ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

GRADO EN INGENIERÍA TELEMÁTICA

EVALUACIÓN DE LA TECNOLOGÍA DE COMUNICACIONES INALÁMBRICAS BLE MEDIANTE ENTORNOS SIMULADOS Y EXPERIMENTALES

Autor: Álvaro Bartolomé Ucero

Director: David Contreras Bárcena

Madrid

Julio 2018

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Evaluación de la tecnología de comunicaciones inalámbricas BLE mediante entornos simulados y experimentales

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2017/18 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: Fecha: ……/ ……/ ……

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Fecha: ……/ ……/ ……

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.

El autor D. Álvaro Bartolomé Ucero DECLARA ser el titular de los derechos de propiedad intelectual de la obra: Evaluación de la tecnología de comunicaciones inalámbricas BLE mediante entornos simulados y experimentales, que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar "marcas de agua" o cualquier otro sistema de seguridad o de protección.
b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
e) Asignar por defecto a estos trabajos una licencia Creative Commons.
f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente).*

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

a) Que la Universidad identifique claramente su nombre como autor de la misma
b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
c) Solicitar la retirada de la obra del repositorio por causa justificada.
d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

El autor se compromete a:

a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e intereses a causa de la cesión.

d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción de derechos derivada de las obras objeto de la cesión.

6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 11 de Julio de 2018

ACEPTA

Fdo……

Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

GRADO EN INGENIERÍA TELEMÁTICA

EVALUACIÓN DE LA TECNOLOGÍA DE COMUNICACIONES INALÁMBRICAS BLE MEDIANTE ENTORNOS SIMULADOS Y EXPERIMENTALES

Autor: Álvaro Bartolomé Ucero

Director: David Contreras Bárcena

Madrid

Julio 2018

EVALUACIÓN DE LA TECNOLOGÍA DE COMUNICACIONES INALÁMBRICAS BLE MEDIANTE ENTORNOS SIMULADOS Y EXPERIMENTALES

Autor: Bartolomé Ucero, Álvaro.
Director: Contreras Bárcena, David.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

Palabras clave: BLE, Bluetooth Low Energy, Bluetooth Smart, NS-3, Omnet++, consumo, scanner, maestro, advertiser, esclavo, intervalo de publicidad, intervalo de escaneo.

1. Introducción

BLE es una tecnología digital inalámbirca de Bluetooth centrada en sensores o pequeños dispositivos. Al igual que el Bluetooth convencional, nos permite establecer una comunicación entre una gran variedad de dispositivos, acercándonos cada vez más a lo conocido como el Internet de las Cosas.

La aparición de esta tecnología surge de la necesidad de minimizar el consumo y el tamaño de los dispositivos participantes en una comunicación Bluetooth. Antes eran necesarias baterías muy grandes para cubrir el elevado consumo de estos dispositivos. Sin embargo, BLE asienta las bases en este ámbito gracias a su gran flexibilidad, a su baja energía y a su amplio soporte para “Smartphones” y “Tablets”.

2. Objetivos del proyecto

El objetivo principal de este proyecto es el de estudiar y analizar la tecnología de comunicaciones inalámbricas BLE mediante entornos simulados y experimentales. Para ello, lo primero es recopilar información acerca de esta tecnología y resumir alguno de los aspectos más importantes, como su funcionamiento y su arquitectura.

Tras terminar con la parte teórica y entender cómo funciona el Bluetooth de Baja Energía, podemos pasar a analizar la tecnología BLE mediante simuladores. En los seis meses de tiempo de vida del proyecto se utilizan dos simuladores de red. En un primer momento se comienza usando el NS-3, con el que trabajamos desde una máquina virtual de Linux. Sin embargo, debido a una serie de problemas con la implementación del módulo BLE en el propio simulador, decidimos utilizar otro.

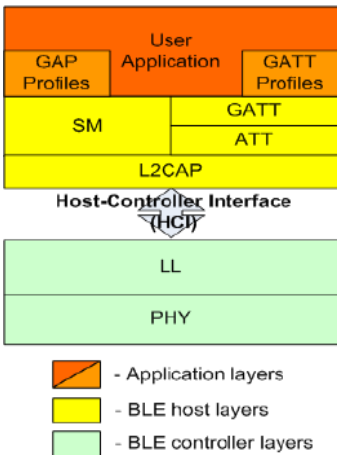
El segundo y definitivo simulador empleado para el estudio de esta tecnología Bluetooth fue el Omnet++, con el que trabajamos desde Windows. El primer objetivo importante para comenzar a trabajar con el simulador era el de encontrar un módulo BLE que se implemente sin problemas. Tras esto, podemos probar varios tipos de comunicaciones BLE simuladas y parametrizadas de una manera concreta para así obtener las conclusiones pertinentes en cuanto al consumo, tanto del dispositivo Maestro o Scanner, como del dispositivo Esclavo o Advertiser. Además, gracias al simulador somos capaces de ver cómo trancurre esta comunicación desde varios puntos de vista y con distintas configuraciones.

3. Arquitectura BLE

El Bluetooth de Baja Energía sigue una organización por capas que comentaremos más adelante, pero antes es importante entender cómo se comportan los dispositivos participantes en una comunicación BLE. Estos pueden actuar como dispositivos centrales (Scanner o Maestro) o como periféricos (Advertiser o Esclavo). Estos últimos suelen ser sensores de baja potencia que se conectan al dispositivo central transmitiéndole información.

Por otro lado, la información enviada por el Advertiser puede ser de dos tipos: paquetes de aviso y datos de respuesta de escaneo. Los primeros son necesarios para que una comunicación BLE funcione y son transmitidos por un dispositivo periférico siempre que sea "visto" por otro. Cuando otro dispositivo recibe los datos de este primer periférico, puede solicitar datos adicionales. Estos datos son los segundos antes mencionados, los datos de respuesta de escaneo, que contienen información acerca de ese periférico.

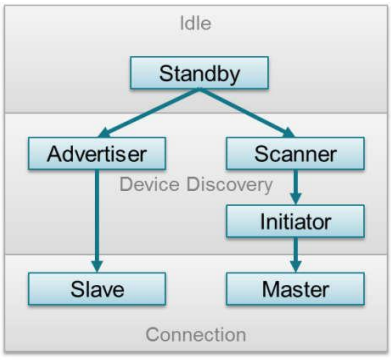
La tecnología BLE tiene una distribución por capas y subcapas que mostraremos en la siguiente imagen:



Se pueden distinguir tres grandes secciones con una serie de capas en cada una de ellas, excepto la capa de Aplicación, que no posee ninguna subcapa.

- Aplicación: Contiene la lógica, la interfaz del usuario y el manejo de los datos.
- Host: GAP, GATT, L2CAP, ATT, SM Y HCI (lado del host).
- Controlador: LL, PHY y HCI (lado del controlador).

En este pequeño resumen no me voy a parar a explicar cada una de las capas, exceptuando la que controla los perfiles genéricos de acceso (GAP), ya que tiene una especial importancia en este proyecto por encargarse de establecer el estado de los dispositivos participantes en la comunicación BLE: Advertiser o Scanner. En la siguiente imagen podremos ver el diagrama del estado de GAP:



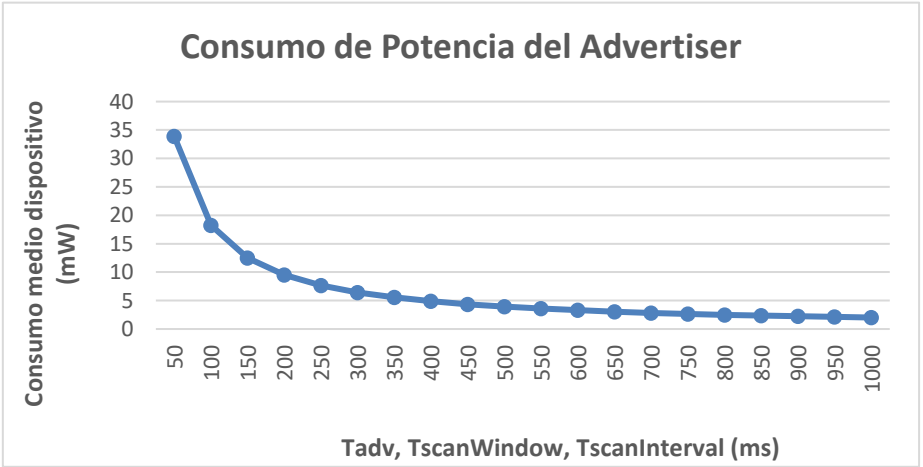
Es muy importante el papel o rol que desempeña cada uno de los dispositivos dentro de la comunicación, teniendo cada uno su propia funcionalidad:

- Standby: Está en modo inactivo.
- Advertiser: Está anunciándose al resto de dispositivos para que se conecten.
- Scanner: Está recibiendo anuncios y enviando respuestas de escaneo.
- Initiator: Está preparado para establecer una conexión con otro dispositivo.
- Slave: Dispositivo que estaba en modo Advertiser tras la conexión.
- Master: Dispositivo que estaba en modo Initiator tras la conexión.

4. Resultados

Los resultados obtenidos en estre proyecto tras todas las simulaciones parametrizadas realizadas son de cuatro tipos: simulación en tiempo real, archivos gráficos y textuales que resumen la comunicación BLE, cambios en la radio y el consumo de los dispositivos. Todos los resultados obtenidos fueron mediante el simulador Omnet ++ y en este pequeño resumen se va a mostrar un pequeño adelanto respecto al consumo de los dispositivos. Todos los demás resultados se pueden ver en la memoria completa, en los capítulos finales.

A la hora de parametrizar las simulaciones, hay una gran cantidad de parámetros que se pueden manipular para ver sus efectos en la comunicación entre los dispositivos BLE. En nuestro caso, se manipularán los intervalos de publicidad (Tadv) y de escaneo (TscanInterval), la ventana de escaneo (TscanWindow) será siempre igual al intervalo de escaneo para transmitir el 100 % del tiempo. Además, se va a cambiar el número de Scanners y Advertisers para contemplar diferentes tipos de comunicaciones.



En la figura podemos ver uno de los parámetros medidos, el consumo de potencia del Advertiser al cambiar los intervalos conjuntamente desde 50 a 1000 ms. Se puede contemplar que, al aumentar los intervalos, dicha potencia consumida comienza disminuyendo a gran velocidad hasta mantenerse casi constante en un valor ligeramente superior a los 0 mW. Esto es bastante relevante, ya que indica que cuanto más tiempo pasa desde que el dispositivo Esclavo se comunica con el Maestro hasta que lo vuelve a hacer, consume mucha menos potencia.

5. Conclusiones

Gracias al simulador Omnet++ se ha podido analizar cómo cambian los consumos dentro de una comunicación BLE, siendo esta tecnología conocida precisamente por eso, su bajo consumo. Se intentaron estudiar también los cambios en el tiempo de descubrimiento de los dispositivos, pero por problemas con el módulo, no se pudo realizar. En un futuro sería intersante mejorar dicho módulo BLE para analizar esos tiempos, entre otros muchos más parámetros relevantes, y comparar los resultados con los obtenidos en un entorno real.

6. Referencias

[1] "Introduction to Bluetooth Low Energy (BLE) v4.0". Argenox Technologies LLC. http://www.argenox.com/bluetooth-low-energy-ble-v4-0development/library/introduction-to-bluetooth-low-energy-v4-0/

[2] "Generic Access Profile (GAP)". Texas Instruments. http://dev.ti.com/tirex/content/simplelink_cc2640r2_sdk_1_40_00_45/docs/blestack/ble_user_guide/html/ble-stack-3.x/gap.html

[3] "BLE images", ResearchGate. https://www.researchgate.net/figure/BLE-stack-architecture_fig3_259333192?_sg=uM9lARDyzIMww6oRHjqVXfYz8xdv3sh3sR13X1-XvFqef8U_qM-5pv2gEEJh5aVJ57pJF2QUzlVMN-Mf3t_Ssg

[4] "Bluetooth Low Energy – Part 1: Introduction to BLE". MicroElektronika. https://www.mikroe.com/blog/bluetooth-low-energy-part-1-introduction-ble

[5] "Bluetooth Low Energy". Wikipedia The Free Encyclopedia. https://en.wikipedia.org/wiki/Bluetooth_Low_Energy

EVALUATION OF BLE WIRELESS COMMUNICATIONS TECHNOLOGY THROUGH SIMULATED AND EXPERIMENTAL ENVIRONMENTS

Author: Bartolomé Ucero, Álvaro.
Director: Contreras Bárcena, David.
Collaborative Entity: ICAI – Universidad Pontificia Comillas

ABSTRACT

Keywords: BLE, Bluetooth Low Energy, Bluetooth Smart, NS-3, Omnet++, consume, scanner, master, advertiser, slave, advertising interval, scan interval.

1. Introduction

BLE is a Bluetooth wireless digital technology focused on sensors or small devices. Like conventional Bluetooth, it allows us to establish a communication between a great variety of devices, getting closer and closer to what is known as the Internet of Things.

The appearance of this technology arises from the need to minimize the consumption and the size of the devices participating in a Bluetooth communication. Before, very large batteries were needed to cover the high consumption of these devices. However, BLE establishes the foundations in this field thanks to its great flexibility, its low power and its wide support for "Smartphones" and "Tablets".

2. Project objectives

The main objective of this project is to study and analyze BLE wireless communications technology through simulated and experimental environments. For this, the first thing is to gather information about this technology and summarize some of the most important aspects, such as its operation and architecture.

After finishing with the theoretical part and understanding how the Bluetooth of Low Energy works, we can move on to analyze the BLE technology through simulators. In the six months of project life time, two network simulators are used. At first you start using the NS-3, with which we work from a Linux virtual machine. However, due to a series of problems with the implementation of the BLE module in the simulator itself, we decided to use another one.

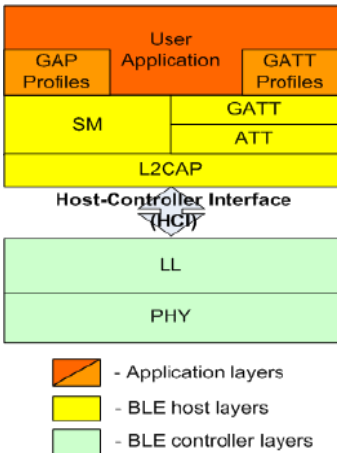
The second and final simulator used to study this Bluetooth technology was the Omnet ++, with which we work from Windows. The first important goal to start working with the simulator was to find a BLE module that is implemented without problems. After this, we can test several types of BLE communications simulated and parameterized in a specific way to obtain the relevant conclusions regarding the consumption, both Master or Scanner device, and Slave or Advertiser device. In addition, thanks to the simulator we can see how this communication runs from several points of view and with different configurations.

