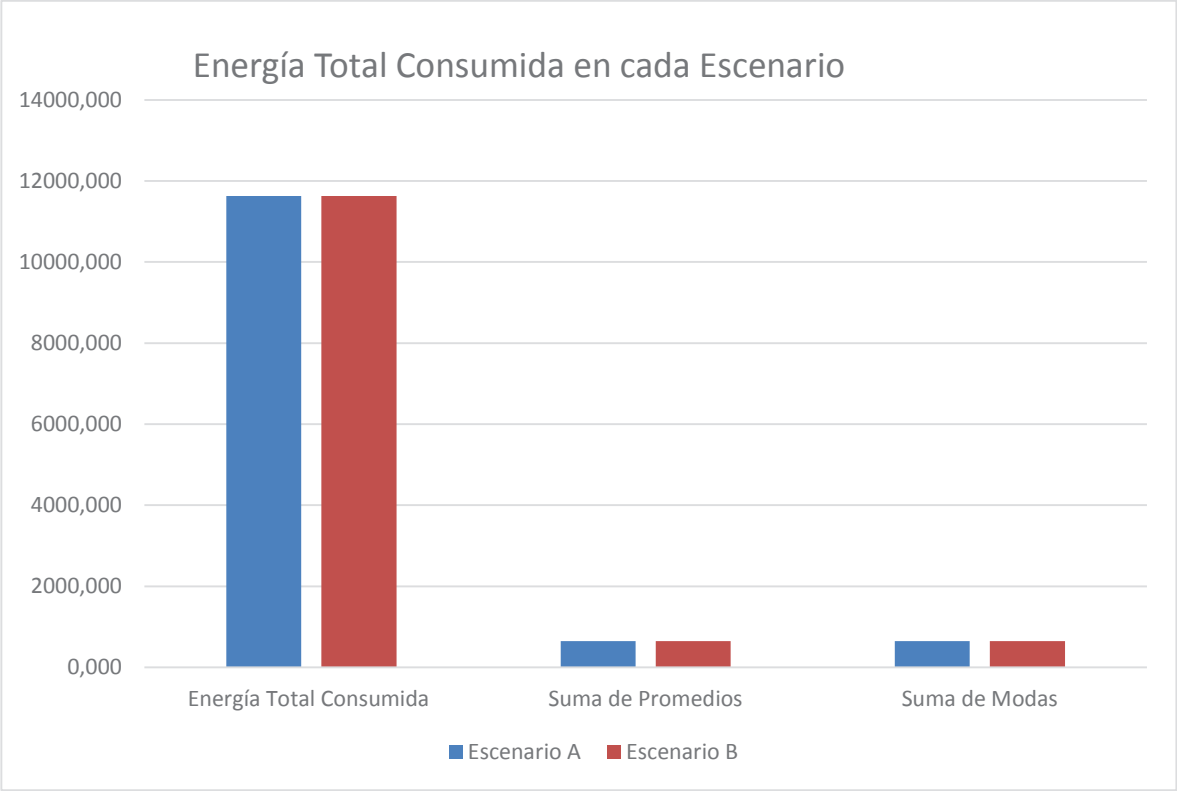


Figura 3-76 Consumo de Energía total por Escenario de simulación.

Para la Tabla 3-6 y la Figura 3-76 se consideran todas las simulaciones para el escenario “A” y para el escenario “B”, entonces se suma el consumo de energía por escenario para obtener la energía total consumida, luego se suman todos los promedios por escenario y finalmente se suman las modas por escenario, es decir, los valores con mayor frecuencia en la distribución de los datos.

3.4.2.1.2 Por Modelo de Mota

	Energía Total Consumida [Joules]	Suma de Promedios [Joules]	Suma de Modas [Joules]
Imote2	7752.132	430.674	430.68
TelosB	7752.132	430.674	430.68
Z1	7752.132	430.674	430.68

Tabla 3-7 Consumo de Energía total por modelo de mota simulada.

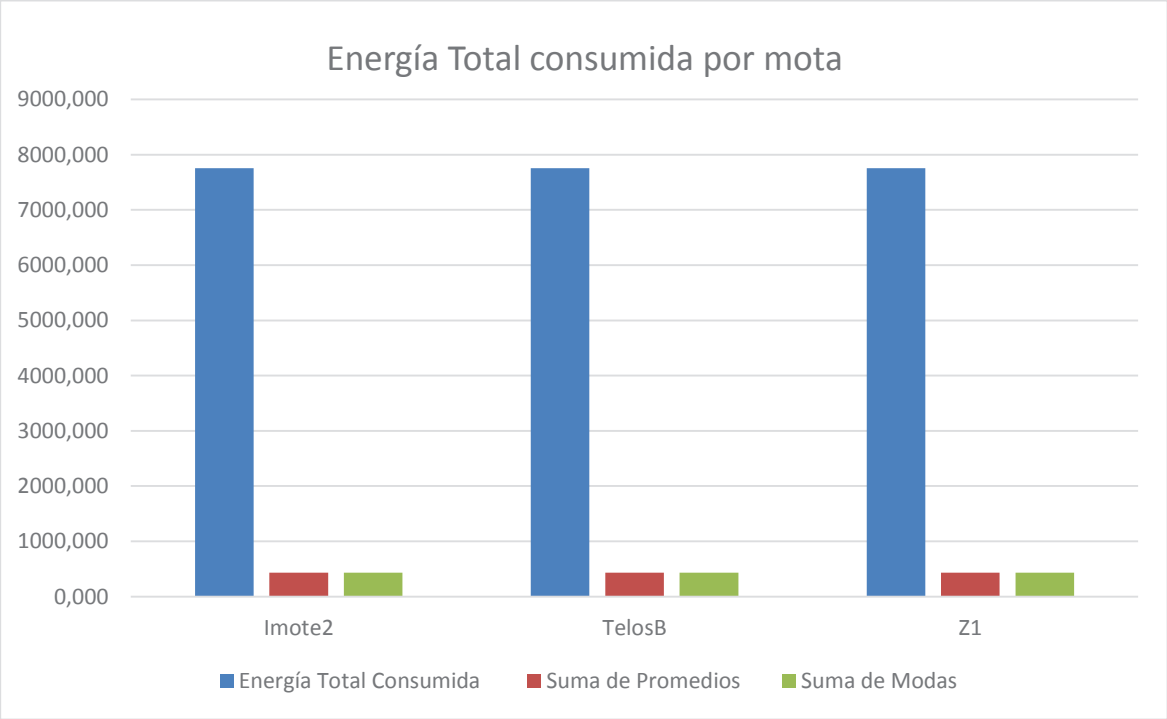


Figura 3-77 Consumo de Energía total por modelo de mota simulada.

En la Tabla 3-7 y en la Figura 3-77 se reúnen todas las simulaciones para los modelos de mota Imote2, TelosB y Zolertia, entonces se suma el consumo de energía por modelo de mota para obtener la energía total consumida, luego se suman todos los promedios de consumo de energía por modelo de mota y finalmente se suman las modas por modelo de mota, es decir, los valores con mayor frecuencia en la distribución de los datos.

3.4.2.1.3 Por Método de Resolución del TSP

	Energía Total Consumida [Joules]	Suma de Promedios [Joules]	Suma de Modas [Joules]
BB	11628.198	646.011	646.02
MST	11628.198	646.011	646.02

Tabla 3-8 Consumo de Energía total por método de resolución del TSP.

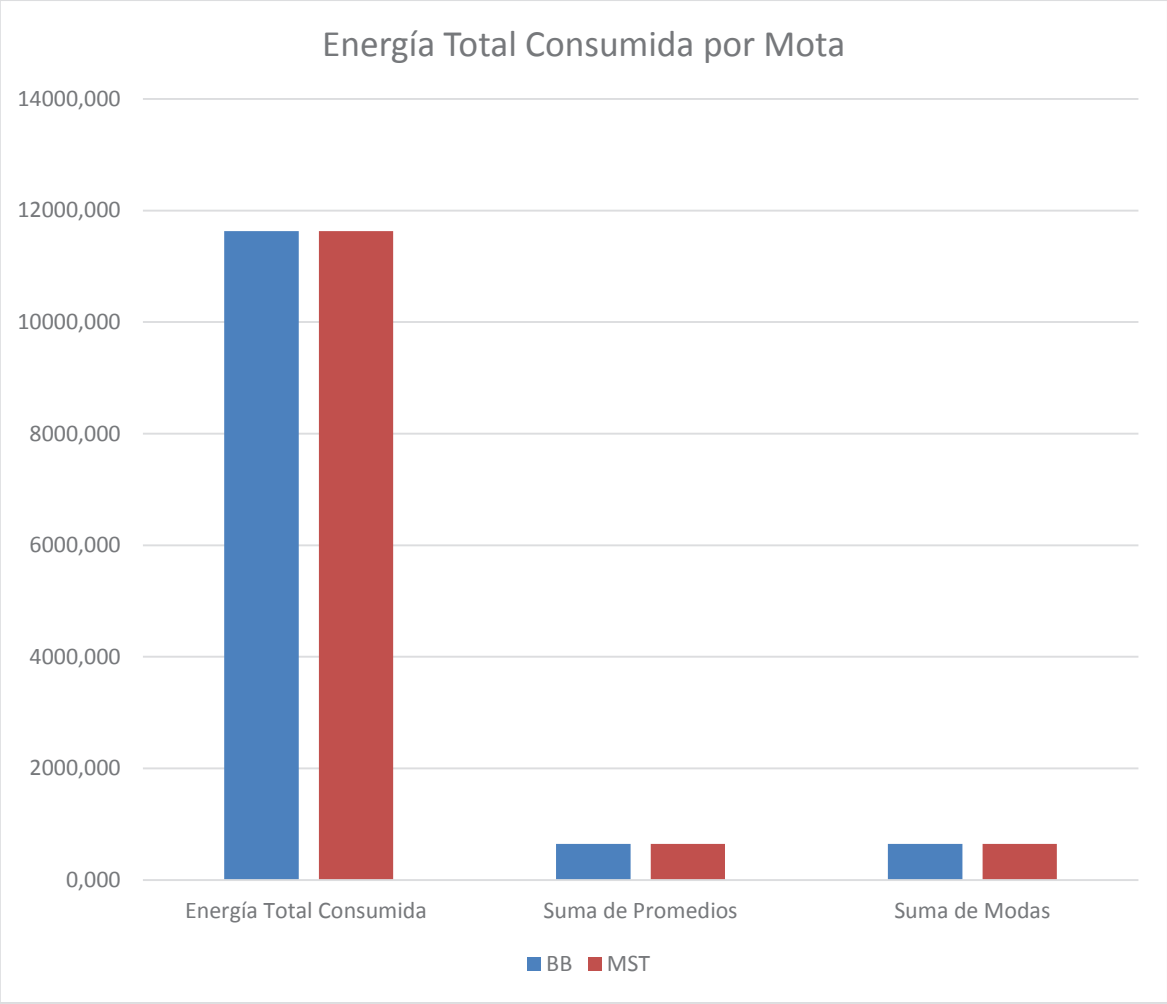


Figura 3-78 Consumo de Energía total por método de resolución del TSP.

Para la Tabla 3-8 y la Figura 3-78 se observan todas las simulaciones por método de resolución del TSP, entonces se suma el consumo de energía por método de resolución del TSP para obtener la energía total consumida, luego se suman todos los promedios de consumo de energía por método de resolución del TSP y finalmente se suman las modas por método de resolución del TSP, es decir, los valores con mayor frecuencia en la distribución de los datos.

3.4.2.2 Desempeño

Debido a que para el presente trabajo se utilizaron parámetros estándar del simulador, no se tiene grandes diferencias en cuanto a los parámetros medidos (esto se deduce también en 3.5.2.1).

Por lo tanto a continuación se grafica el parámetro “Paquetes fallidos debido al Ruido Térmico”, por el motivo de que es el parámetro distintivo respecto al resto de valores obtenidos, además se puede observar que la disposición de los nodos en el área de sensado juega un papel preponderante al momento de las comunicaciones entre los mismos.

3.4.2.2.1 Por Escenario

Paq. Fallidos: Ruido Térmico	Total	Suma de Promedios	Suma de Modas
Escenario A	19494	1083	1044
Escenario B	16920	940	924

Tabla 3-9 Paquetes errados por la presencia de Ruido Térmico en cada Escenario.

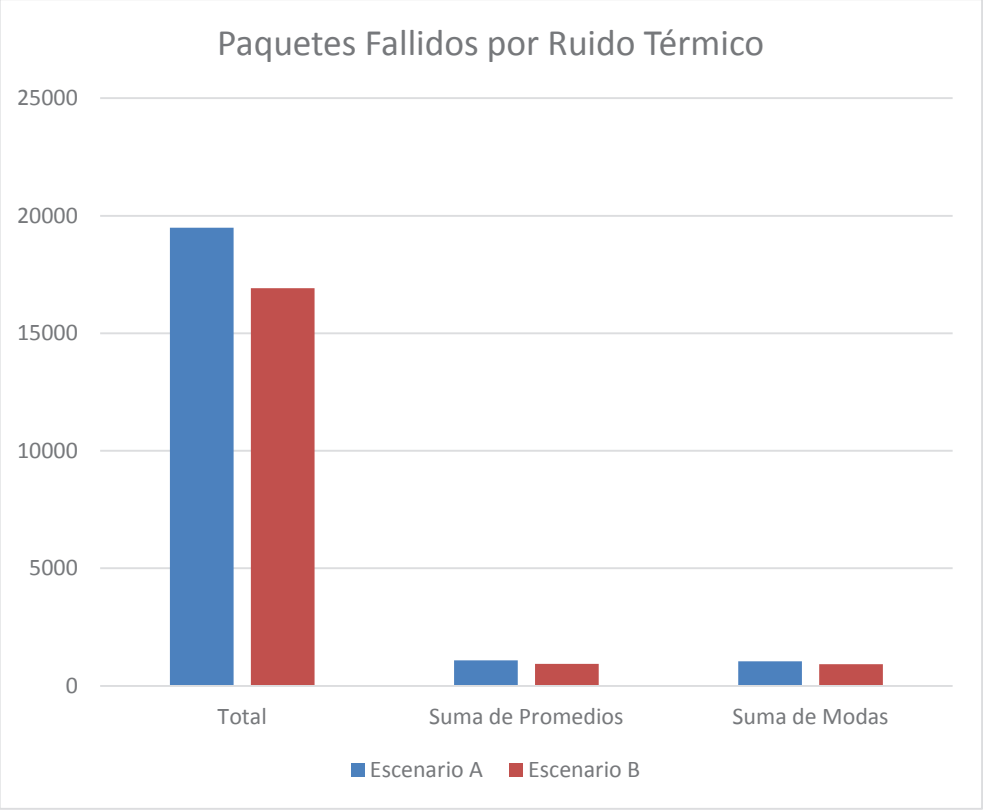


Figura 3-79 Paquetes errados por la presencia de Ruido Térmico en cada Escenario.

Para la Tabla 3-9 y la Figura 3-79 se toma en cuenta todas las simulaciones para el escenario “A” y para el escenario “B”, entonces se suman los paquetes fallidos debido al ruido térmico por escenario para obtener el total, luego se suman todos los promedios por escenario y finalmente se suman las modas por escenario, es decir, los valores con mayor frecuencia en la distribución de los datos.

3.4.2.2.2 Por Modelo de Mota

Paq. Fallidos: Ruido Térmico	Total	Suma de Promedios	Suma de Modas
Imote2	12138	674.3333333	656
TelosB	12138	674.3333333	656
Z1	12138	674.3333333	656

Tabla 3-10 Paquetes errados debido a la presencia de Ruido Térmico por cada modelo de mota.