3. BLE architecture

Bluetooth Low Energy follows a layered organization that we will discuss later, but before it is important to understand how the participating devices behave in a BLE communication. These can act as central devices (Scanner or Master) or as peripherals (Advertiser or Slave). The latter are usually low-power sensors that connect to the central device transmitting information.

On the other hand, the information sent by the Advertiser can be of two types: warning packages and scan response data. The first ones are necessary for a BLE communication to work and are transmitted by a peripheral device whenever it is "seen" by another. When another device receives the data from this first peripheral, you can request additional data. These data are the second ones, the scan response data, which contain information about that peripheral.

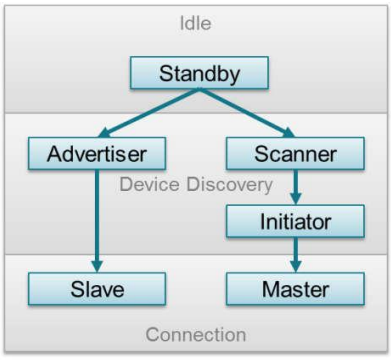
The BLE technology has a distribution by layers and sublayers that we will show in the following image:



You can distinguish three large sections with a series of layers in each of them, except the Application layer, which does not have any sublayer.

- Application: Contains the logic, the user interface and the data management.
- Host: GAP, GATT, L2CAP, ATT, SM Y HCI (host side).
- Controller: LL, PHY y HCI (controller side).

In this short summary I will not stop to explain each of the layers, except for the one that controls the generic access profiles (GAP), since it has a special importance in this project because it is in charge of establishing the state of the participating devices in BLE communication: Advertiser or Scanner. In the following image we can see the diagram of the state of GAP:



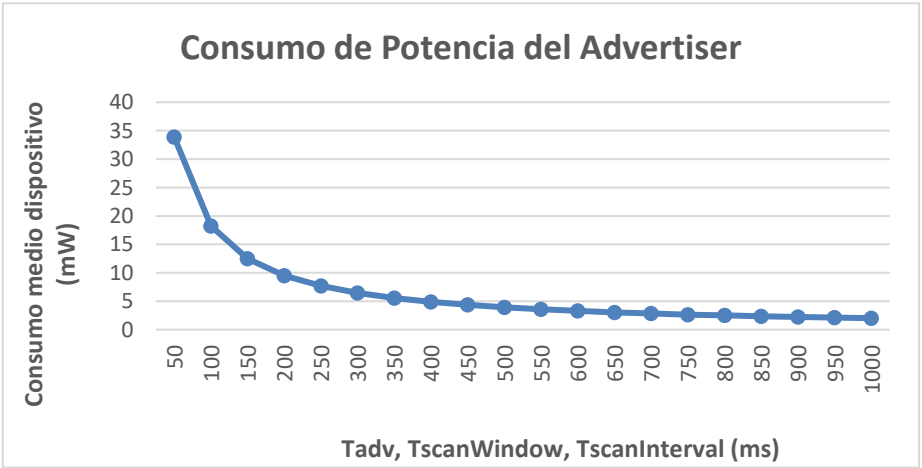
It is very important the role or role played by each of the devices within the communication, each having its own functionality:

- Standby: It is in idle mode.
- Advertiser: It is announcing to the other devices to connect.
- Scanner: It is receiving announcements and sending scan answers.
- Initiator: It is ready to establish a connection with another device.
- Slave: Device that was in Advertiser mode after connection.
- Master: Device that was in Initiator mode after connection.

4. Results

The results obtained in this project after all the parameterized simulations carried out are of four types: real-time simulation, graphic and textual files that summarize the BLE communication, changes in the radio and the consumption of the devices. All the results obtained were through the Omnet ++ simulator and in this short summary we will show a small advance regarding the consumption of the devices. All other results can be seen in the complete memory, in the final chapters.

When parameterizing simulations, there are a large number of parameters that can be manipulated to see their effects on communication between BLE devices. In our case, the advertising (Tadv) and scanning (TscanInterval) intervals will be manipulated, the scanning window (TscanWindow) will always be equal to the scan interval to transmit 100% of the time. In addition, the number of Scanners and Advertisers will be changed to contemplate different types of communications.



In the figure we can see one of the measured parameters, the power consumption of the Advertiser when changing the intervals jointly from 50 to 1000 ms. When increasing the intervals, said consumed power begins decreasing at a high speed until it remains almost constant at a value slightly higher than 0 mW. This is quite relevant, since it indicates that the longer it takes since the Slave device communicates with the Master until it does it again, it consumes much less power.

5. Conclusion

Thanks to the Omnet ++ simulator it has been possible to analyze how the consumption changes within a BLE communication, being this technology known precisely for that reason, its low consumption. We also tried to study the changes in the discovery time of the devices, but due to problems with the module, it could not be done. In the future it would be interesting to improve said BLE module to analyze those times, among many other relevant parameters, and compare the results with those obtained in a real environment.

6. References

[6] "Introduction to Bluetooth Low Energy (BLE) v4.0". Argenox Technologies LLC. http://www.argenox.com/bluetooth-low-energy-ble-v4-0development/library/introduction-to-bluetooth-low-energy-v4-0/

[7] "Generic Access Profile (GAP)". Texas Instruments. http://dev.ti.com/tirex/content/simplelink_cc2640r2_sdk_1_40_00_45/docs/blestack/ble_user_guide/html/ble-stack-3.x/gap.html

[8] "BLE images", ResearchGate. https://www.researchgate.net/figure/BLE-stack-architecture_fig3_259333192?_sg=uM9lARDyzIMww6oRHjqVXfYz8xdv3sh3sR13X1-XvFqef8U_qM-5pv2gEEJh5aVJ57pJF2QUzlVMN-Mf3t_Ssg

[9] "Bluetooth Low Energy – Part 1: Introduction to BLE". MicroElektronika. https://www.mikroe.com/blog/bluetooth-low-energy-part-1-introduction-ble

[10] "Bluetooth Low Energy". Wikipedia The Free Encyclopedia. https://en.wikipedia.org/wiki/Bluetooth_Low_Energy

Índice de la memoria

Índice de figuras

Índice de tablas

Capítulo 1. INTRODUCCIÓN

1.1. DEFINICIÓN DE LA TECNOLOGÍA BLE

BLE son las siglas de "Bluetooth low energy", Bluetooth de baja energía en español. También se conoce como Bluetooth LE, Bluetooth ULP y Bluetooth Smart. Es una tecnología digital inalámbrica de Bluetooth que se centra en pequeños dispositivos. Ofrece comunicación entre dispositivos móviles u ordenadores y otros más pequeños al igual que la tecnología Bluetooth convencional. Sin embargo, la tecnología BLE está diseñada para funcionar con mucha menos energía.

Las tecnologías de comunicación inalámbrica BLE permiten la comunicación entre dispositivos de pila de botón y dispositivos Bluetooth, opera en 2.4 GHz y presenta una tasa de transferencia de 1 Mbps en la capa física. Un sensor BLE está compuesto por un microchip de bajo costo, pero con grandes opciones para su empleo en la industria. Además, tiene el mismo tamaño que un sensor Bluetooth convencional, como los que se encuentran en un teléfono móvil. Estos sensores también tienen soporte para seguridad, ya que emplea un sistema de cifrado AES y esquemas de seguridad configurables.

Antes de BLE, el Bluetooth clásico necesitaba baterías más grandes debido al gran consumo de energía, además de un costoso chip de autenticación. Estos nuevos sensores resuelven estos problemas a la hora de conectarse con dispositivos móviles Apple o Android. Uno de los aspectos más poderosos de BLE es su flexibilidad y baja energía, que permite intercambiar información en forma genérica, a diferencia de la estructura rígida del Bluetooth clásico. Además, el amplio soporte de esta tecnología en Smartphones y tabletas hace que las aplicaciones BLE sean muy fáciles de implementar y de usar para los consumidores.

1.2. DIFERENCIAS CON EL BLUETOOTH CLÁSICO

En este apartado se contemplarán las principales diferencias entre la tecnología Bluetooth clásica y la mejora de esta tecnología respecto al consumo (BLE). Recordemos que BLE es una tecnología que comenzó a utilizar Bluetooth 4.0 para reducir su consumo y sus costes, empleándose en comunicaciones con pequeños dispositivos.

Especificaciones	***Bluetooth clásico***	***BLE***
Rango teórico	100 m	> 100 m
Velocidad aire	1-3Mbit/s	125-2000Kbit/s
Rendimiento	0.7-2.1Mbit/s	0.27Mbit/s
Esclavos	7	No definido
Seguridad	56-128-bit	128-bit AES
Latencia	100ms	6ms
Tiempo de envío	100ms	3ms
Soporte Voz	Sí	No
Topología	Scatternet	Scatternet
Potencia	1W	0.01-0.5W
Pico corriente	<30mA	<15mA
Concepto perfil	Sí	Sí

Tabla 1 – Diferencias entre el Bluetooth clásico y el BLE

En la tabla 1 es interesante observar la menor potencia de BLE con respecto al Bluetooth convencional. Podemos ver que BLE tiene menor rango, menor velocidad de transmisión en el aire y un menor rendimiento, por eso se suele usar para pequeños dispositivos que no requieran de tan elevadas prestaciones. Por otro lado, observamos que una comunicación Bluetooth hace uso de un dispositivo maestro y hasta 7 esclavos, mientras que en BLE puede

haber incluso más, no está definido. En cuanto a la seguridad no hay grandes diferencias, solo destacar que BLE presenta una algo menos resistente.

Ahora es cuando llegan las principales ventajas de BLE respecto al Bluetooth clásico. Observamos que la latencia, el tiempo que se tarda en transmitirse un paquete dentro de una red, es más de 10 veces menor en BLE, así como el tiempo de envío de los datos que es más de 20 veces menor.

No solo se reduce notablemente el tiempo de transmisión de los datos, sino que, como era de esperar, el consumo de potencia y de corriente también es mucho menor en BLE. Estas son las dos grandes ventajas de esta tecnología respecto a la clásica. Por otro lado, hay que destacar que usan la misma topología de red, Scatternet.

1.3. COMPARATIVA CON OTRAS TECNOLOGÍAS

Es interesante observar la diferencia entre BLE y otras tecnologías de comunicación inalámbrica, como pueden ser WiFi, Zigbee y otros mejores en ciertas aplicaciones concretas. La realidad es que BLE no tiene por qué ser siempre la mejor solución.

Especificaciones	***BLE***	***Wi-Fi***	***Zigbee***
Banda Frec.	2.4GHz	2.4-5GHz	2.4GHz
Modulación	GFSK	OFDM, DSS	DSSS
Rango	<100m	<300m	<100m
Topología	Scatternet	Star	Mesh
Velocidad	1Mbps	11-150Mbps	250kbps
Corriente pico	<15mA	60-200mA	19-35mA R/TX
Corr. en espera	<2uA	<100uA	5uA

Tabla 2 – Comparativa de BLE con otras tecnologías

En la tabla 2 vemos que los tres protocolos usan la misma banda ISM de 2.4GHz, sin embargo, tienen capacidades muy dispares. La tecnología Bluetooth de baja energía es la que menor alcance posee, especialmente comparándola con el WiFi. Por otro lado, es recomendable que el consumo de corriente pico sea reducido, y el BLE es el que más bajo lo tiene.

A diferencia de Zigbee, BLE no puede formar redes en malla, pero hay algunas empresas como CSR que están proporcionando implementaciones específicas para ello. El problema es que en una comunicación punto a punto, que es lo que utiliza BLE, los dispositivos tienen que estar en el rango del dispositivo transmisor para controlarlo. Otra posible opción sería la de utilizar una pasarela inalámbrica para conectar los dispositivos BLE a un router. El problema de esta medida es el incremento en el costo final del producto.

1.4. MERCADO

En primer lugar, destacar que el 18 de julio de 2017, la empresa Bluetooth incorporó una serie de sistemas de perfiles y modelos que permitieron utilizar las comunicaciones BLE en mucha más cantidad de dispositivos y aplicaciones, como en la automatización de las casas, sensores de redes, aplicaciones de IoT y otras muchas como, por ejemplo:

- Cuidados de la salud: Utilizados comúnmente para medir la presión sanguínea o la temperatura corporal.
- Fitness: Por ejemplo, para medir la cadencia y velocidad de las ruedas de una bicicleta.
- Conectividad a Internet: Mediante una serie de protocolos de internet.
- Sensores genéricos: Permitiendo conectar mediante BLE, ratones, teclados y otros dispositivos con una batería de gran durabilidad.
- Detección de proximidad: Por ejemplo, al recibir una llamada a tu teléfono móvil, si te encuentras en casa, gracias a BLE la llamada se podrá redirigir al teléfono fijo de tu propia casa.
- Perfiles de alertas y horarios: Permite al usuario recibir notificaciones y alertas de otro dispositivo.

Después de observar algunas de las aplicaciones más importantes de esta tecnología, es interesante ver las estadísticas del volumen de mercado de los dispositivos que hacen uso de BLE en los últimos años.

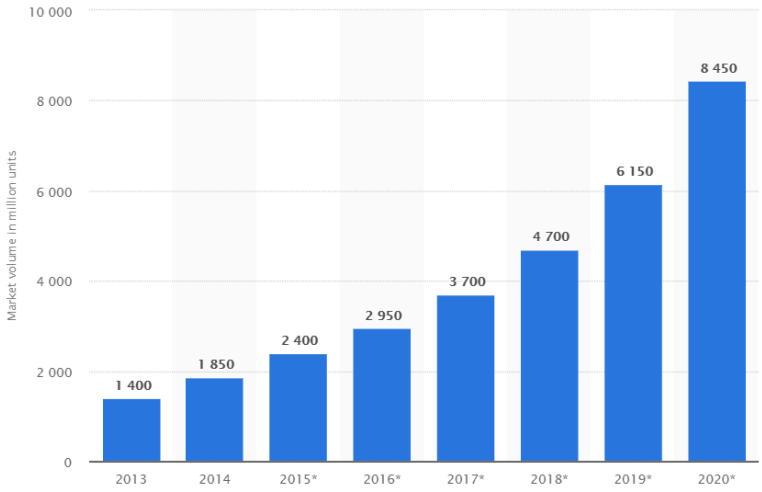


Figura 1 – Volumen de mercado de BLE en los últimos años

En la figura 1 podemos observar el masivo incremento, desde el 2013 hasta el 2020, de los dispositivos que utilizan Bluetooth LE. Mientras que en 2013 había 1.400 millones, en 2020 se prevé que pueda llegar a haber hasta 8.450 millones de dispositivos. Lo más seguro es que siga aumentando con el paso del tiempo.