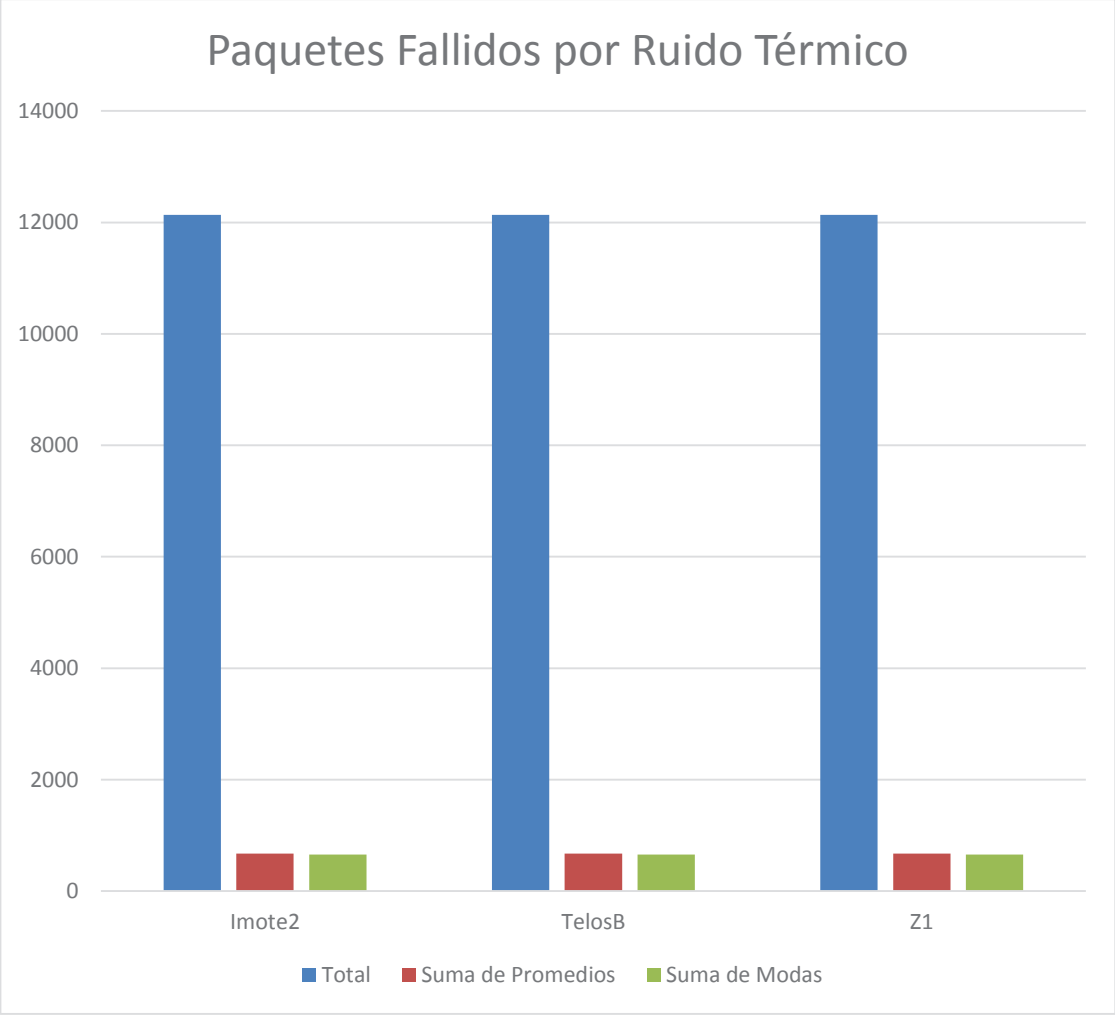


Figura 3-80 Paquetes errados debido a la presencia de Ruido Térmico por cada modelo de mota.

En la Tabla 3-79 y la Figura 3-80 se consideran todas las simulaciones para los modelos de mota Imote2, TelosB y Zolertia, entonces se suma los paquetes fallidos debido al ruido térmico por modelo de mota para obtener el total, luego se suman todos los promedios de los paquetes fallidos debido al ruido térmico por modelo de mota y finalmente se suman las modas por modelo de mota, es decir, los valores con mayor frecuencia en la distribución de los datos.

3.4.2.2.3 Por Método de Resolución del TSP

Paq. Fallidos: Ruido Térmico	Total	Suma de Promedios	Suma de Modas
BB	18207	1011.5	984
MST	18207	1011.5	984

Tabla 3-11 Paquetes errados debido a la presencia de Ruido Térmico por cada método de resolución del TSP.

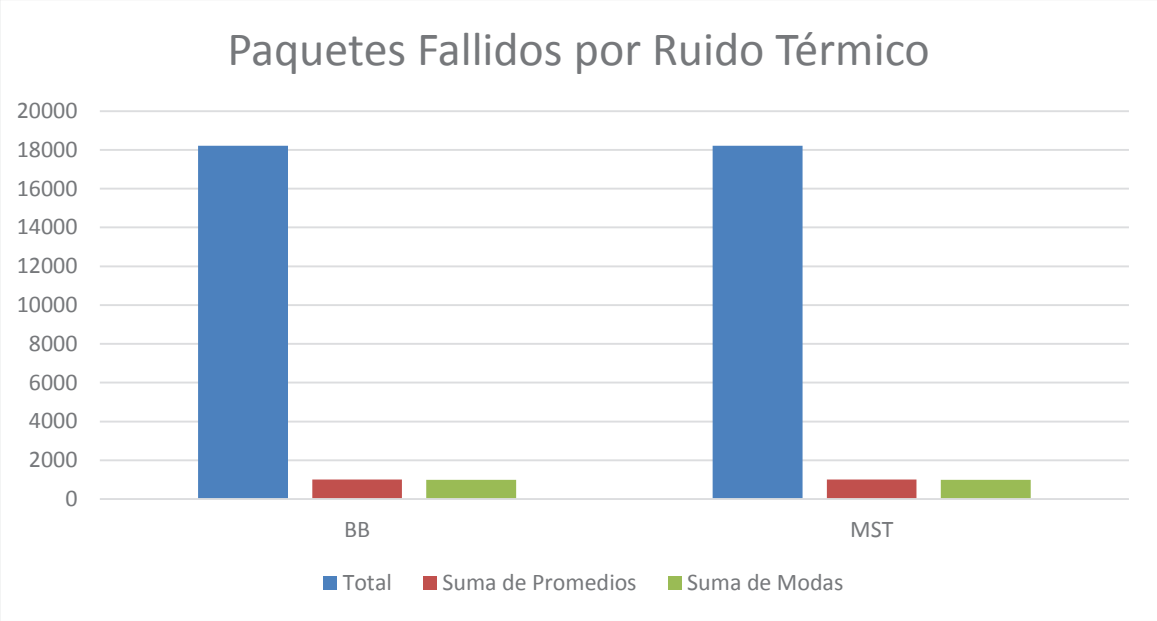


Figura 3-81 Paquetes errados debido a la presencia de Ruido Térmico por cada método de resolución del TSP.

En la Figura 3.81 se agrupan todas las simulaciones por método de resolución del TSP, entonces se suman los paquetes fallidos debido al ruido térmico por método de resolución del TSP para obtener el total, luego se suman todos los paquetes fallidos debido al ruido térmico por método de resolución del TSP y finalmente se suman las modas por método de resolución del TSP, es decir, los valores con mayor frecuencia en la distribución de los datos.

3.5 COMPROBACIÓN DE LA CORRECTA RESOLUCIÓN DEL TSP UTILIZANDO TSPSG

Se compara las resoluciones obtenidas usando los dos métodos de resolución del TSP propuestos, con los resultados conseguidos de la aplicación de código abierto TSPSG [65].

3.5.1 MATRIZ DE PAQUETES RECIBIDOS DESDE EL NODO "I" AL "J"

Se toma un archivo plano (ArchPlanoMatCostsCastalia_jtito) que contiene la cantidad de paquetes recibidos desde el nodo [i], hasta el nodo [j], como se puede apreciar en la Figura 3-82:

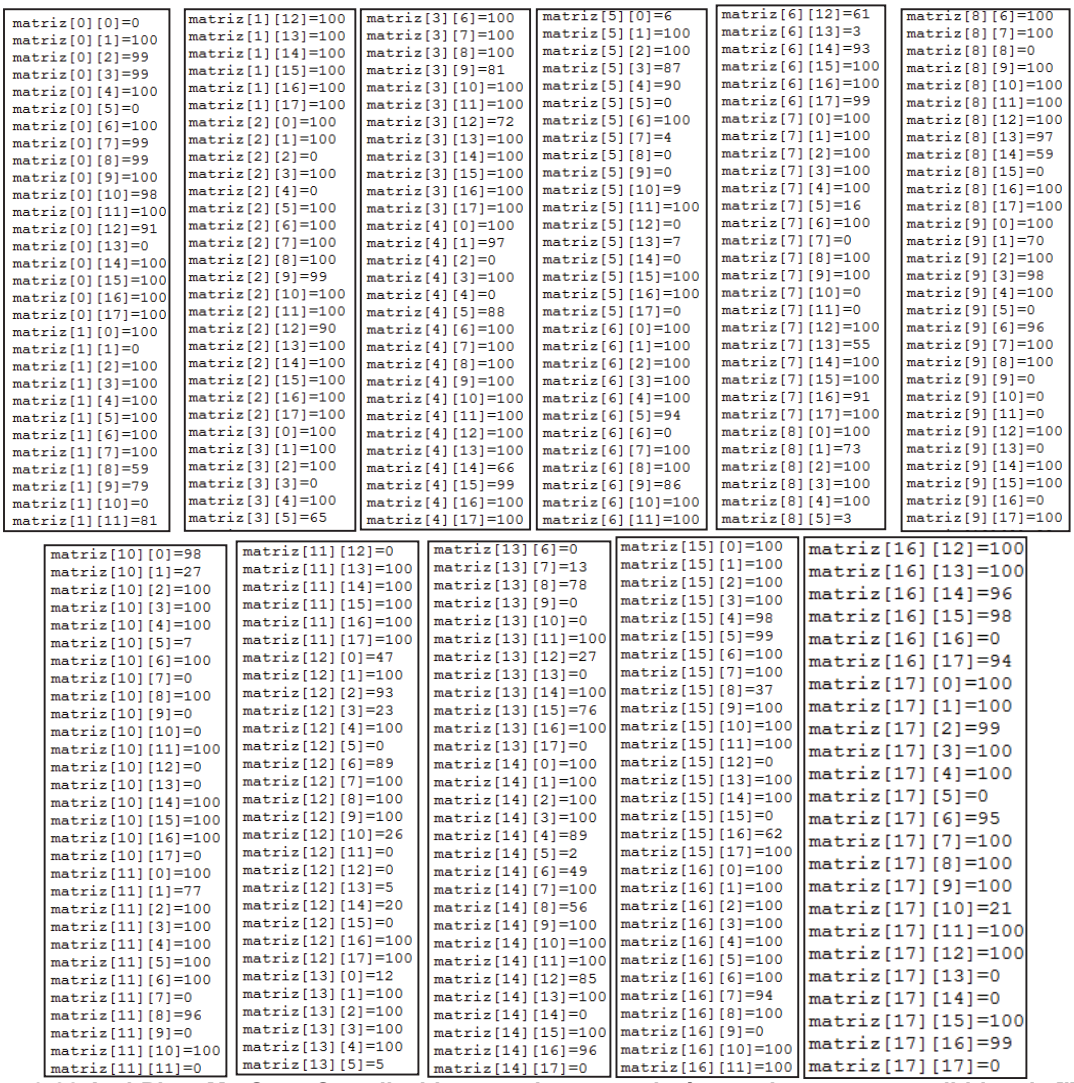


Figura 3-82 ArchPlanoMatCostsCastalia_jtito completo, con el número de paquetes recibidos de [i] a [j].

El máximo número de paquetes recibidos por un nodo es de 100 paquetes, teniendo en cuenta las consideraciones mencionadas en 3.1.2.1.2.1 y 3.1.2.1.2.2, se realiza la conversión de escala implementada en código, pero esta vez se la implementa en Excel, con el objetivo de obtener el costo de cada enlace.

La conversión tiene en cuenta que si se reciben 100 paquetes de 100 paquetes enviados, es porque el costo del enlace es prácticamente nulo, mientras que si se reciben 0 paquetes de 100 enviados se debe a que el enlace es inexistente (de costo infinito).

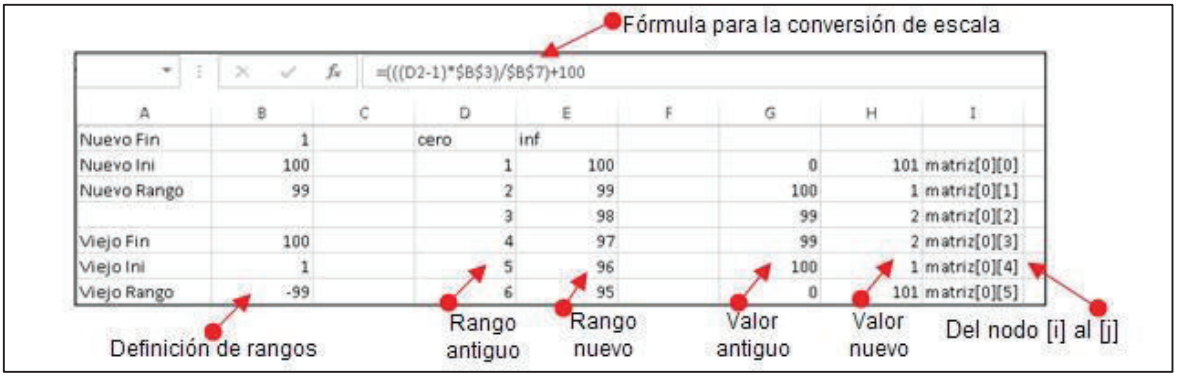


Figura 3-83 Archivo auxiliar en Excel: Se realiza la conversión de rangos para la obtención del costo del enlace a partir del número de paquetes recibidos exitosamente.

En la Figura 3-83 se reemplazan los valores nuevos iguales a 101 por valores “infinitos”, esto antes de copiar la información del costo del enlace convertido, desde el archivo auxiliar de Excel al programa TSPSG.

Además, se debe tener en cuenta que en el programa TSPSG se llama “City” a los nodos, y son numerados del 1 al 18, mientras la numeración propuesta para el presente proyecto es del 0 al 17. Los enlaces infinitos se pueden apreciar con la simbología (---), tal como se puede observar en la Figura 3-84.

```
Optimal path:
  City 1 -> City 2 -> City 3 -> City 4 -> City 5 -> City 7 -
> City 8 -> City 9 -> City 10 -> City 13 -> City 18 -> City
16 -> City 11 -> City 15 -> City 14 -> City 17 -> City 6 ->
City 12 -> City 1
The price is 18 units.
```

Figura 3-84 Tour que resuelve el TSP de prueba, y su costo total.

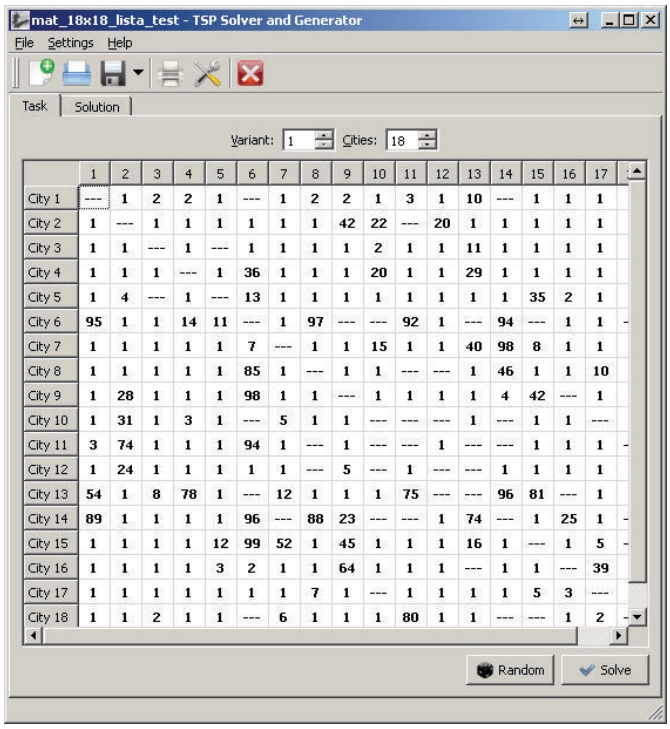


Figura 3-85 Matriz de costos en el programa TSPSG.

A pesar de que el grafo tiene varios enlaces “infinitos” (como se observa en la Figura 3-85), es lo suficientemente conexo como para que el TSP sea resuelto. TSPSG utiliza el método de *Branch and Bound* para resolver el TSP y genera un resumen de los pasos en un archivo de formato “pdf”. A continuación, se muestra solo la parte final del archivo desarrollado, donde el archivo puede ser revisado en el Anexo 9.

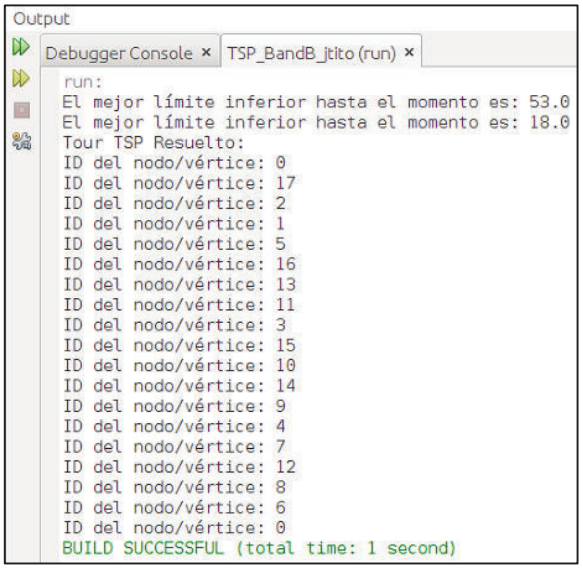


Figura 3-86 Tour y costo del TSP resuelto usando B&B, en el grafo de prueba.

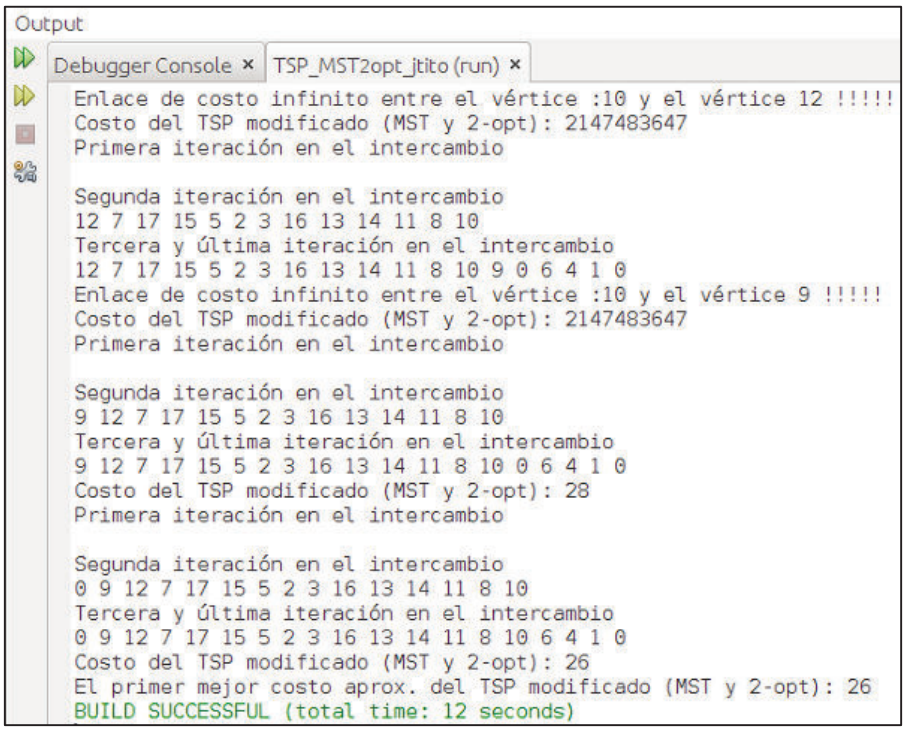


Figura 3-87 Tour y costo del TSP resuelto utilizando la aproximación por MST, en el grafo de prueba.

En la Figura 3-86, se puede observar el TSP fue resuelto exactamente con un costo total de 18 unidades. Al ser el método por MST un método aproximado, el valor mostrado en la Figura 3-87 (26 unidades) como costo total del TSP resuelto, es un valor dentro de los parámetros esperados.

4 CAPÍTULO 4: CONCLUSIONES Y RECOMENDACIONES

En este capítulo se presentan las conclusiones, recomendaciones y aplicaciones futuras obtenidas a lo largo de la elaboración de este proyecto.

4.1 CONCLUSIONES

- A partir del análisis realizado en el presente proyecto de titulación se concluye que para los parámetros expuestos (cómo ejemplo, el número de nodos), el mejor método (de los dos métodos propuestos) para resolver el TSP en una WSN corresponde al método de *Branch and Bound*.

- El método por B&B utilizando el límite inferior de Held-Karp se tarda 1 segundo en resolver el TSP, mientras el método aproximado por el MST tarda 12 segundos en llevar a cabo la misma tarea, por lo tanto a pesar de que B&B es más complejo de implementar, también es más rápido, aunque esto dependerá de las facilidades de programación que se tenga en la WSN específica.

- Con la resolución del TSP se comprueba que la minimización del consumo energético en la WSN se cumple al observarse que cada nodo consume alrededor de 53 Joules, en llevar a cabo la implementación el TSP para todos los casos de WSN planteadas.

- Castalia sobresale entre los simuladores de WSN debido a que puede ser utilizado también para simular las BAN *(Body Area Networks* – Redes de área corporal), tiene una documentación muy detallada, al ser completamente de código abierto puede ser modificado a conveniencia según se requiera según términos afines a las licencias de software libre, existe una comunidad activa planteando/respondiendo inquietudes sobre el simulador. Además, simula

todos los subsistemas que conforman un nodo sensor (subsistemas de sensado, procesamiento, comunicaciones y potencia).

- Debido a que la resolución del TSP debe cumplir con la condición de pasar por todos los nodos antes de regresar al nodo fuente utilizando el camino de menor costo posible, para grafos no muy conexos existe una fase en la resolución del TSP para ambos algoritmos, donde se llega a un punto en el que no se puede escoger otro camino más que el enlace infinito. Por lo tanto se concluye existen WSN en las que el TSP no podrá ser resuelto

- A causa de las permutaciones introducidas por el algoritmo 2-opt, necesarias para obtener la resolución del TSP utilizando el método aproximado por el MST, el desempeño obtenido por este método es muy inferior al desempeño obtenido por el método de B&B.

- La obtención de los resultados idénticos observados en 3.3.4.1 y 3.3.4.2 se origina en que el simulador Castalia fue modificado en sus parámetros más básicos para llegar a obtener las simulaciones planteadas. Es decir, no se realizaron cambios significativos en los parámetros por defecto, ni en el código fuente para los módulos del simulador especificados en 3.1.2.2.1 (especialmente en los módulos de enrutamiento, de control de acceso al medio, de movilidad, de fenómenos físicos y de gestión de recursos).

- Como se puede observar en 3.4.1 y 3.4.5, con la versión 3.2 de Castalia no se obtienen los datos suficientes para calcular la tasa de paquetes recibidos[45], la tasa de paquetes perdidos[46] y la tasa de paquetes reenviados[47] en la tercera etapa de la simulación.

---

[45] *Packets Successful Rate* — PSR.
[46] *Packets Dropped Rate* — PDR.
[47] *Packets Retry Rate* — PRR.

- El Escenario B (Configuración compleja de red con el nodo fuente en el centro), presenta menor ruido térmico que el Escenario A (Configuración compleja de red con el nodo fuente en el extremo izquierdo).

4.2 RECOMENDACIONES

- Con el objetivo de obtener simulaciones más cercanas a la realidad se recomienda la modificación del simulador Castalia en lo que respecta a los parámetros y código fuente de los módulos: de enrutamiento, de control de acceso al medio, de movilidad, de fenómenos físicos y de gestión de recursos.

- Se recomienda, según el nivel de precisión que se necesite por parte del simulador, la modificación de los módulos de Castalia especificados en 3.1.2.2.1, comparando los resultados obtenidos por las modificaciones realizadas al simulador con los resultados obtenidos de una WSN real, de tal forma que del resultado de la comparación mencionada se pueda volver a modificar los módulos de Castalia, consiguiendo cada vez simulaciones más cercanas a la realidad.

- Se recomienda modificar el módulo que gestiona el canal inalámbrico en Castalia, con el objetivo de implementar modelos de atenuación que tengan en cuenta las características propias del entorno en el que se despliega la WSN, para el caso de Ecuador específicamente. De hecho, si se toma en cuenta la recomendación de mejorar los resultados obtenidos con el simulador mediante la comparación con los resultados obtenidos de una WSN real, el modelo de atenuación a implementarse debería tener en cuenta el entorno del laboratorio o del lugar donde se despliegue la WSN en cuestión.

- Cualquiera de las topologías mencionadas en 1.1.4.3 podría ser transformada en una topología del tipo malla, teniendo en cuenta lo explicado en torno a la figura 1-22, por lo tanto se recomienda el uso de enlaces de costo “infinito”

para transformar el grafo original en un grafo completo, teniendo en cuenta que la WSN debe ser lo más conexa posible.

- En 1.1.4.1.4 se sugiere que debido a que los nodos sensores son limitados en términos de procesamiento, almacenamiento y consumo de potencia, los protocolos de la capa de transporte deberían explotar las capacidades colaborativas de estos nodos sensores y mover la inteligencia al nodo fuente. Por lo tanto se recomienda la resolución del TSP debería ser relacionada con el nodo fuente, y en el mejor de los casos, al dispositivo de mayor capacidad de computo asociado a la WSN.

- La difusión de información podría ser mejorada si en vez de enviar paquetes hacia todos los nodos, se enviase un solo paquete que recorra la WSN como lo propone el TSP (atravesando cada nodo una sola vez, para regresar al nodo fuente utilizando el camino de menor costo).

- Se recomienda que si se va a aplicar el método de resolución del TSP usando la aproximación por el MST en una WSN, se tenga muy en cuenta el tamaño de la WSN, debido a que si se trabaja con una WSN de alrededor de 5 nodos sensores no es necesario utilizar otro algoritmo para mejorar el costo total del TSP obtenido (en el presente trabajo se utilizó el algoritmo 2-opt).