1.5. APARICIÓN DE BLE

La tecnología BLE fue desarrollada en un primer momento por la compañía Nokia en 2001, además, fue acogida y apoyada por otras muchas empresas para asegurar una rápida disponibilidad de esta nueva tecnología. La interoperabilidad se realiza entre los principales fabricantes de semiconductores, los vendedores de dispositivos y los proveedores de servicio, como por ejemplo Nordic Semiconductor, Broadcom, Corporation, CSR y Epson. Nokia comenzó a desarrollar esta nueva tecnología siguiendo los estándares de Bluetooth, proporcionando una menor potencia y un menor coste. Los resultados fueron publicados en 2004 usando el nombre de "Bluetooth Low End Extension".

Después de varios años de desarrollo, esta nueva tecnología se lanzó al público en octubre de 2006 con la marca Wibree. Tras muchas negociaciones con Bluetooth, se llegó a un acuerdo en junio de 2007 para incluir Wibree en futuras especificaciones de Bluetooth. Esto sería conocido como "Bluetooth Low Energy Technology". Esta integración llegó en la versión 4.0 de Bluetooth en 2010 y el primer smartphone en acogerla fue el iPhone 4S, lanzado al público en octubre de 2011. Al año siguiente, aparecieron más dispositivos que implementaron esta tecnología.

La compañía Bluetooth desveló el 16 de junio de 2016, durante un evento en Londres, la tecnología conocida como Bluetooth 5, y no Bluetooth 5.0 o 5.0 LE como se esperaba. Esta nueva versión cuadruplicaría el rango de la anterior mediante el uso de una mayor potencia de transmisión y una capa física codificada, doblaría la velocidad gracias al uso de la mitad del tiempo de símbolo y proporcionaría un aumento de 8 veces la capacidad de transmisión de datos al aumentar la longitud de los datos. Todo esto sería de gran importancia para aplicaciones de IoT donde los nodos están conectados a través de toda una casa.

Finalmente, Bluetooth lanzó las especificaciones de Mesh Profile y Mesh Model oficialmente el 18 de julio de 2017. Esto permitiría usar el Bluetooth de Baja Energía para comunicaciones entre dispositivos, para domótica, para redes de sensores y otras muchas aplicaciones.

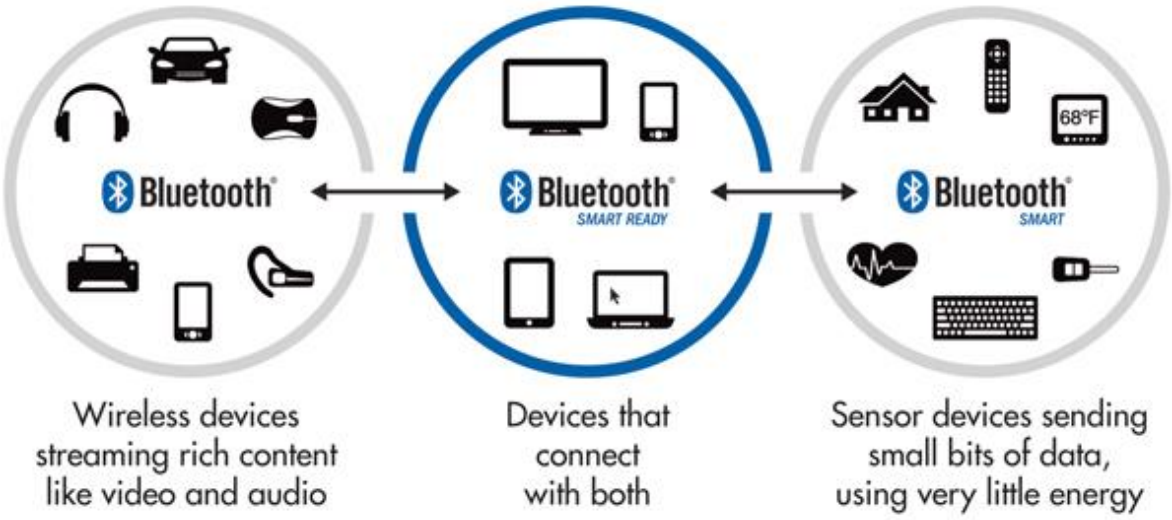


Figura 2 – Evolución del Bluetooth Smart o BLE con el paso del tiempo

En la figura 2 podemos ver como el Bluetooth clásico tan solo estaba presente en vehículos y en unos pocos dispositivos electrónicos. Además, se limitaba a proporcionar al usuario ciertas prestaciones como radio o vídeo. Con la llegada del Bluetooth Smart Ready, una gran parte de dispositivos electrónicos como móviles, ordenadores o tablets, se apropiaron de dicha tecnología para mejorar sus prestaciones y poder conectarse entre sí. Finalmente, con la llegada del Bluetooth Smart, prácticamente todo está conectado por Bluetooth: las casas, las llaves, sensores para salud y otras aplicaciones e incluso electrodomésticos comunes. En este momento es cuando los dispositivos envían pequeñas cantidades de bits de datos usando muy poca energía para transmitir información.

1.6. MOTIVACIÓN

En primer lugar, el estudio de una nueva tecnología de comunicaciones siempre es recomendable y es algo que nunca debe dejar de hacer un buen ingeniero. En segundo lugar, esta tecnología está en auge, por lo que es el momento propicio para estudiarla más a fondo llegando a mejorar enormemente en un futuro cercano. En tercer lugar, al tener diversas aplicaciones en IoT, gana más peso dentro del mundo de la ingeniería. En cuarto lugar, el trabajar con distintos simuladores da una visión más global de cómo funcionan las comunicaciones en entornos simulados y experimentales.

Capítulo 2. DEFINICIÓN DEL TRABAJO

2.1 MOTIVACIÓN

El objetivo principal de este proyecto es el de estudiar y analizar la tecnología de comunicaciones inalámbricas BLE mediante entornos simulados y experimentales. Para ello, lo primero será recopilar información acerca de esta tecnología y resumir algunos de los aspectos más importantes, como su funcionamiento, su arquitectura o su aparición.

Tras entender cómo funciona el Bluetooth de Baja energía, es hora de pasar al análisis mediante simuladores. El primero de ellos será el simulador de red NS3, donde la versión utilizada será la 24. Más adelante veremos más detalles de su simulación y los problemas obtenidos con el módulo BLE adquirido para dicho simulador.

Al haber tenido problemas para analizar resultados del módulo BLE con el simulador NS3, el siguiente paso será probar con el Omnet ++. Este también es un simulador de red y con él, a diferencia de con el anterior, trabajaremos en Windows, en lugar de en una máquina virtual de Ubuntu. Más adelante veremos los resultados obtenidos con este simulador ya que el módulo BLE se implementó a la perfección.

Finalmente, tras haber recopilado información sobre la tecnología de comunicaciones inalámbricas BLE, haber probado a analizarla con el simulador NS3 (y ver que no ha dado los resultados deseados) y por fin haberla analizado correctamente con el Omnet ++, veremos y explicaremos una serie de resultados gráficos y numéricos de este módulo BLE. Para lograrlo, emplearemos el esqueleto de modelado MiXiM, así como una herramienta conocida como Git Bash para analizar los datos numéricos obtenidos de la traza resultante de la simulación. Todos estos resultados los veremos en los apartados finales del presente documento.

2.2 METODOLOGÍA DEL TRABAJO

Este proyecto comenzó aproximadamente a inicios de enero del 2018, en donde el primer mes y medio fue dedicado, entre otras cosas, a recopilar información sobre la tecnología de comunicaciones inalámbricas BLE. A finales de ese primer período se entregó el Anexo B, que constaba de los mismos apartados que la introducción de este documento. Cabe destacar que, tras los problemas encontrados en la simulación del módulo BLE con el NS3, tuvimos que decidirnos por otro simulador y espera que funcionase correctamente, por lo que dicho anexo difiere levemente de los explicado en este documento final.

Tras entender el funcionamiento de la tecnología Bluetooth de Baja Energía, a mediados de febrero de 2018 comenzamos a instalar y aprender a utilizar el simulador de red NS3. Todo este proceso fue realizado en una máquina virtual de Oracle dentro de una máquina Windows, para poder trabajar en Ubuntu, donde era más eficiente el simulador. En un inicio comenzamos utilizando la versión 27, que era la más nueva, pero finalmente nos decantamos por la 24, ya que el módulo BLE que encontramos utilizaba dicha versión. Más adelante mostraré algunos resultados y códigos obtenidos tras esta etapa.

Sin embargo, como ya he dicho, los resultados obtenidos no fueron los esperados, por lo que, a principios de mayo de 2018 tuvimos que buscar una alternativa. Finalmente nos decantamos por intentar analizar la tecnología BLE con el simulador Omnet ++. Algunas ventajas de utilizar este simulador frente al NS3 eran que podríamos trabajar sin problemas en Windows y que tenía grandes similitudes con Eclipse (programa ya utilizado en asignaturas de programación pasadas en la universidad). En primera instancia y, tras instalar el simulador junto a la herramienta de modelado MiXiM, y tras implementar el módulo BLE encontrado en internet, parecía que no lograríamos obtener resultados sustanciales. Sin embargo, finalmente lo logramos y esto será lo que analizaremos en los apartados finales de resultados y conclusiones de este documento.

Finalmente, a finales de mayo de 2018 ya teníamos el simulador Omnet ++ funcionando correctamente y con el módulo BLE implementado en la herramienta de modelado MiXiM.

Desde ese momento, comenzamos a obtener y analizar los resultados obtenidos, para ello también hicimos uso de Git Bash, para analizar con mayor facilidad la traza obtenida de las simulaciones. Este documento será entregado a mediados de julio de 2018, habiendo transcurrido aproximadamente cinco meses desde su inicio en enero de 2018.

2.3 RECURSOS QUE SE EMPLEARÁN

Todos los recursos necesarios para la consecución de este proyecto serán de software, ya que la brevedad y el poco tiempo de vida del proyecto han impedido que se use algo de hardware. Todos los programas empleados son gratuitos, por lo que no habrá coste alguno. Los empleados para el análisis de la tecnología BLE mediante el primer simulador son la máquina virtual de Oracle, para poder trabajar en Ubuntu desde una máquina Windows; el simulador de red NS-3, para poder simular entornos de comunicaciones BLE y otros muchos programas; y el módulo BLE para el simulador, que será necesario para la ejecución de programas que utilicen su amplia librería.

Los programas empleados para el estudio de esta tecnología mediante el segundo simulador utilizado son el propio simulador, el Omnet ++, que también nos permite simular entornos de comunicaciones BLE; el esqueleto de modelado MiXiM, que es el más utilizado en el Omnet ++ para modelar redes inalámbricas; el módulo BLE necesario para poder analizar programas y redes que empleen su librería; y Git Bash, para estudiar de una manera más sencilla y ordenada los resultados numéricos obtenidos de la traza de la simulación.

Capítulo 3. SIMULADORES

3.1 NS-3

En este apartado veremos algunas de las características del primer simulador utilizado, el NS3, así como sus ventajas e inconvenientes. Finalmente veremos detenidamente las conclusiones obtenidas y por qué cambiamos de simulador para el estudio de la tecnología BLE.

3.1.1 DEFINICIÓN

Antes de hablar de las características del NS3, hay que mencionar alguna de su versión anterior, NS2:

- Entre el año 2000 y el 2004, fue el simulador más utilizado para la investigación de redes. Más del 50% se las simulaciones de redes realizadas por ACM y IEEE usaron dicho simulador.
- Estuvo sin mantenimiento un largo período de tiempo.
- Tenía un anticuado diseño de código debido a que no tenía en cuenta ciertas bases de la programación moderna.
- No era tan escalable como algunos simuladores de la época.
- Los sistemas de traza eran algo difíciles de utilizar.

Tras conocer algunos datos de su versión previa, podemos comenzar a definir y caracterizar el NS3. Este es un simulador de redes basado en eventos discretos que se suele usar en entornos educativo y de investigación. Tiene la opción de simular protocolos unicast y multicast y se utiliza mucho en la investigación de redes móviles ad-hoc. Además, da la posibilidad de implementar un gran número de protocolos de redes cableadas e inalámbricas. Por otro lado, esta versión puede soportar todo el flujo de trabajo de la simulación desde la configuración hasta el análisis de las tramas. Además, es un software libre y se ofrece bajo la versión 2 de la GNU General Public License. Algunas de sus características son:

- Es un nuevo simulador como tal, realmente no es una evolución del NS2
- Los lenguajes de programación que utiliza son C++ y Python, a diferencia del NS2 que solo usaba C++.
- El proyecto del simulador comenzó en 2006 y actualmente sigue en desarrollo.
- Sus fundadores oficiales fueron la universidad de Washington, INRIA Sophia Antipolis y la universidad de tecnología de Georgia.

Por otro lado, es interesante observar la gran variedad de modelos de capa de enlace diferentes que tiene:

- Point-to-point (Enlaces PPP)
- Csma (Enlaces Ethernet)
- Bridge (Enlaces 802.1D)
- Wifi (Enlaces 802.11)
- Mesh (Enlaces 802.11s)
- Wimax (Enlaces 802.16)
- Tap-bridge, emu

Por último, cabe destacar que, gracias a su forma de generar el tráfico en la red, tiene una gran cantidad de aplicaciones, como, por ejemplo:

- Onoff: genera corrientes alternando entre períodos de encendido-apagado, además tiene unas posibilidades de parametrización muy elevadas.
- Packet sink: recibe paquetes o conexiones TCP.
- Ping6, v4ping: envía solicitudes ICMP ECHO.
- Udp-client/server: envía paquetes UDP con números de secuencia.
- Udp-echo: envía paquetes UDP sin números de secuencia.
- Radvd: anuncios del router con IPv6.

3.1.2 VENTAJAS

El simulador de red NS3 tiene una serie de características muy innovativas y útiles. En primer lugar, se encuentra la escalabilidad. Los paquetes pueden tener cero bytes virtuales, lo que hace que no ocupen memoria y se reduzca el tiempo de la simulación. Además, en cuanto a los nodos, en IPv4 no se gasta memoria en los nodos que no lo necesiten, no es necesario utilizar el modelo de movilidad y se pueden añadir fácilmente mejoras en un futuro. Resumiendo, el simulador puede aumentar o disminuir la capacidad y el tamaño de un sistema, en función de los requisitos, sin comprometer el funcionamiento y la cantidad de este.

En segundo lugar, otra característica importante de este simulador es la flexibilidad. Tiene una gran cantidad de protocolos incorporados y una buena cantidad de módulos, lo que hace que posea una extensa librería con la que poder realizar todo tipo de programas relacionados con diversos tipos de comunicaciones.