- Los inconvenientes suscitados en las WSN, por la caída de nodos, podrían ser mitigados con el diseño de un protocolo de comunicaciones que realice un sondeo de la red cada cierto tiempo. La exploración de la WSN recurriría a un *broadcast*[48] típico en una primera etapa para evaluar el costo de cada enlace y posteriormente en cada nuevo sondeo requerido, utilizaría alguno de los métodos de resolución del TSP en la WSN (el método escogido dependerá de

[40] Difusión de información en una red de comunicaciones.

la aplicación puntual) para no volver a utilizar el *broadcast* innecesariamente, inundando la red de paquetes.

- Se podría modificar el protocolo *Direct Diffusion*[49] para que incluya la resolución del TSP, ya sea para difundir información por toda la WSN o para resolver TSP en segmentos de la WSN que incluyan a los nodos interesados en la información difundida.

- Si en un futuro los nodos sensores se vuelven más robustos en cuanto a poder de procesamiento y consumo energético, se podría investigar la resolución del TSP utilizando el poder de cómputo de toda la red. Por ejemplo las soluciones parciales de *Branch and Bound* podrían ser evaluadas de manera distribuida por cada uno de los nodos que conforman la WSN, o se podría usar el algoritmo de Borůvka para construir el MST encargando a cada nodo de la WSN la construcción de los supernodos para posteriormente resolver el TSP de forma aproximada.

- Según la aplicación lo requiera, la WSN podría ser dividida a conveniencia en clústeres, y resolver el TSP en cada uno de los clústeres.

4.3 APLICACIONES FUTURAS

- Actualmente la comunidad en torno al simulador Castalia sigue trabajando en la implementación del protocolo LEACH[50], basta con revisar las últimas inquietudes planteadas en el foro del simulador [75] para comprobarlo. Adicionalmente, existen nuevos protocolos que funcionan en base al protocolo

---

[41] Protocolo de enrutamiento centrado en los datos (*data-centric*) en el cual se difunde mensajes considerados importantes para la WSN y solo los nodos interesados responden. Es así que se van creando caminos entre el nodo fuente y los nodos interesados en la información difundida [6].

[42] *Low-energy adaptive clustering hierarchy* (Jerarquia de agrupamiento adaptativa de baja energía), es un protocolo basado en agrupamiento (*clustering*) que utiliza la rotación aleatoria de grupos locales de estaciones base (*cluster-heads*) para eventualmente distribuir la carga de energía entre los sensores en la red [93].

LEACH, de estos protocolos en [80] se mencionan: LEACH-A, LEACH-B, LEACH-C, LEACH-Cell, LEACH-E, LEACH-EEE, LEACH-F, LEACH-K, LEACH-M, LEACH-Multihop, LEACH-S, LEACH-TL y LEACH-V.

- El Dr. CongDuc Pham [81], continua con sus investigaciones basadas en WSN e IoT[51], varias de sus investigaciones se las puede encontrar en [81], siendo la transmisión de audio y video sobre WSN [82] uno de los temas que sobresalen debido a que usa VideoSense (Un modelo de simulación de sensores de imagen en OMNeT++/Castalia).

[43] *Internet of Things* (Internet de las cosas)

REFERENCIAS BIBLIOGRÁFICAS

[1] I. Khan, F. Belqasmi, R. Glitho, N. Crespi, M. Morrow y P. Polakos, «Wireless Sensor Network Virtualization: Early Architecture and Research Perspectives,» *IEEE Network,* p. 104, 2015.

[2] A. M. Ortiz, *Técnicas de enrutamiento inteligente para redes de sensores inalámbricas,* Albacete: Universidad de Castilla-La Mancha, 2011, pp. 38-37.

[3] G. Tirado, El problema del viajante con múltiples pilas, Madrid: Universidad Complutense de Madrid, 2011, p. 2.

[4] Clay Mathematics Institute, «P vs NP Problem,» [En línea]. Available: http://www.claymath.org/millennium-problems/p-vs-np-problem. [Último acceso: 6 Junio 2015].

[5] L. Farrugia, Computer Science, Technology and Applications : Wireless Sensor Networks, Hauppauge: Nova Science Publishers, Inc., 2011, p. 47.

[6] O. Jumira y S. Zeadally, FOCUS Series, Volume 1 : Energy Efficiency in Wireless Networks, Somerset: John Wiley & Sons, 2013, p. 29.

[7] L. Farrugia, Computer Science, Technology and Applications : Wireless Sensor Networks, Hauppauge,: Nova Science Publishers, Inc., 2011, p. 137.

[8] U. Colesanti y S. Santini, *A Performance Evaluation Of The Collection Tree Protocol Based On Its Implementation For The Castalia Wireless Sensor Networks Simulator,* Zurich: ETH Zurich, 2010, p. 2.

[9] Libelium Comunicaciones Distribuidas, «Mote Runner,» [En línea]. Available: http://www.libelium.com/development/waspmote/examples/?cat=mote-runner. [Último acceso: 2016 06 15].

[10] S. K. Sarkar, Wireless Sensor and Ad Hoc Networks Under Diversified Network Scenarios, Norwood: ARTECH HOUSE, 2012, p. 133.

[11] J. M. Linares Arenas, Simulación e implementación de una red de sensores inalámbrica multisalto para la medición de consumo energético en un edificio., Cataluña: Universidad Politécnica de Cataluña, 2014, pp. 3-5.

[12] Z. Guan, A Reliability Evaluation of Wireless Sensor Network Simulator: Simulation vs. Testbed, Auckland: UNITEC, 2011, pp. 8-9.

[13] A. M. Ortiz, *Técnicas de enrutamiento inteligente para redes de sensores inalámbricas,* Albacete: Universidad de Castilla-La Mancha, 2011, p. 10.

[14] Z. Guan, *A Reliability Evaluation of Wireless Sensor Network Simulator: Simulation vs. Testbed,* Auckland: UNITEC, 2011, pp. 6-7.

[15] S. Chen, M. Coolbeth, H. Dinh, . Y.-A. Kim y B. Wang, *Data Collection with Multiple Sinks in Wireless Sensor Networks,* Connecticut: University of Connecticut, 2009.

[16] I. Akyildiz y M. Can Vuran, Wireless Sensor Networks, Singapur: Wiley, 2010, pp. 2-15.

[17] W. Dargie y C. Poellabauer, Fundamentals of Wireless Sensor Networks (Theory and Practice), Singapur: Wiley, 2010, pp. 47-51.

[18] J. M. Linares Arenas, Simulación e implementación de una red de sensores inalámbrica multisalto para la medición de consumo energético en un edificio., Cataluña: Universidad Politécnica de Cataluña, 2014, p. 6.

[19] I. Akyildiz y M. Can Vuran, Wireless Sensor Networks, Singapur: Wiley, 2010, pp. 2-5.

[20] Tangient LLC, «Xbee Home,» 2016. [En línea]. Available: https://xbee.wikispaces.com. [Último acceso: 2016 Junio 16].

[21] u-blox, «GSM/GPRS,» 2016. [En línea]. Available: https://www.u-blox.com/en/gsmgprs. [Último acceso: 2016 Junio 16].

[22] J. M. Linares Arenas, *Simulación e implementación de una red de sensores inalámbrica multisalto para la medición de consumo energético en un edificio.,* Cataluña: Universidad Politécnica de Cataluña, 2014, pp. 9-12.

[23] D. Gascón, «802.15.4 vs ZigBee,» 28 Abril 2009. [En línea]. Available: http://www.libelium.com/802-15-4-vs-zigbee/. [Último acceso: 2016 Junio 16].

[24] I. Akyildiz y M. Can Vuran, Wireless Sensor Networks, Singapur: Wiley, 2010, p. 5.

[25] A. M. Ortiz, *Técnicas de enrutamiento inteligente para redes de sensores inalámbricas,* Albacete: Universidad de Castilla-La Mancha, 2011, pp. 18-19.

[26] I. Akyildiz y M. Can Vuran, Wireless Sensor Networks, Singapur: Wiley, 2010, pp. 5-7.

[27] V. Voloshin, Introduction to graph theory, Nueva York: Nova, 2009, pp. 1-7.

[28] E. Coto, Algoritmos Básicos de Grafos, Caracas: Universidad Central de Venezuela, 2003, p. 4.

[29] V. Voloshin, Introduction to graph theory, Nueva York: Nova, 2009, pp. 14-15.

[30] V. Voloshin, Introduction to graph theory, Nueva York: Nova, 2009, p. 17.

[31] Stack Exchange Inc, «Difference between connected vs strongly connected vs complete graphs [closed],» Alexandru Moșoi, 25 Noviembre 2009. [En línea]. Available: http://mathoverflow.net/questions/6833/difference-between-

connected-vs-strongly-connected-vs-complete-graphs. [Último acceso: 17 Junio 2016].

[32] I. Cicekli, «Graphs Spring 2015CS202 - Fundamental Structures of Computer Science II1 Initially prepared by Dr. Ilyas Cicekli; improved by various Bilkent CS202 instructors.,» 2015. [En línea]. Available: http://slideplayer.com/slide/4319050/. [Último acceso: 17 Junio 2016].

[33] E. Coto, Algoritmos Básicos de Grafos, Caracas: Universidad Central de Venezuela, 2003, pp. 6-9.

[34] M. Martinez-Rangel, *Una revisión del Problema de Calendarizar las Tareas en un Taller de Trabajo (Job Shop Scheduling Problem) mediante el Modelo de Grafos Disyuntivo y el Modelo de Programación Entera Mixta,* México D.F.: Universidad Autónoma del Estado de México, 2009.

[35] E. Coto, Algoritmos Básicos de Grafos, Caracas: Universidad Central de Venezuela, 2003, p. 23.

[36] L. Yifu, «Traveling Salesman Problem (TSP). An Example A truck needs to leave node 1, visit each of the other nodes one and only one time, and back to node 1 –What.,» Diciembre 2015. [En línea]. Available: http://slideplayer.com/slide/8783733/. [Último acceso: 19 Junio 2016].

[37] G. Gutin y A. Punnen, The traveling salesman problem and its variations, Nueva York: Kluwer Academic Publisher, 2004, pp. 3-4.

[38] E. Horowitz, S. Sahni y S. Anderson-Freed , «CHAPTER 61 CHAPTER 6 GRAPHS All the programs in this file are selected from Ellis Horowitz, Sartaj Sahni, and Susan Anderson-Freed “Fundamentals of Data.,» Junio 2015. [En línea]. Available: http://slideplayer.com/slide/3349604/. [Último acceso: 19 Junio 2016].

[39] D. Davendra, Traveling Salesman Problem, Theory and Applications, Rijeka: InTech, 2010, pp. 1-17.

[40] T. Ikikardes, «Routing Algorithms for Safety Critical Wireless Sensor Network,» Swiss Federal Institute of Technology Zurich, Zurich, 2006.

[41] M. A. Dallaali, *Network Design and Optimization (Slide pack 6),* Melbourne: The University of Melbourne, 2013.

[42] S. Ramos, «Heurísticas y Problemas Combinatorios,» Universidad de Buenos Aires, [En línea]. Available: http://webcache.googleusercontent.com/search?q=cache:http://materias.fi.uba.ar/7114/Docs/ApunteHeuristicas.pdf. [Último acceso: 16 05 2016].

[43] E. Coto, Algoritmos Básicos de Grafos, Caracas: Universidad Central de Venezuela, 2003, pp. 18-20.

[44] T. Ikikardes, «Routing Algorithms for Safety Critical Wireless Sensor Network,» Swiss Federal Institute of Technology Zurich, Zurich, 2006.

[45] D. Eppstein, «ICS 161: Design and Analysis of Algorithms,» Universidad de California, Irvine, 06 02 1996. [En línea]. Available: https://www.ics.uci.edu/~eppstein/161/960206.html. [Último acceso: 16 05 2016].

[46] E. Correa, «BORUVKA PPT VIDEO,» 23 Noviembre 2011. [En línea]. Available: https://www.youtube.com/watch?v=ZLfSD_1L3zg. [Último acceso: 20 Junio 2016].

[47] J. G. A. Figueroa, «Algorithms and More,» 02 Abril 2012. [En línea]. Available: https://jariasf.wordpress.com/2012/04/02/disjoint-set-union-find/. [Último acceso: 28 Noviembre 2016].

[48] Zolertia, «Mainpage:Contiki Lesson 0,» Sourceforge, [En línea]. Available: http://zolertia.sourceforge.net/wiki/index.php/Mainpage:Contiki_Lesson_0. [Último acceso: 2016 05 16].

[49] SmartSantander, «TelosB,» SmartSantander, 30 11 2012. [En línea]. Available: http://smartsantander.eu/wiki/index.php/Main/TelosB. [Último acceso: 2016 05 16].

[50] K. Lorincz, «IMote2 Installation Instructions,» Intel Research, Berkeley, 14 06 2005. [En línea]. Available: http://www.eecs.harvard.edu/~konrad/projects/imote2Camera/IMote2-Installation-Instructions.html. [Último acceso: 16 05 2016].

[51] J. Straub, C Programming: Data structures and Algorithms, Washington D.C.: University of Washington, 2006.

[52] I. Podsechin, «EURAXESS,» 2008. [En línea]. Available: http://www.euraxess.fi/Tiedostot/Tiedostot/Viksu/Viksu%202008/IgorPodsechin.pdf. [Último acceso: 16 05 2016].

[53] A. Paterson, «Approximating Travelling Salesman Problem Solution With A Minimum Spanning Tree In Java,» Depth Last, 3 Abril 2016. [En línea]. Available: http://depthlast.com/2016/04/03/approximating-travelling-salesman-problem-solution-with-a-minimum-spanning-tree-in-java/. [Último acceso: 12 06 2016].

[54] R. Simha, «The Traveling Salesman Problem (TSP),» George Washington University, [En línea]. Available: https://www.seas.gwu.edu/~simhaweb/champalg/tsp/tsp.html. [Último acceso: 11 05 2016].

[55] A. Paterson, «Performing 2-OPT Updates To TSP Solution Approximation In Java,» Depth Last, 6 Abril 2016. [En línea]. Available:

http://depthlast.com/2016/04/06/performing-2-opt-updates-to-tsp-solution-approximation-in-java/. [Último acceso: 12 Junio 2016].

[56] L. D. Applegate, E. R. Bixby, V. Chvátal y J. W. Cook, The Traveling Salesman Problem, A Computational Study, New Jersey: Princeton University Press, 2006, pp. 41-42.

[57] C. Floudas, Nonlinear and Mixed-Integer Optimization : Fundamentals and Applications, Oxford: Oxford University Press, USA, 1995, pp. 98-107.

[58] Erik-Jan, «Calculating the Held Karp Lower bound For The Traveling Salesman(TSP),» Stack Overflow, 02 07 2014. [En línea]. Available: http://stackoverflow.com/questions/22985590/calculating-the-held-karp-lower-bound-for-the-traveling-salesmantsp. [Último acceso: 11 05 2016].

[59] C. Valenzuela y A. Jones, *Estimating the Held-Karp lower bound for the geometric TSP,* 2001.

[60] H. Yeo, *Course Note: IEOR 251 – Facility Design and Logistics,* 2005.

[61] J.-L. Goffin, «Subgradient optimization in nonsmooth optimization (including the Soviet revolution),» *Documenta Mathematica,* vol. Volumen extra ISMP, pp. 277 - 290, 2012.

[62] M. Held, P. Wolfe y H. Crowder, «Validation of subgradient optimization,» *Mathematical Programming,* nº 6, pp. 62-88, 1973.

[63] OsmAnd, «osmand,» [En línea]. Available: https://code.google.com/archive/p/osmand/. [Último acceso: 16 05 2016].

[64] Stack Overflow, «Optimized TSP algorithms,» [En línea]. Available: http://stackoverflow.com/questions/7159259/optimized-tsp-algorithms. [Último acceso: 16 05 2016].

[65] O. Serdiuk, «TSPSG TSP Solver and Generator,» 2014. [En línea]. Available: http://tspsg.info/. [Último acceso: 12 Junio 2016].

[66] N. G. Bueno, *Simulador de redes vehiculares mediante integración de motores de videojuegos y simulador de redes,* Cartagena: Universidad Politécnica de Cartagena, 2015, pp. 12-13.

[67] OpenSim Ltd., «OMNeT++,» 2015. [En línea]. Available: https://omnetpp.org/. [Último acceso: 10 06 2016].

[68] M. E. Deschamps Espinosa, *Modelado de Mecanismos de Transición a IPv6,* Cholula: Universidad de las Américas Puebla, 2007.

[69] V. Mainanwal, «Castalia :- Simulator Installation steps with OMNET++ in ubuntu 14.04,» 30 Enero 2015. [En línea]. Available: http://vikashmainanwal.blogspot.com/2015/01/castalia-simulator-installation-steps.html. [Último acceso: 2016 Junio 2016].

[70] D. C. Herrera, *Evaluación de redes de sensores inalámbricos mediante el Simulador OMNeT++,* Valencia: Universidad Politécnica de Valencia, 2014, pp. 11-12.

[71] A. Boulis, *Castalia: A simulator for Wireless Sensor Networks and Body Area Networks,* Sydney: NICTA, 2011, p. 26.