En tercer lugar, tiene un claro y sencillo diseño, lo que hace que el uso del simulador sea muy sencillo e intuitivo. Basta con conocer los comandos necesarios para la ejecución de programas y otros igual de básicos. Además, el simulador solo dispone de capacidad para programar en c++ o en Python, que son lenguajes de programación muy sencillos que no requieren de un gran nivel para la realización de programas básicos.

En cuarto lugar, los módulos integrados están directamente relacionados con entornos de comunicaciones inalámbricas y alámbricas reales, por lo que es muy recomendable el uso de este antes de llevar a cabo acciones que requieran algo de hardware. En definitiva, es un simulador cuyo objetivo principal es el de llevar a cabo diversas pruebas previas al montaje y configuración de una comunicación real.

Por otro lado, también tiene algunas ventajas relevantes en cuanto a prestaciones. Por un lado, es uno de los simuladores más rápidos del mundo, ya que posee una velocidad de transmisión muy elevada con respecto al resto de simuladores de red. Por otro lado, es el simulador que más eficientemente redistribuye la memoria.

3.1.3 INCONVENIENTES

El principal inconveniente que te encuentras nada más instalar el NS3, es el que no tenga interfaz gráfica de usuario. La mayoría de los simuladores, como por ejemplo el Omnet ++, tienen dicha interfaz y resulta muy cómodo trabajar en ella. La realización de programas se hace de una forma muy intuitiva y ordenada, además no hay que saberse de memoria una serie de comandos, como en el caso del NS3, para poder realizar, compilar y ejecutar los programas pertinentes. Bastaría con saber utilizar los elementos básicos de dicha interfaz y conocer el lenguaje de programación que estas utilizando para la realización de tus programas.

Sin embargo, el hecho de que no posea una GUI definida es tan solo un inconveniente que afecta a la comodidad del usuario, pero no afecta a las posibilidades de programación dentro del simulador. Por otro lado, otra de las grandes desventajas que presenta es que aún no hay implementados dentro del simulador (ni si quiera en la última versión, la 3.28) demasiados módulos para trabajar. Aquí nos encontramos con el primer problema serio que tuvimos en la realización de este proyecto.

El simulador NS3 traía incorporados una serie de módulos como por ejemplo el de antenas, aodv, core, energía, lte, control del tráfico, wifi, … Además de otros muchos que se podían obtener desde su propia página web. Sin embargo, el módulo BLE que nosotros necesitábamos había sido eliminado de la página web a mediados de 2017 por dar problemas y, a día de hoy, no han sido capaces de arreglarlo para poder ser utilizado por los usuarios. Sin embargo, gracias a la página GitHub, logramos encontrar un módulo BLE con el que empezamos a trabajar en nuestro simulador, pero más adelante veremos que no era lo esperado.

Finalmente, una de las características más atractivas del simulador era el poder visualizar gráficamente los resultados de los programas ejecutados. El problema viene cuando ese módulo de visualización aún está en fase de experimentación, por lo que daba algunos problemas. En mi caso, realicé una serie de programas sencillos de comunicaciones punto a punto para comprobar su funcionamiento, y funcionaba a la perfección. Sin embargo, hay

algunos módulos que no son compatibles con dicho módulo de visualización, como es el caso del módulo BLE. Tras muchas pruebas, no logramos implementar ambos módulos en el simulador sin que se produjese un error. Lo que estaba claro es que el fallo veía del módulo BLE, ya que no era el oficial de la página de NSNAM. Por estos dos últimos inconvenientes, y por la falta de resultados concluyentes tras las pruebas con el módulo BLE, decidimos finalmente optar por otra alternativa alejada del NS3.

3.1.4 CONCLUSIONES

En primer lugar, mostraré el funcionamiento y los resultados que muestra el simulador empleando una simple comunicación punto a punto junto a una csma, ambos son módulos que están implementados en él. Esto será útil para ver como se ejecuta un programa desde el terminal del NS3 y para ver el funcionamiento del módulo de visualización de Python incorporado en el simulador, ya que los módulos point-to-point y csma sí son compatibles con el de visualización.

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 */

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/csma-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/ipv4-global-routing-helper.h"

// Default Network Topology
//
//       10.1.1.0
// n0 -------------- n1   n2   n3   n4
//    point-to-point  |    |    |    |
//                    ================
//                      LAN 10.1.2.0


using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("SecondScriptExample");

int
main (int argc, char *argv[])
{
```

Figura 3 - Primera parte del código de una comunicación punto a punto con csma

```
main (int argc, char *argv[])
{
  bool verbose = true;
  uint32_t nCsma = 3;

  CommandLine cmd;
  cmd.AddValue ("nCsma", "Number of \"extra\" CSMA nodes/devices", nCsma);
  cmd.AddValue ("verbose", "Tell echo applications to log if true", verbose);

  cmd.Parse (argc,argv);

  if (verbose)
    {
      LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
      LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
    }

  nCsma = nCsma == 0 ? 1 : nCsma;

  NodeContainer p2pNodes;
  p2pNodes.Create (2);

  NodeContainer csmaNodes;
  csmaNodes.Add (p2pNodes.Get (1));
  csmaNodes.Create (nCsma);

  PointToPointHelper pointToPoint;
  pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
  pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

  NetDeviceContainer p2pDevices;
  p2pDevices = pointToPoint.Install (p2pNodes);

  CsmaHelper csma;
  csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
  csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));

  NetDeviceContainer csmaDevices;
  csmaDevices = csma.Install (csmaNodes);
```

Figura 4 - Segunda parte del código de una comunicación punto a punto con csma

```
  NetDeviceContainer csmaDevices;
  csmaDevices = csma.Install (csmaNodes);

  InternetStackHelper stack;
  stack.Install (p2pNodes.Get (0));
  stack.Install (csmaNodes);

  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");
  Ipv4InterfaceContainer p2pInterfaces;
  p2pInterfaces = address.Assign (p2pDevices);

  address.SetBase ("10.1.2.0", "255.255.255.0");
  Ipv4InterfaceContainer csmaInterfaces;
  csmaInterfaces = address.Assign (csmaDevices);

  UdpEchoServerHelper echoServer (9);

  ApplicationContainer serverApps = echoServer.Install (csmaNodes.Get (nCsma));
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 9);
  echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

  ApplicationContainer clientApps = echoClient.Install (p2pNodes.Get (0));
  clientApps.Start (Seconds (2.0));
  clientApps.Stop (Seconds (10.0));

  Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

  pointToPoint.EnablePcapAll ("second");
  csma.EnablePcap ("second", csmaDevices.Get (0), true);

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

Figura 5 - Tercera parte del código de una comunicación punto a punto con csma

En las figuras 3, 4 y 5 se muestra en c, el código del programa de una simple comunicación punto a punto con csma. En esta comunicación intervienen cinco nodos, de entre los cuales, cuatro de ellos están estableciendo una comunicación csma dentro de una misma LAN (10.1.2.0) y el quinto está estableciendo una comunicación punto a punto con uno de los otros cuatro nodos (10.1.1.0). En el programa, se comienza eligiendo el tipo de comunicación que habrá (en nuestro caso hay dos, punto a punto y csma). Después, se eligen los nodos que van a participar en dicha comunicación y se añaden datos como el retardo y la velocidad de transmisión, esto se realiza para ambas comunicaciones. A continuación, se instalan los nodos en cada una de las comunicaciones a las que pertenezcan y se asignan las direcciones IP que he mencionado antes a cada una de ellas. Más adelante se eligen los tiempos de inicio y final de las dos comunicaciones y se asignan más datos como el máximo número de paquetes a transmitir, el tamaño de estos, y el intervalo de envío de paquetes. Finalmente, se detiene la simulación para poder analizar los resultados desde el terminal del simulador.

```
alvarobartolo7@alvarobartolo7-VirtualBox:~/Escritorio/NS3/ns3/ns-allinone-3.27/n
s-3.27$ ./waf --run Ejemplo1
Waf: Entering directory `/home/alvarobartolo7/Escritorio/NS3/ns3/ns-allinone-3.2
7/ns-3.27/build'
Waf: Leaving directory `/home/alvarobartolo7/Escritorio/NS3/ns3/ns-allinone-3.27
/ns-3.27/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (1.017s)
At time 2s client sent 1024 bytes to 10.1.2.4 port 9
At time 2.0078s server received 1024 bytes from 10.1.1.1 port 49153
At time 2.0078s server sent 1024 bytes to 10.1.1.1 port 49153
At time 2.01761s client received 1024 bytes from 10.1.2.4 port 9
alvarobartolo7@alvarobartolo7-VirtualBox:~/Escritorio/NS3/ns3/ns-allinone-3.27/n
s-3.27$
```

Figura 6 - Resultado mostrado desde el terminal tras la simulación del programa

En la figura 6, se muestra el resultado de la simulación del programa anterior. Podemos ver como la comunicación comienza en el segundo 2, como indicamos en el código, y es donde el cliente envía 1024 bytes (tamaño de paquete) al puerto 9 del servidor, cuya dirección IP es 10.1.2.4. En el segundo 2.0078, el servidor recibe ese paquete del puerto 49153 del cliente, cuya dirección IP es 10.1.1.1. En ese mismo instante el servidor reenvía la respuesta del mismo tamaño al cliente y en el segundo 2.0161, el cliente recibe dicha respuesta. Esto se repetirá en intervalos de 1 segundo, hasta llegar al segundo 10, que es donde indicamos que finalizaría la comunicación.

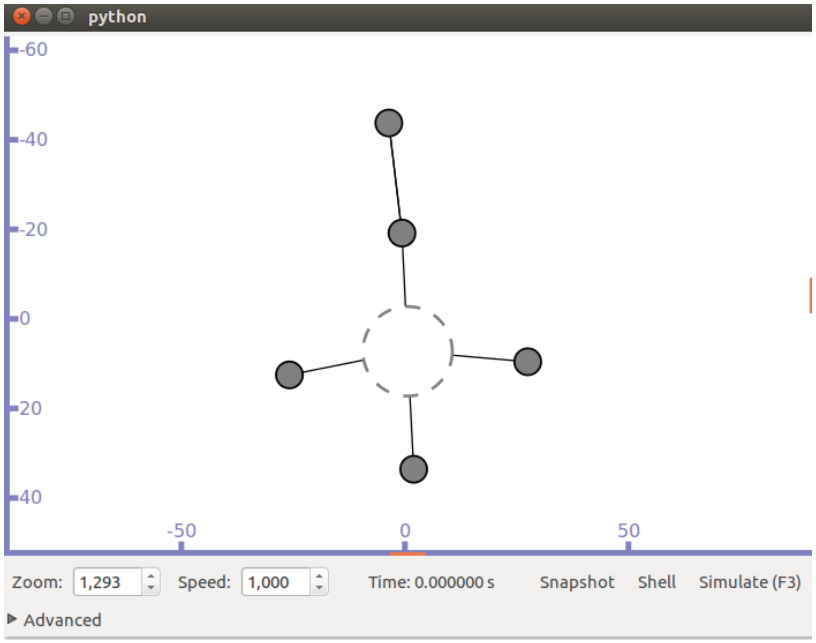


Figura 7 - Visualización del programa antes de comenzar la comunicación

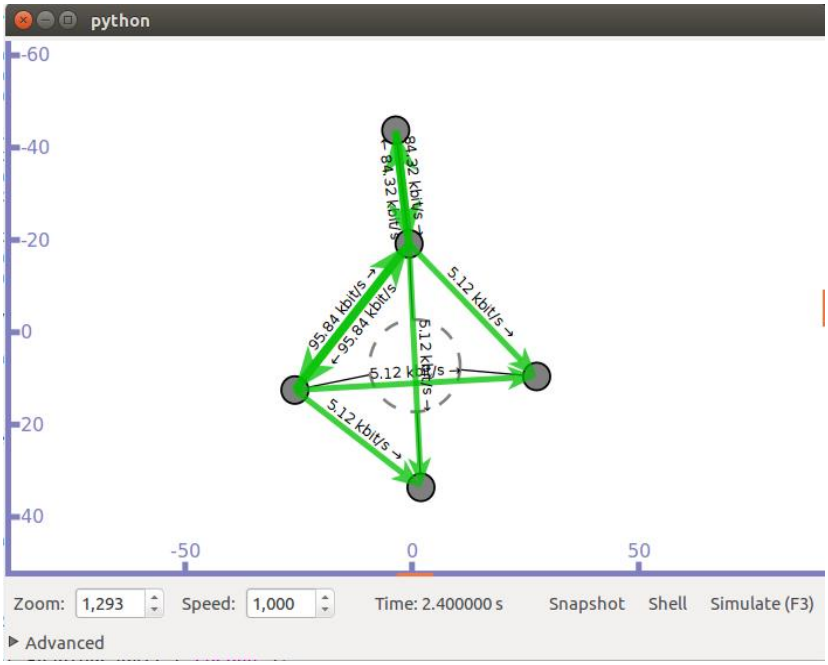


Figura 8 - Visualización del programa al iniciar la comunicación

En las figuras 7 y 8 vemos visualmente, gracias al módulo de visualización de Python incorporado, la simulación del programa antes comentado. Podemos ver como la comunicación empieza en el segundo 2 (la imagen está detenida en a los 2.4 segundos), y en ese momento los nodos empiezan a transmitir. Vemos también como las velocidades de transmisión cambian dependiendo del tipo de comunicación en la que se encuentren los nodos, pudiendo ser o bien csma (los cuatro de abajo), o bien punto a punto (los dos de arriba). Por este motivo el nodo que participa en ambas comunicaciones presenta velocidades de transmisión diferentes en función del nodo con el que se esté comunicando.

Después de ver en profundidad cómo funciona el simulado con un simple ejemplo de una comunicación que no es BLE, podemos ver que sucede en una comunicación BLE. El módulo conseguido de GitHub, traía consigo una serie de ejemplos, de entre los cuales, tan solo uno “funcionaba”. El código de este ejemplo en concreto, junto con los códigos pertinentes participantes en este simulador se mostrarán al final de este documento en el Anexo.

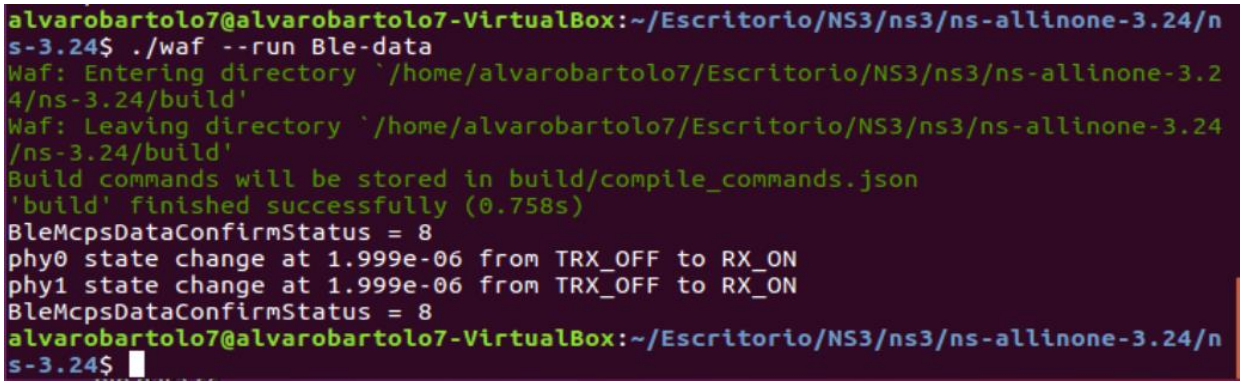


Figura 9 - Visualización del programa al iniciar la comunicación

En la figura 9 vemos el resultado de este programa que tan solo nos muestra algunos cambios de estado que sufren ciertos elementos de la comunicación BLE configurada en el programa. Como ya he dicho antes, el resto de los ejemplos no funcionan y muestran una serie de errores por pantalla. Además, al no poder emplear el módulo de visualización de Python por incompatibilidad, los resultados mostrados son casi nulos. Lo más probable es que hubiese errores graves en el módulo BLE, por esa razón la propia página web del simulador eliminó dicho módulo indefinidamente. En un futuro quizás vuelva a estar operativo sin fallos, pero hasta que ese día llegue, nosotros continuaremos nuestro análisis de la tecnología de comunicaciones inalámbricas BLE mediante el simulador de eventos discretos Omnet ++.

3.2 OMNET ++

En este segundo apartado veremos las características, ventajas e inconvenientes del simulador Omnet ++. En el apartado de conclusiones, no nos extenderemos demasiado ya que será en los apartados finales del documento donde extraeremos las conclusiones tras analizar el módulo BLE con este simulador.

3.2.1 DEFINICIÓN

Omnet ++ es una herramienta de modelado y de simulación pública que está basada en componentes modulares y posee un ambiente de simulación de arquitectura abierta con una fuerte interfaz gráfica de usuario. Este es un gran punto a favor para este simulador, ya que, como ya dije anteriormente, el NS3 no poseía una GUI. Su principal aplicación es la de simular redes de comunicación y, gracias a su arquitectura genérica y flexible, ha sido muy utilizado para la ejecución de redes basadas en colas de espera y arquitectura de hardware.

Como ya se ha mencionado con anterioridad, este simulador posee una arquitectura modular donde sus componentes son módulos programados en C ++. Mas tarde son ensamblados en componentes más grandes mediante lenguaje de alto nivel. Omnet ++ no es un simulador de red como tal, sin embargo, está ganando mucha popularidad como plataforma de simulación, ya que ejecuta la misma función. Además, tiene la posibilidad de correr en Linux, Unix-like y Win32. A continuación, mencionaremos una serie de características de este simulador:

- Este simulador es una versión libre con fines académicos, sin embargo, también tiene una versión comercial desarrollada por Omnets Global Inc, llamada OMNEST.
- Está orientado a simular objetos y eventos discretos en redes de comunicaciones.
- Como ya he mencionado antes, tiene una gran cantidad de herramientas y puede ser utilizado en plataformas UNIX y Windows.
- El lenguaje utilizado por este simulador es NED, que está basado en C++, para crear topologías de red. Este lenguaje en alto nivel ayuda a reducir la complejidad, ya que todos los modelos estarán creados por módulos que pueden contener estructuras complejas de datos con sus propios parámetros.

- Gracias a este lenguaje NED, podemos modelar el tráfico de información sobre redes y protocolos de red, además de poder definir tres tipos diferentes de módulos: simples, compuestos y de redes.
- Omnet++ hace uso de una potente interfaz gráfica de usuario que cuenta con una herramienta gráfica para editar el código NET, llamada GNED. Con dicha herramienta podemos generar el propio código para así configurar y simular las redes pertinentes.
- El simulador no solo cuenta con una GUI, sino que posee varias diferentes. De entre ellas, la más avanzada permite al usuario visualizar el modelo, controlar su ejecución y cambiar los propios objetos de dicho modelo.
- El usuario tiene la posibilidad de depurar de forma sencilla las configuraciones aplicadas a las redes que ha creado, gracias a su simple y acertada interfaz gráfica de usuario.

Por otro lado, será interesante mencionar alguno de los componentes más destacables del simulador. En primer lugar, se encuentran las librerías de simulación inteligente:

- Nede: Compilador para la topología de descripción de lenguaje NED
- Tkenv: GUI para la ejecución del simulador con enlaces dentro de las simulaciones ejecutables.
- Cmdenv: Interfaz del usuario para ver las líneas de comandos de la simulación ejecutada.
- Plove: Herramienta gráfica que utiliza vectores para mostrar las gráficas de los resultados.
- Scalars: Herramienta de visualización escalar para mostrar las gráficas de los resultados.
- Opp_neddoc: Herramienta de documentación de modelos.
- Utilidades: Herramienta de generación de números aleatorios usada para la creación de archivos.
- Documentación, simulaciones de muestra, etc.

3.2.2 VENTAJAS

En primer lugar, la ventaja más evidente frente al NS3 y a otros muchos simuladores, es su excelente interfaz gráfica de usuario o GUI. Además de ser muy intuitiva y sencilla de utilizar, es muy completa y avanzada. Esta es una de las principales razones por las que el Omnet++, pese a no ser un simulador como tal, es uno de los programas más utilizados para tareas de simulación. Como opinión personal, noté mucho la diferencia entre trabajar con el NS3 y con este simulador, es de agradecer que posea una GUI tan bien organizada.

En segundo lugar, otro gran punto a favor para este simulador frente al NS3, es su comodidad para trabajar en diferentes tipos de sistemas. Mientras que simulador anterior ofrecía mejores prestaciones para Linux y se quedaba algo corto para Windows, este ofrece las mismas ventajas tanto para sistemas UNIX-like (Linux, MacOS X) como para Windows. Esto se agradece bastante ya que no obliga a instalarse una máquina virtual en una máquina Windows para poder trabajar en Linux, que fue lo que tuve que hacer con el NS3.

En tercer lugar, posee integración con Eclipse, que es una de las plataformas de software más utilizadas hoy en día entre programadores e ingenieros. Esto es de gran ayuda, no solo porque la interfaz de usuario es muy parecida en ambos programas, sino también porque se pueden importar proyectos entre ambos, siempre y cuando estén programados en C++, que es el lenguaje de programación que utiliza Omnet++.

En cuarto, y último lugar, este simulador ofrece la posibilidad de realizar tus propios programas a base de código en c, pero también pueden generarse programas en base a una red construida con los elementos que ofrece el simulador. Esto ofrece dos caminos diferentes para la realización de programas, pero igual de importantes, dando la posibilidad de elegir al usuario entre programar una simple comunicación o simplemente colocar los elementos necesarios para su correcto funcionamiento.

3.2.3 INCONVENIENTES

Uno de los mayores problemas del simulador es que no ofrece la posibilidad de dar marcha atrás al realizar cambios en los proyectos. En el caso de darse cuenta de que se ha arrastrado un error desde el inicio del proyecto, haría falta reescribir y adaptar casi todos los ficheros participantes en dicho proyecto. Esto es un enorme problema, ya que no es tan raro darse cuenta al final, al ejecutar el proyecto por completo, de que se ha arrastrado un error desde casi el inicio. Este inconveniente podría afectar enormemente al tiempo de realización de la mayoría de los proyectos dentro del simulador.

Por otro lado, el Omnet++ solo tiene soporte para un lenguaje de programación (C++), por lo que limita mucho las opciones para programadores expertos que necesiten utilizar lenguajes de programación más avanzados para tareas mas completas. De todas formas, este lenguaje nos basta para realizar comunicaciones, como es nuestro caso.

Otro problema que quizás no es tan relevante en caso de realizar proyectos no tan complejos es que no dispone de una larga lista de protocolos como otros simuladores. Esto podría ocasionar que, en ocasiones, el usuario se encuentre con algunos errores en el código que no sabe solucionar y que el simulador no ofrezca una solución apropiada. En definitiva, cuantos más protocolos lleve incorporados el simulador, más eficiente será este frente a la detección de errores.

Por último, y aunque sea menos relevante, hay un leve problema de compatibilidad. Hay una mayor posibilidad, frente a otros simuladores, de que se reporte fallos al combinar modelos de diferentes usuarios que hayan sido desarrollados por separado. Cabe la posibilidad de que un grupo de desarrolladores trabajen por separado en un mismo proyecto y, en el momento que junten sus modelos, puede que los programas incluyan ciertos errores que no existirían de haber sido realizados en un mismo simulador.

3.2.4 CONCLUSIONES

A diferencia del apartado de conclusiones del NS3, en este no me voy a extender demasiado, ya que en este caso sí que obtuvimos resultados valiosos en relación con la tecnología BLE, que mostraré en apartados posteriores. Como ya dije antes, el principal motivo por el que cambié del NS3 al Omnet++, fue el hecho de que el módulo BLE del otro simulador no funcionaba correctamente. Por no decir, que las ventajas que ofrece este nuevo simulador frente al NS3, son más relevantes para lo que nuestro proyecto nos exige. De todas formas, es interesante observar algunas prestaciones en las que el NS3 supera al Omnet++, no todo es negativo para el simulador.

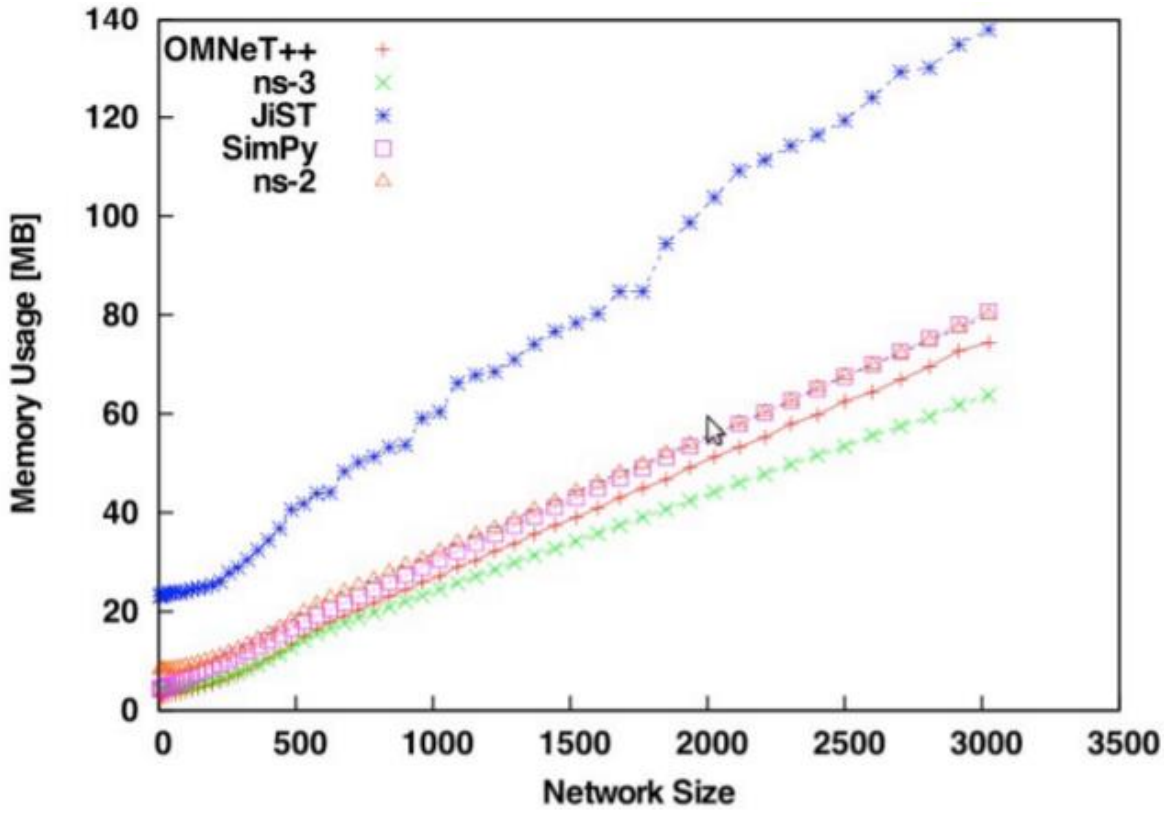


Figura 10 - Comparación de la memoria usada de diferentes simuladores

En la figura 10 podemos ver las diferencias en el uso de memoria entre diferentes simuladores de red. Entre ellos hay que destacar que cuanto mayor es el tamaño de la red, más memoria utiliza el Omnet++ con respecto al NS3.

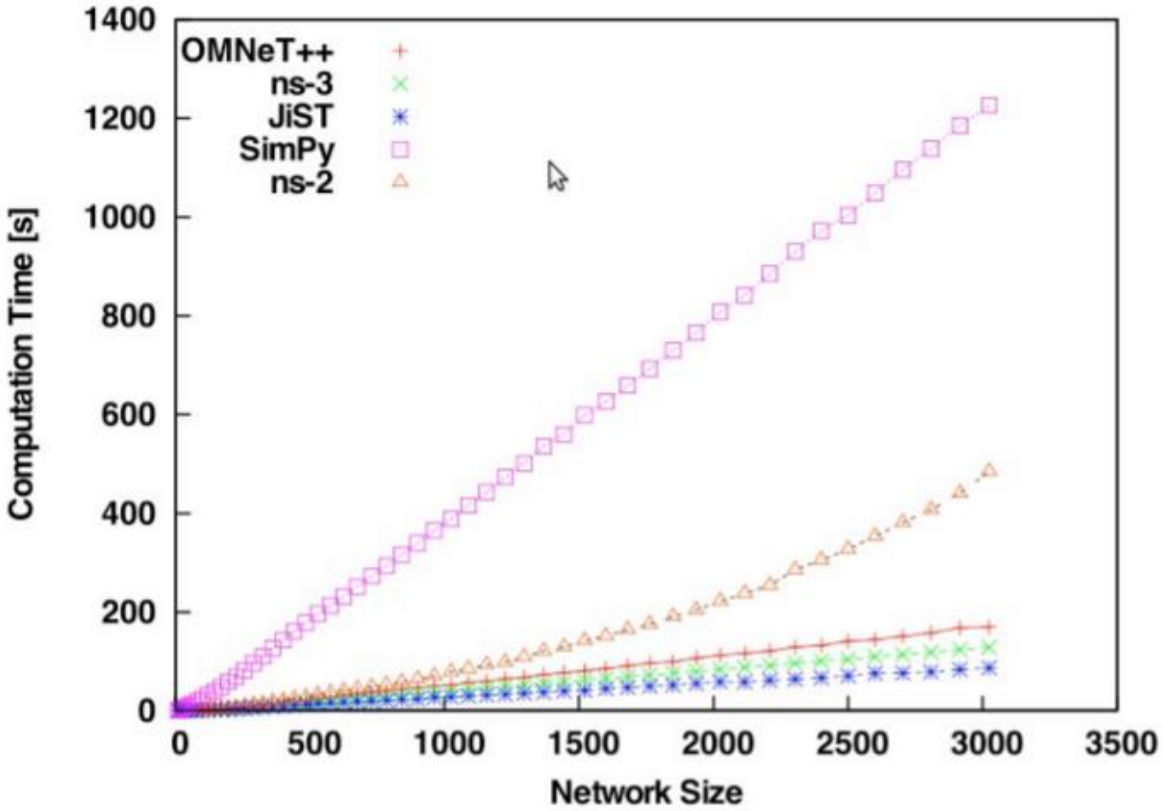


Figura 11 - Comparación del tiempo de computación de diferentes simuladores

En la figura 11 vemos algo similar a la figura anterior, ahora referente a los tiempos. Podes ver que el simulador Omnet++ necesita un mayor tiempo de computación que el NS3 para redes del mismo tamaño. Estas ventajas para el NS3 son debidas a que, al presentar menos prestaciones que otros simuladores (como no tener interfaz gráfica de usuario), puede emplear mayores recursos para otras características como estas vistas en ambas gráficas.