[72] Z. Guan, *A Reliability Evaluation of Wireless Sensor Network Simulator: Simulation vs. Testbed,* Auckland: UNITEC, 2011.

[73] M. Stehlík, *Comparison of Simulators for Wireless Sensor Networks,* Brno: MASARYK UNIVERSITY, 2011, pp. 57-58.

[74] Research Gate, «How can I calculate position of each node in omnet++?,» 11 Mayo 2015. [En línea]. Available:

https://www.researchgate.net/post/How_can_I_calculate_position_of_each_node_in_omnet. [Último acceso: 2016 Junio 11].

[75] Google Groups, «Castalia Simulator,» [En línea]. Available: https://groups.google.com/forum/#!forum/castalia-simulator. [Último acceso: 11 Junio 2016].

[76] Crunchify, «Shell Script: Append TimeStamp to file name,» 15 12 2014. [En línea]. Available: http://crunchify.com/shell-script-append-timestamp-to-file-name/. [Último acceso: 07 06 2016].

[77] W. J. Shotts, «Flow Control - Part 2,» LinuxCommand.org, [En línea]. Available: http://linuxcommand.org/wss0120.php. [Último acceso: 07 06 2016].

[78] A. Paterson, «Alexander Paterson,» 2016. [En línea]. Available: http://alexanderpaterson.com/. [Último acceso: 12 06 2016].

[79] G. Latorre, «Setters & Getters,» Programación II, 22 Marzo 2010. [En línea]. Available: http://gl-epn-programacion-ii.blogspot.com/2010/03/setters-getters.html. [Último acceso: 12 Junio 2016].

[80] K. Singh, «WSN LEACH based protocols: A structural analysis,» Lovely Professional University, Jalandhar , 2015.

[81] C. Pham, «Congduc Pham's web page,» 12 Enero 2016. [En línea]. Available: http://cpham.perso.univ-pau.fr/#biography. [Último acceso: 2016 Junio 29].

[82] C. Pham, «Multimedia transmission on Wireless Sensor Networks: image and audio,» 24 Marzo 2015. [En línea]. Available: http://cpham.perso.univ-pau.fr/WSN-MODEL/tool-html/tools.html. [Último acceso: 29 Junio 2016].

[83] A. Rastogi, A. K. Shrivastava, N. Payal y R. Singh, «A proposed solution to Travelling Salesman Problem using Branch and Bound,» *International Journal of*

Computer Applications (0975 – 8887), vol. 65, nº 5, pp. 44-49, 2013.

[84] FieldComm Group, «WirelessHART - How it works,» 2014. [En línea]. Available: http://en.hartcomm.org/hcp/tech/wihart/wireless_how_it_works.html. [Último acceso: 17 Junio 2016].

[85] J. Gutierrez, «IEEE Std. 802.15.4 Enabling Pervasive Wireless Sensor Network,» 2005. [En línea]. Available: http://people.eecs.berkeley.edu/~prabal/teaching/cs294-11-f05/slides/day21.pdf. [Último acceso: 17 Junio 2016].

[86] K. Delaney, Ambient Intelligence with Microsystems, augmented materials and smart objects, Bishoptown: Springer, 2008, p. 178.

[87] F. Karkory y A. Abudalmola, «Implementation of Heuristics for Solving Travelling Salesman Problem Using Nearest Neighbour and Minimum Spanning Tree Algorithms,» *International Journal of Mathematical, Computational, Physical, Electrical and Computer Engineering,* vol. VII, nº 10, pp. 1524 - 1534, 2013.

[88] E. D. Cortés, «La Notación Big O,» 23 Noviembre 2013. [En línea]. Available: http://www.lnds.net/blog/2013/11/la-notacion-big-o.html. [Último acceso: 20 Junio 2016].

[89] M. Fowler y A. Maneiro, «Librería VS Framework,» No sólo software, [En línea]. Available: http://nosolosoftware.com/glosario/libreria-vs-framework/. [Último acceso: 2016 Junio 23].

[90] P. O., «Diferencia entre librería y framework,» 25 Septiembre 2014. [En línea]. Available: http://notasjs.blogspot.com/2014/09/diferencia-entre-libreria-y-framework.html. [Último acceso: 2016 Junio 23].

[91] Qt Company , 31 05 2016. [En línea]. Available: http://wiki.qt.io/About_Qt. [Último acceso: 23 Junio 2016].

[92] A.-R. Elshafei, «Introduction to Omnet++ and its Services,» [En línea]. Available: http://www.sce.carleton.ca/faculty/wainer/students/DEVSnet/_private/Omnet%20report.doc. [Último acceso: 23 Junio 2016].

[93] Texas Instruments Incorporated, 2016. [En línea]. Available: http://www.ti.com/product/CC2420. [Último acceso: 23 Junio 2016].

[94] A. Viklund, «MiXiM,» 2011. [En línea]. Available: http://mixim.sourceforge.net/. [Último acceso: 23 Junio 2016].

[95] «Resumen La utilización de una red corporal de sensores inalámbricos es una posible alternativa para el cuidado de la salud que está en pleno desarrollo.,» [En línea]. Available: http://med.10-multa.com/pravo/19311/index.html?page=9. [Último acceso: 05 abril 2016].

[96] C. P. Salinas y N. Histchfeld, «Tutorial de UML,» [En línea]. Available: http://users.dcc.uchile.cl/~psalinas/uml/introduccion.html. [Último acceso: 24 Junio 2016].

[97] W. R. Heinzelman, A. Chandrakasan y H. Balakrishnan, «Energy-Efficient Communication Protocol for Wireless Microsensor Networks,» de *Proceedings of the 33rd Hawaii International Conference on System Sciences - 2000*, Hawaii, 2000.

[98] ATMEL, «Vegard Wollan Talks AVR: How Atmel and AVR First Teamed Up,» youtube.com, 24 Agosto 2014. [En línea]. Available: https://www.youtube.com/watch?v=kY7rS39N0FA. [Último acceso: 14 Agosto 2016].

[99] A. B. García-Hernando, J. F. Martínez-Ortega, J. M. Lopéz Navarro, A. P. y L. Redondo López, Problem Solving for Wireless Sensor Networks, Londres: Springer, 2008, p. 173.

[100] B. Parikh, «RS232: Basics, Implementation & Specification,» Engineers Garage, 2012. [En línea]. Available: http://www.engineersgarage.com/articles/what-is-rs232. [Último acceso: 16 Agosto 2016].

[101] PDA cortex, «Intel XScale Technology,» 2001. [En línea]. Available: http://www.pdacortex.com/intel_xscale_technology.htm. [Último acceso: 16 Agosto 2016].

[102] B. Underdahl, iPAQ for Dummies, Indianapolis: Wiley, 2004.

[103] Informática Moderna, «LA MEMORIA RAM TIPO DIMM - SDRAM,» 2016. [En línea]. Available: http://www.informaticamoderna.com/Memoria_DIMM.htm. [Último acceso: |6 Agosto 2016].

[104] ARM, «ARM7 Processor Family,» 2016. [En línea]. Available: https://www.arm.com/products/processors/classic/arm7. [Último acceso: 16 Agosto 2016].

[105] TinyOS Wiki, «TinyOS Overview,» 12 Mayo 2013. [En línea]. Available: http://tinyos.stanford.edu/tinyos-wiki/index.php/TinyOS_Overview. [Último acceso: 16 Agosto 2016].

[106] ContikiOS, «Contiki: The Open Source OS for the Internet of Things,» [En línea]. Available: http://www.contiki-os.org/. [Último acceso: 17 Agosto 2016].

ANEXOS

Los anexos enumerados a continuación se encuentran en el CD adjunto al presente trabajo, han sido clasificados en directorios cuyo nombre es el mismo que el identificativo del anexo.

ANEXO 1: *Shell Bash Script* para el Subproceso de control de la simulación.

ANEXO 2: Archivo de configuración para el primer subproceso de simulación, ConnectivityMap.

ANEXO 3: Archivos auxiliares generados en la primera etapa de la simulación.

ANEXO 4: Código fuente para la resolución del TSP utilizando el método de aproximación por el MST y permutaciones mediante el algoritmo 2-opt (segunda etapa de la simulación).

ANEXO 5: Código fuente para la resolución del TSP utilizando el método de *Branch and Bound* y el límite inferior de Held-Karp (segunda etapa de la simulación).

ANEXO 6: Archivos auxiliares generados en la segunda etapa de la simulación

ANEXO 7: Archivo de configuración para el tercer subproceso de simulación: ThroughputTest.

ANEXO 8: Archivos auxiliares generados en la tercera etapa de la simulación.

ANEXO 9: Comprobación del correcto funcionamiento de los programas para resolver el TSP.