Capítulo 4. ARQUITECTURA

4.1 TOPOLOGÍA DE RED

Los dispositivos BLE pueden tener dos roles distintos: actuar como dispositivos centrales o como periféricos. Los dispositivos centrales suelen ser teléfonos móviles u ordenadores con un elevado poder de procesamiento de la CPU. Por otro lado, los periféricos normalmente son sensores o dispositivos de baja potencia que se conectan a uno central.

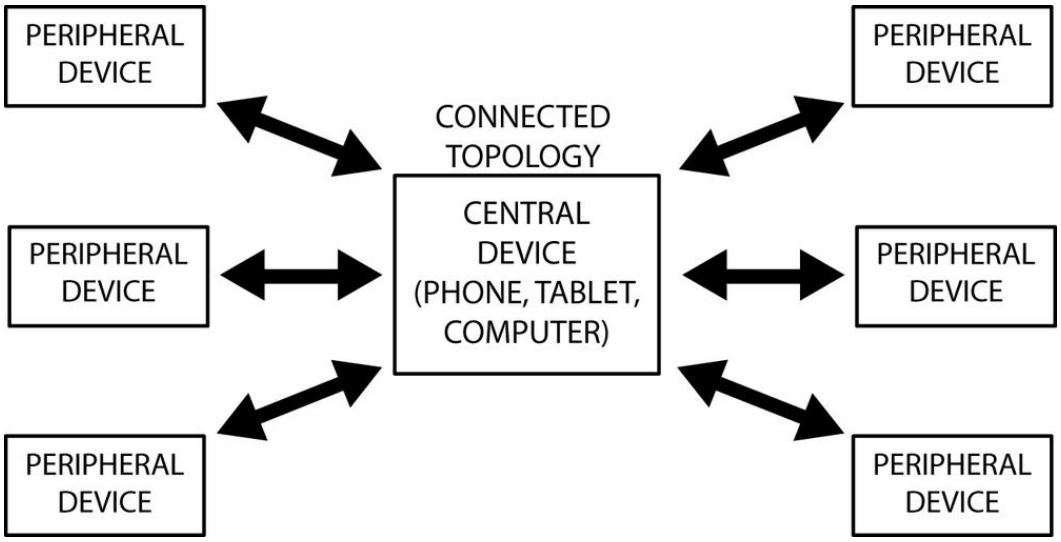


Figura 12 - Topología de una comunicación BLE habitual

En la figura 12 vemos esto comentado, un dispositivo central comunicándose constantemente con otros seis dispositivos periféricos. Esta es la comunicación BLE más habitual, en donde el dispositivo central actúa como "maestro" y los dispositivos periféricos actúan como "esclavos". Esto se debe a que el dispositivo central tiene mucha mayor potencia que los periféricos y estos no podrían transmitir por su cuenta con otros dispositivos de igual potencia.

Un dispositivo BLE puede enviar dos tipos de datos: paquetes de aviso y datos de respuesta de escaneos. Los paquetes de aviso son necesarios para que una comunicación BLE funcione correctamente, y son transmitidos constantemente desde un dispositivo periférico siempre que sea "visto" por otros dispositivos. Cuando otro dispositivo recibe los datos de este periférico, este puede solicitar datos adicionales del periférico. Es entonces cuando comenzaría a enviar los datos de respuesta de escaneos.

Por otro lado, estos dispositivos pueden comunicarse con los dispositivos cercanos de dos maneras distintas: mediante broadcast o mediante simples conexiones. El broadcast es el acto de enviar datos a todos los dispositivos que estén en modo escucha. En este tipo de comunicaciones broadcast, se pueden distinguir dos roles: El broadcaster y el observador. El primero de ellos es el que envía periódicamente paquetes de avisos a cualquier dispositivo que acepte recibirlos. Mientras que el observador es el dispositivo que recibe dichos paquetes de aviso, además de tener la posibilidad de solicitar al broadcaster los datos de respuesta de escaneos. Además, hay que destacar que este modo de comunicación, el broadcast, es la única manera que tiene el dispositivo transmisor de enviar datos a más de un par de destinatarios a la vez.

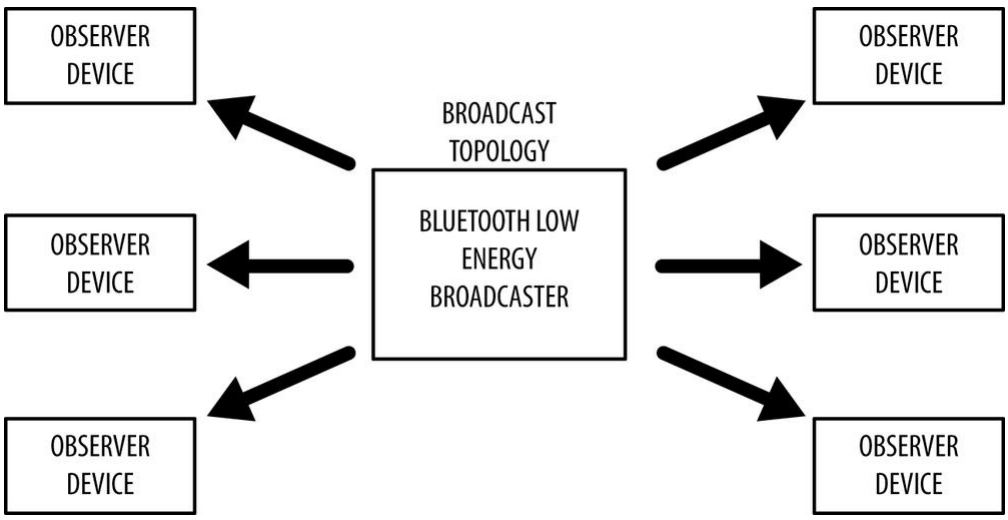


Figura 13 - Topología de una comunicación BLE en modo Broadcast

En la figura 13 vemos lo comentado antes, una comunicación broadcast con un broadcaster y seis observadores. Podemos observar como el dispositivo conocido como broadaster envía mensajes a todos los demás dispositivos observadores. Hay que destacar que este modo de comunicación es muy similar al de una comunicación BLE convencional, ya que hay un dispositivo central comunicándose con varios periféricos u observadores. La diferencia es que en el modo broadcast, el único mensaje que pueden enviar los observadores al broadcaster es el de solicitar los datos de respuesta de escaneo, mientras que en la comunicación BLE los periféricos se intercambian información constantemente con el dispositivo central.

Para finalizar este subapartado, vamos a explicar cómo suele transcurrir una comunicación BLE. En primer lugar, hay que mencionar que una conexión es un permanente y periódico intercambio de datos entre dos dispositivos. En el caso de una comunicación BLE, el maestro (dispositivo central) escanea la frecuencia de los paquetes de aviso a los que se podría conectar y cuando sea posible, establecer la conexión. Una vez que la conexión ha sido establecida, el dispositivo central inicia comienza a transmitir e inicia el contador del intercambio periódico de datos. Mientras que el esclavo (dispositivo periférico) envía los paquetes de aviso periódicamente y acepta la primera solicitud de conexión que le llegue. Cuando se haya establecido la conexión, el periférico adaptará el contador del dispositivo central y comenzará a intercambiar datos regularmente con él. Cuando esto sucede, se conoce como evento de conexión, que es un intercambio periódico de datos en distintos intervalos de tiempo. Además, esta es una de las claves para ahorrar potencia, ya que los dos dispositivos tan solo se encienden, intercambian los datos necesarios e, inmediatamente pasan a modo de espera hasta que llegue el próximo evento.

4.2 APLICACIÓN

Antes de centrarse en explicar la capa de aplicación del bluetooth de baja energía, vamos a ver cómo es la organización en capas de esta tecnología. BLE, como otras muchas tecnologías inalámbricas, está organizada en capas donde cada una tiene su funcionalidad y tiene su propio rol para que un dispositivo BLE funcione correctamente. Todas estas capas pueden llegar a parecer un tanto complejas a primera vista, pero yendo paso a paso y explicando cada de sus campos, podremos entender mejor su funcionamiento. Estas tres capas principales son: aplicación, host y controlador.

La de aplicación es la capa más alta de todas y es la que contiene toda la lógica, la interfaz del usuario y el manejo de datos del dispositivo. Su arquitectura depende enormemente del proyecto desarrollado con BLE, por lo que esta capa será muy diferente en función del dispositivo que necesitemos diseñar, a diferencia del resto de capas, que tienen una organización más común a todos los proyectos BLE.

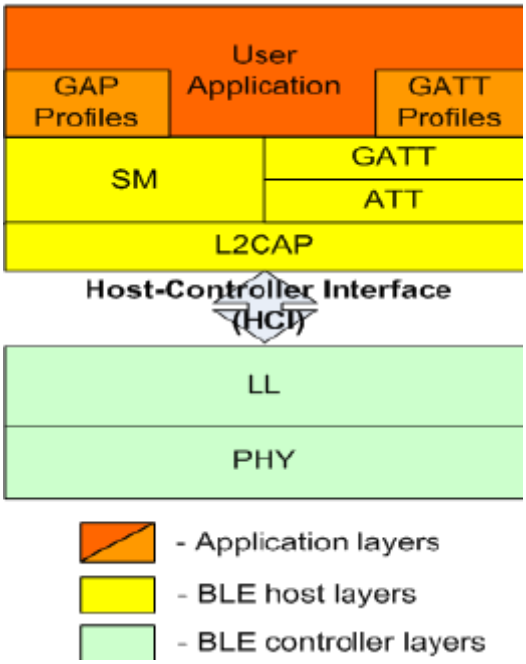


Figura 14 - Distribución de las capas en un dispositivo BLE

En la figura 14 vemos como se distribuyen las capas y subcapas dentro de un dispositivo BLE. Vemos como la capa de aplicación es la más elevada de todas junto con las subcapas de los perfiles genéricos de acceso (GAP) y los perfiles genéricos de atributo (GATT), de la capa de host. Por otro lado, las capas más bajas son la física y la de enlace de datos, ambas de la capa de controlador.

4.3 HOST

La capa de host consta de una serie de subcapas que la componen. En este apartado nos centraremos en explicar cada una de estas capas, que son:

- Perfil genérico de acceso (GAP)
- Perfil genérico de atributo (GATT)
- Protocolo de control lógico y adaptación de enlace (L2CAP)
- Protocolo de atributo (ATT)
- Gestión de seguridad (SM)
- Interfaz de host-controlador (HCI), lado del host

4.3.1 PERFIL GENÉRICO DE ACCESO (GAP)

Esta capa es la responsable de las funcionalidades de la conexión de un dispositivo BLE. Además, se encarga de los modos de acceso y de ciertos procedimientos que lleva a cabo el dispositivo, incluyendo el descubrimiento de otros dispositivos, el establecimiento y terminación de enlaces, la inicialización de las características de seguridad y la configuración del dispositivo.

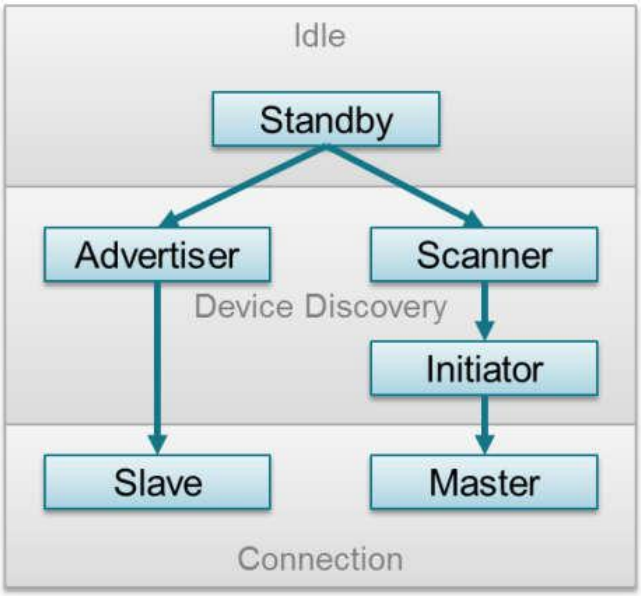


Figura 15 - Diagrama del estado de GAP

En la figura 15 vemos el diagrama del estado de GAP, basado en el rol que el dispositivo tiene configurado. Dicho diagrama muestra los estados del dispositivo, estos son los siguientes:

- Standby: El dispositivo se encuentra en modo inactivo al reiniciarse.
- Advertiser: En este estado, el dispositivo está anunciándose al resto enviando datos a los demás dispositivos con posibilidades de conectarse. Este anuncio contiene la dirección del propio dispositivo y puede contener información adicional como su nombre.
- Scanner: El dispositivo está recibiendo el anuncio y le envía una respuesta de escano al advertiser, entonces éste le vuelve a enviar otra respuesta de escaneo. Este proceso se conoce como descubrimiento. El dispositivo que está en modo scanner es consciente de que el dispositivo en modo advertiser quiere iniciar una conexión con él.

- Initiator: En este estado, el dispositivo debe seleccionar un par de direcciones de dispositivos a los que conectarse. Si se recibe un anuncio con una dirección incluida en ese par de direcciones, entonces se envía una respuesta para establecer el enlace o conexión con el dispositivo que mandó dicho anuncio y asignando los parámetros de conexión apropiados. Estos parámetros son: intervalo de conexión, latencia del esclavo y el tiempo de supervisión.
- Slave/Master: Cuando se ha formado una conexión, el dispositivo funcionará como esclavo si era el que estaba en modo advertiser y como maestro si era el que estaba en modo initiator.

4.3.2 PERFIL GENÉRICO DE ATRIBUTO (GATT)

Esta capa define la estructura jerárquica de los datos que se va a conectar a los dispositivos BLE. Además, permite mantener una completa interoperabilidad con otros dispositivos Bluetooth. El perfil describe un caso de uso, unos roles y unos comportamientos generales basados en la funcionalidad GATT. Los servicios son una colección de características y

relaciones con otros servicios que encapsulan el comportamiento de cada una de las partes de un dispositivo. Esto también incluye una jerarquía de servicios, características y atributos usados en los atributos del servidor.

GATT está construido siguiendo el protocolo de atributo (ATT), el cual usa datos GATT para definir la forma en la que dos dispositivos Bluetooth de baja energía envían y reciben mensajes. Por último, destacar que GATT no se usa en las implementaciones BR/EDR de Bluetooth, las cuales se limitan a adoptar otros perfiles.

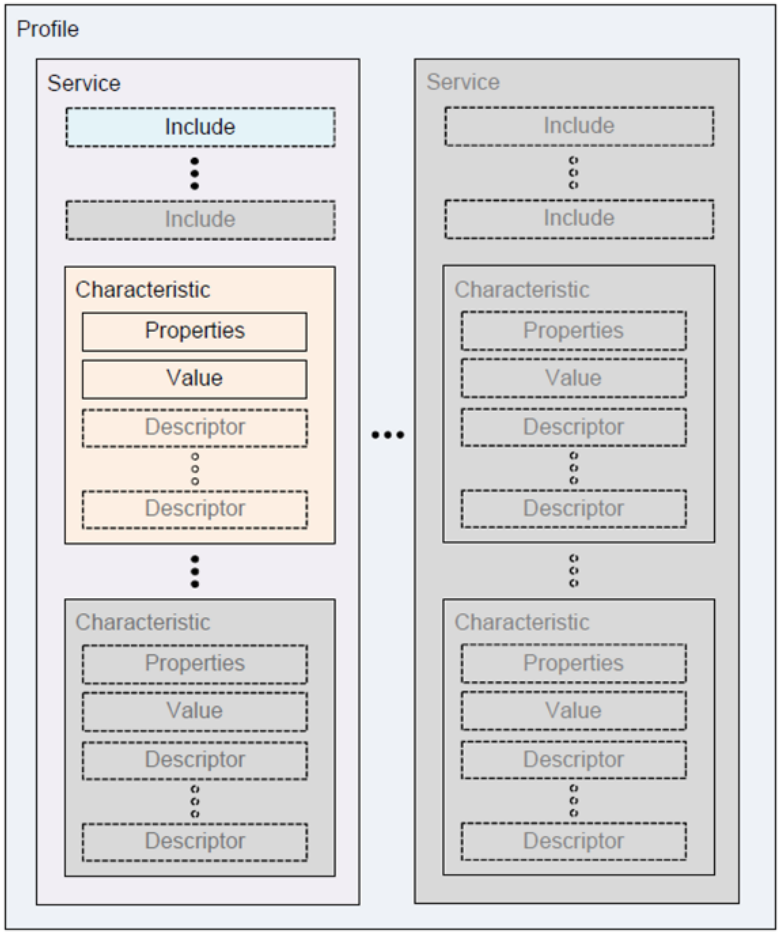


Figura 16 - Jerarquía de los perfiles GATT

En la figura 16 podemos observar la distribución de la jerarquía en un perfil GATT convencional. El nivel más alto de la jerarquía es un perfil, compuesto por uno o más servicios necesarios para cumplir un caso de uso. Por otro lado, un servicio está compuesto de características o referencias a otros servicios. Una característica consta de un tipo (representado por un UUID), un valor, un set de propiedades indicando las operaciones soportadas y un permiso relacionado con la seguridad. Además, puede contener uno o más descriptores, que son metadatos o flags de configuración relacionados con las propias características del dispositivo.

Por otro lado, esta capa también agrupa estos servicios para encapsular el comportamiento de una parte de un dispositivo, y describe un caso de uso, unos roles y uno comportamientos generales basados en las funcionalidades GATT. Este espacio de trabajo define procesos, formatos de servicios y sus propias características, incluyendo el descubrimiento de otros dispositivos, la lectura y escritura de datos y las notificaciones, así como el configurar las características de las conexiones broadcast.

Por último, mientras que la capa GAP se encargaba la mayor parte de las conexiones relacionadas con la funcionalidad, la capa GATT del protocolo BLE es usada por la aplicación para la comunicación de datos entre dos dispositivos conectados. Los datos son transferidos y almacenados en forma de características que son guardadas en la memoria del dispositivo BLE. Desde un punto de vista de GATT, cuando dos dispositivos están conectados, pueden tener uno de estos dos roles:

- Servidor GATT: El dispositivo contiene las características de una base de datos que está siendo leída o escrita por un cliente GATT.
- Cliente GATT: El dispositivo que está leyendo o escribiendo datos para el servidor GATT o de este.

4.3.3 PROTOCOLO DE CONTROL LÓGICO Y ADAPTACIÓN DE ENLACE (L2CAP)

La capa L2CAP se sitúa en lo más alto de la capa HCI, en el lado del host, y transfiere datos entre las capas superiores del host (GAP, GATT, Aplicación) y las capas inferiores de la pila de protocolos. Esta capa es la responsable del protocolo de la capacidad de multiplexación, de la segmentación y del reensamblaje de operaciones para el intercambio de datos entre el host y la pila de protocolos. L2CAP permite la utilización de protocolos de mayor nivel y que las aplicaciones transmitan y reciban paquetes de datos de capas superiores, de hasta 64KB de largo.

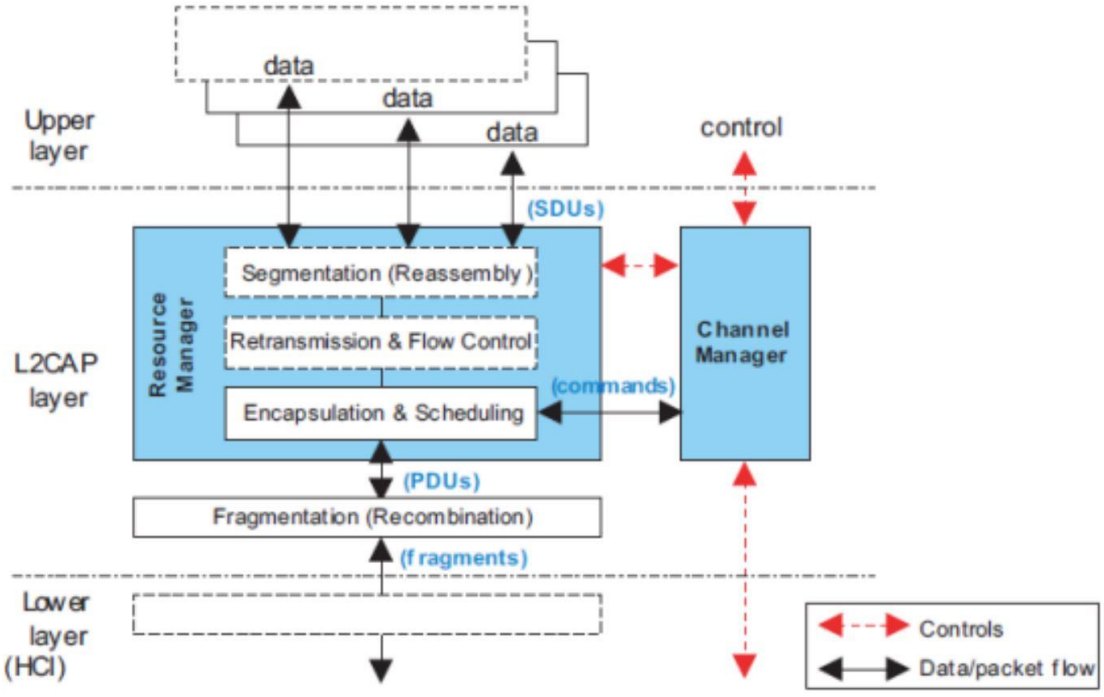


Figura 17 - Arquitectura de bloques de la capa L2CAP

En la figura 17 podemos ver como se organiza la arquitectura de bloques de la capa L2CAP. A la imagen le podemos añadir la terminología general de dicha capa para mejorar el entendimiento de la misma:

- Canal L2CAP: Las conexiones lógicas entre dos puntos finales en un par de dispositivos, caracterizados por sus identificadores de canal (CIDs).
- SDU o L2CAP SDU: Unidad de datos de servicio: un paquete de datos que L2CAP intercambia con una capa superior y transporta de manera transparente datos por un canal L2CAP usando los procedimientos específicos.
- PDU o L2CAP PDU: Unidad de datos de protocolo: un paquete de datos que contiene campos de información del protocolo L2CAP, información del control y información de los datos de capas superiores.
- Máxima unidad de transferencia (MTU): El tamaño máximo de los datos de carga, en octetos, que la capa superior puede aceptar. La MTU corresponde con el tamaño máximo del SDU.
- Tamaño de carga máximo de la PDU: El máximo tamaño de los datos de carga, en octetos, que la capa L2CAP puede aceptar. La MPS corresponde al tamaño de carga máximo de la PDU.

4.3.4 PROTOCOLO DE ATRIBUTO (ATT)

ATT es un simple protocolo de cliente-servidor basado en los atributos presentados por un dispositivo. Un cliente solicita datos de un servidor, y este se los envía. Es importante recordar que, si aún sigue pendiente una solicitud, no se pueden enviar más peticiones hasta que la solicitud haya sido respondida. Cada servidor contiene datos organizados en forma de atributos, y a cada uno se le asignan 16-bits del mango de atributo, un identificador único universal (UUID), un grupo de permisos y un valor. El mango de atributo simplemente es un identificador usado para acceder a un valor de atributo, mientras que el UUID es usado para especificar el típico y la naturaleza de los datos dentro del valor. El cliente envía las respuestas de lectura o escritura pertinentes, y el servidor responde de acuerdo con ello.

Cuando un cliente quiere leer o escribir valores de atributo desde o a un servidor, envía una respuesta de lectura o escritura al servidor para que se haga cargo de ella. Entonces, el servidor responde con el valor de atributo o con una respuesta de reconocimiento. En el caso de que la operación sea de lectura, el cliente debe analizar el valor y entender el tipo de datos basados en el UUID del atributo. Por otro lado, durante las operaciones de escritura, el cliente debe proporcionar datos que correspondan con el tipo de atributo al que se dirige y el servidor será libre de rechazar la operación si no es el caso.

El grupo de operaciones que son ejecutadas bajo el protocolo de atributo (ATT) son las siguientes: Error Handling, Server Configuration, Find Information, Read Operations, Write Operations, Queued Writes y Server Initiated.

4.3.5 GESTIÓN DE SEGURIDAD (SM)

En la capa de Host, hay un módulo llamado Gestor de Seguridad (SM), que define los métodos y protocolos del emparejamiento y la distribución de claves, las correspondientes cajas de instrumentos, y el Protocolo de Gestión de Seguridad (SMP), que define el formato del comando de emparejamiento. La capa SM usa la distribución de claves para mejorar la identidad y las funcionalidades de encriptación en comunicaciones de radio.

El emparejamiento es realizado para estabilizar las claves que pueden ser usadas para encriptar un enlace. Una distribución específica del transporte de las llaves es entonces realizada para compartir estas con el resto de los dispositivos. Las llaves pueden ser usadas para encriptar enlaces en futuras conexiones, para verificar datos firmados o para mejorar la resolución de las direcciones aleatorias. En general, hay 3 fases en el proceso de emparejamiento:

- Fase 1: Emparejamiento de intercambio de características.
- Fase 2 (Emparejamiento legal LE): Generación de claves a corto plazo (STK).
- Fase 2 (Conexiones seguras LE): Generación de claves a largo plazo (LTK).

❖ Fase 3: Distribución del transporte de las claves específicas.

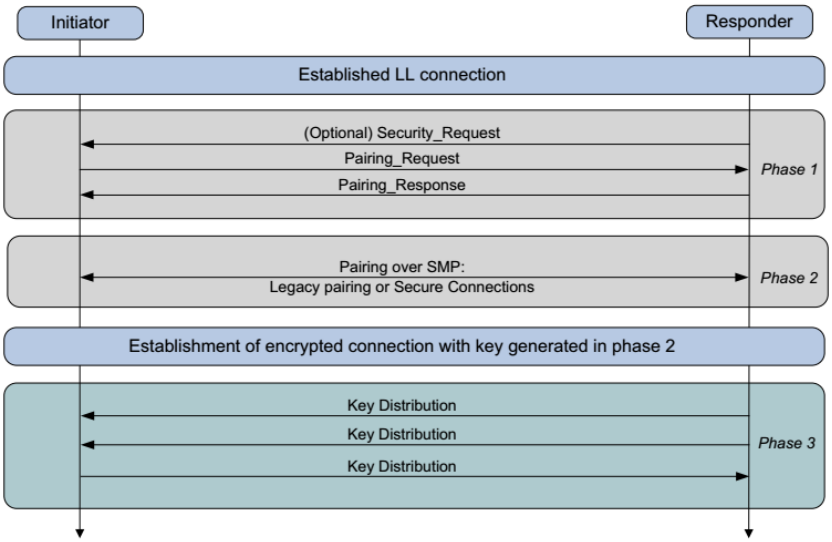


Figura 18 - Distribución de las fases en la etapa de emparejamiento

En la imagen 18 vemos donde aparecen cada una de las fases antes mencionadas en el emparejamiento. Podemos observar las acciones que llevará a cabo cada una de las fases cuando se ha establecido la conexión.

4.3.6 INTERFAZ DE HOST-CONTROLADOR (HCI), LADO DEL HOST

HCI es una fina capa que transporta comandos y eventos entre los elementos del host y los del controlador siguiendo la pila de protocolos de Bluetooth. En una aplicación de procesamiento de la red, la capa HCI es implementada mediante un protocolo de transporte como SPI o UART.

En los proyectos inalámbricos embebidos MCU, como un simple periférico, la capa HCI es implementada mediante funciones de llamada y respuesta con el MCU inalámbrico. Todos los comandos y eventos usados para comunicarse con el controlador, como ATT o GAP, podrán llamar a una HCI API para pasar de capas superiores de la pila de protocolos por la capa HCI al controlador. Además, el controlador envía datos recibidos y eventos al host y a las capas superiores a través de la capa HCI.

Para finalizar con esta capa, hay que destacar que cada mensaje puede tener dos tipos de formato posibles: el paquete de datos y el paquete de anuncio. El primero de ellos tiene una carga útil de datos máxima de 31 bytes, mientras que el segundo tiene una carga útil de datos máxima de 27 bytes. Sin embargo, ambas cargas máximas se suelen ver afectadas y limitadas por protocolos superiores.

4.4 CONTROLADOR

Al igual que mencioné en la capa de Host, en este apartado detallaremos cada una de las subcapas que pertenecen a la capa del controlador. Estas subcapas son:

- Interfaz de host-controlador (HCI), lado del controlador.
- Capa de enlace de datos (LL).
- Capa física (PHY)

4.4.1 INTERFAZ DE HOST-CONTROLADOR (HCI), LADO DEL CONTROLADOR

Esta capa es la que separa la capa del host y la del controlador, por lo que tiene un lado dentro de cada una de las dos capas (de ahí que en cada apartado se mencione qué lado es). El funcionamiento de la capa en sí es el mismo, ya que la capa es la misma, la única diferencia es que en una capa se incluye el lado el HCI incluido dentro de la capa y en la otra se incluye el otro lado. Por lo tanto, al ser igual el funcionamiento, en este apartado no añadiré nada nuevo, ya que se puede ver en la página anterior.

4.4.2 CAPA DE ENLACE DE DATOS (LL)

Mientras que en la capa física se hablaba en resumidas cuentas de la banda de frecuencias que utiliza BLE, en esta capa llamada de enlace de datos es donde verdaderamente sucede el funcionamiento de esta tecnología. Esta capa gestiona las conexiones, y controla todos los paquetes que se envían y se reciben. Un dispositivo BLE puede pasar por una serie de estados, siempre uno al mismo tiempo:

- Espera: El dispositivo ni recibe ni transmite para ahorrar energía, es como si el sistema estuviese dormido.
- Publicidad: En este estado el dispositivo envía paquetes en los canales de publicidad, además de escuchar otras respuestas desde el sistema central. Es un modo difícil de entender debido a que el dispositivo pasará gran parte de su tiempo en él, por lo que el intervalo de tiempo en este modo afectará directamente al consumo, al poder y a la vida del mismo.

- Escaneo: El dispositivo escucha los paquetes de publicidad que se envían por esos mismos canales. Es un modo usado para buscar otros dispositivos mientras se escucha.
- Iniciador: Este es el estado en el que entra un dispositivo central justo antes de establecer una conexión con otro dispositivo. El central escucha anuncios en los periféricos y cuando llega el paquete de publicidad por el periférico deseado, la central puede conectarse enviando los datos correctos a ese otro dispositivo.

Hay que destacar que el estado inicial de todos los dispositivos que actúen como esclavos es el de publicidad. Por otro lado, el estado de conexión es donde el dispositivo periférico (Esclavo) y el dispositivo central (Master) pueden intercambiar datos e información. En BLE los datos se intercambian periódicamente mediante eventos de conexión.

Para enlazar esto con la capa física, los datos se pueden transmitir tan solo en los canales de datos que son los 37 no utilizados por la publicidad, recordemos que 3 de ellos eran para eso en concreto. Los dispositivos que quieran comunicarse se ponen de acuerdo sobre el canal por el que transmitir y alternan el envío de datos estableciendo una serie de normas de comunicación.

4.4.3 CAPA FÍSICA (PHY)

Para comenzar explicando las características de la capa física de BLE, hay que destacar que se relaciona directamente con la manera en la que los dispositivos BLE transmiten y reciben datos. Esta tecnología usa la misma banda de ISM de 2.4GHz, además es común y no requiere de una licencia específica. Esta banda de frecuencias comienza a los 2400MHz y continúa hasta los 2483.5MHz. La versión 4.0 de Bluetooth y sus posteriores especificaciones pueden dividir dicha banda en 40 canales diferentes, donde 3 de ellos son llamados “publicidad” y se utilizan para enviar paquetes de publicidad con información sobre ellos para que otros dispositivos puedan conectarse a ellos a través de BLE. Para evitar la interferencia de las señales Wi-Fi, estos canales suelen seleccionarse de la parte baja, media o alta de la banda.

Por último, cabe destacar que algunas palabras sobre la radio frecuencia están en orden. Al igual que todos los transmisores inalámbricos conocidos, los dispositivos BLE transmiten y reciben señales de radio frecuencia que necesitan de una antena y un cuidadoso diseño para poder funcionar correctamente. Puede llegar a resultar complicado el diseño de la antena, pero cualquiera de 2.4GHz y un ancho de banda necesario se podría llegar a utilizar. Las antenas más comúnmente utilizadas en este tipo de tecnología son las PCB y las Antenas Cerámicas.

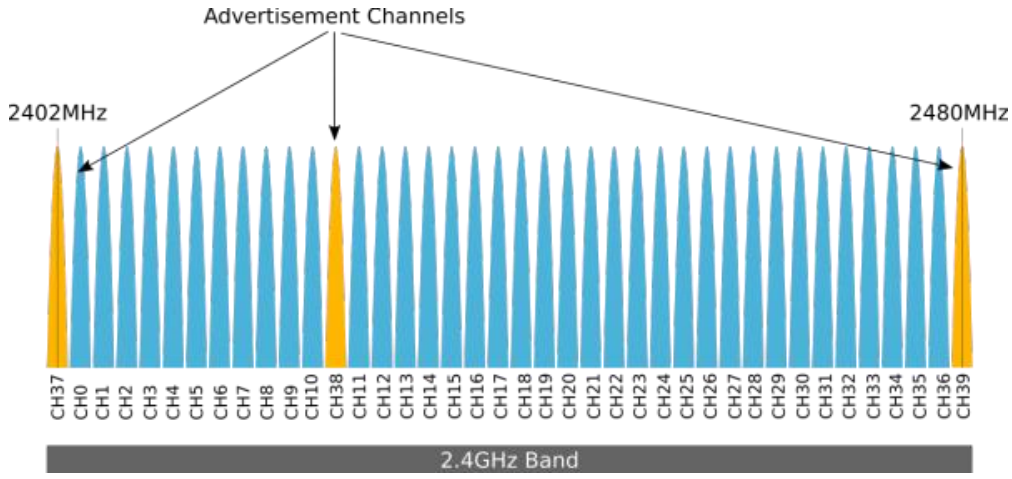


Figura 19 - Rango de frecuencias y canales de anuncio de la tecnología BLE

En la figura 19 podemos observar lo dicho anteriormente, los 40 canales diferentes que utiliza la banda y su rango de frecuencias que va desde 2102MHz hasta 2480MHz, por eso decimos que BLE utiliza la banda de 2.4GHz.

Capítulo 5. MÓDULO BLE

5.1 MIXIM

En este pequeño apartado tan solo voy a mencionar que MiXiM es el framework que utilizo en el simulador Omnet++ para trabajar con el módulo BLE. Además, es el más utilizado por la comunidad para este tipo de simuladores. De todas formas, también se podría haber analizado el módulo BLE sin hacer uso de dicho framework. Sin embargo, el módulo que encontramos por internet hacía uso de MiXiM, por lo que era nuestro deber utilizarlo.

Al importar el framework a nuestro simulador, vimos que venía consigo con una gran cantidad de librerías y proyectos de ejemplo ya realizados que pueden ser de gran ayuda. El módulo BLE que nosotros utilizamos también venía con una serie de librerías. Lo que hicimos fue incorporar dichas librerías a las de MiXiM, para poder trabajar con ellas. Finalmente, todo encajó a la perfección y pudimos comenzar a realizar pruebas del módulo en el framework del simulador Omnet++.

5.2 RED

En este importante apartado procederemos a mostrar las partes más relevantes de la red de la comunicación BLE. Mostraremos un a imagen de cada una ellas y comentaremos sus rasgos más importantes. Más adelante, en el capítulo de resultados, veremos alguna captura de esta misma red en ejecución.

5.2.1 TEST NETWORK

Aquí veremos cómo está organizada la red en el más alto nivel. Todos los elementos pertenecientes, serán debidamente comentados en su apartado correspondiente.

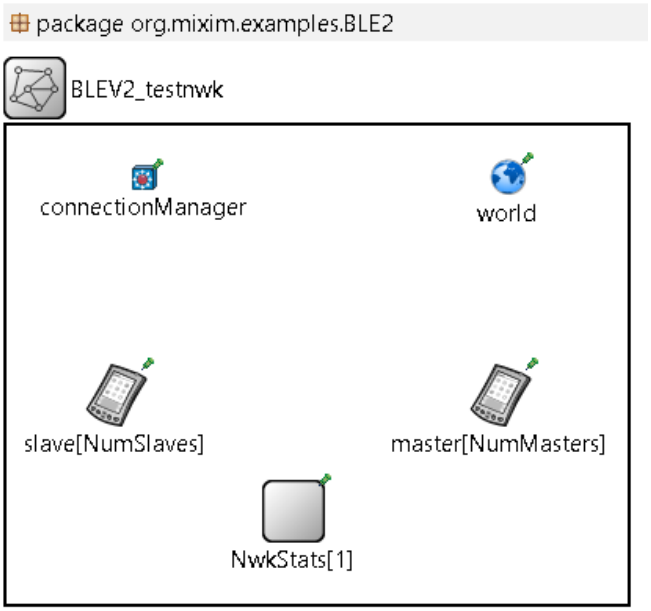


Figura 20 – Elementos generales de nuestra red

En la figura 20, podemos ver alguno de los elementos principales participantes en nuestra red. En primer lugar, "NwkStats[1]" es un módulo que permite registrar la frecuencia usada por cada uno de los canales. "World" es un módulo básico en cualquier red creada en este simulador, proporciona métodos de utilidad y cierta información usada por la mayor parte de los elementos de la red. "ConnectionManager" es el módulo que controla todas las conexiones relacionadas con todos los elementos de la red. Además, es el módulo central

que coordina las conexiones entre todos los nodos, permite la creación de puertas dinámicas y periódicamente se comunica con el módulo de movilidad y el de canal de acceso. Por último, "slave[NumSlaves]" y "master[NumMasters]" hacen referencia a los dispositivos maestro y esclavo dentro de la red.

```
package org.mixim.examples.BLE2;


import org.mixim.base.modules.BaseNetwork;
import org.mixim.modules.node.Host_BLEV2testDevice;
import org.mixim.modules.utility.BLETestNetworkStatistics;

network BLEV2_testnwk extends BaseNetwork
{
    parameters:
        int NumMasters;
        int NumSlaves;

        @display("bgb=396,305");
    submodules:
        master[NumMasters]: Host_BLEV2testDevice {
            parameters:
                @display("b=0.1,0.1,oval,red,red,1;p=315,172");
        }
        slave[NumSlaves]: Host_BLEV2testDevice {
            parameters:
                @display("b=0.1,0.1,oval,green,green,1;p=70,172");
        }
        NwkStats[1]: BLETestNetworkStatistics {

            @display("p=183,246");
        }
    connections allowunconnected:
}
```

Figura 21 – Código de Test Network

En la figura 21 podemos ver el código de Test Network, donde vemos que solo se hace uso de dos parámetros tipo int: "NumMasters" y "NumSlaves", que hacen referencia al número de maestros y de esclavos que participarán en la comunicación, respectivamente. Por otro lado, también tiene tres submódulos que se refieren al dispositivo maestro, al dispositivo esclavo y al módulo de las estadísticas de la propia red. Los dos primeros los comentaremos en mayor profundidad más adelante.

5.2.2 CONNECTION MANAGER

En este apartado mostraré el código de este importante módulo que controla todas las conexiones de la red. Además, se comentarán los parámetros que utiliza para el desempeño de su función.

```
package org.mixim.base.connectionManager;

// Generic ConnectionMananger interface definition.
moduleinterface IConnectionManager
{
    parameters:
        // debug switch for core framework
        bool coreDebug;
        // send directly to the node or create separate gates for every connection
        bool sendDirect;
        // maximum sending power used for this network [mW]
        double pMax @unit(mW);
        // minimum signal attenuation threshold [dBm]
        double sat @unit(dBm);
        // minimum path loss coefficient
        double alpha;
        // minimum carrier frequency of the channel [Hz]
        double carrierFrequency @unit(Hz);
}

simple ConnectionManager like IConnectionManager
{
    parameters:
        // debug switch for core framework
        bool coreDebug;
        // send directly to the node or create separate gates for every connection
        bool sendDirect;
        // maximum sending power used for this network [mW]
        double pMax @unit(mW);
        // minimum signal attenuation threshold [dBm]
        double sat @unit(dBm);
        // minimum path loss coefficient
        double alpha;
        // minimum carrier frequency of the channel [Hz]
        double carrierFrequency @unit(Hz);
        // should the maximum interference distance be displayed for each node?
        bool drawMaxIntfDist = default(false);

        @display("i=abstract/multicast");
}
```

Figura 22 – Código del módulo Connection Manager

En la figura 22 vemos el código del módulo Connection Manager, que solo tiene dos parámetros tipo boolean: "coreDebug", que indica si se va a llevar a cabo una depuración y "sendDirect", que indica si se va a comunicar directamente con el nodo o va a crear puertas separadas para cada una de las conexiones. También incluye cuatro parámetros tipo doublé: "pMax", que indica la cantidad mínima de potencia enviada en miliwatios que puede ser usada por la red; "sat", que indica la cantidad mínima de decibelios que debe tener la atenuación de la señal; "alpha", que indica el mínimo coeficiente de pérdidas que puede tener la conexión en decibelios y "carrierFrecuency", que indica la mínima frecuencia de carga que puede tener el canal en hercios.

5.2.3 WORLD

El módulo World, como ya he dicho en el apartado anterior, desempeña la función de proporcionar distintos métodos de utilidad e información usada por toda la red. Los parámetros que posee controlan el tamaño del área donde se ubican todos los nodos de la red.

```
package org.mixim.base.modules;

//World Utility interface for whole network informations.
moduleinterface IWorldUtility
{
    parameters:
        double playgroundSizeX @unit(m);    // x size of the area the nodes are in (in meters)
        double playgroundSizeY @unit(m);    // y size of the area the nodes are in (in meters)
        double playgroundSizeZ @unit(m);    // z size of the area the nodes are in (in meters)
        bool   useTorus;                    // use the playground as torus?
        @display("i=misc/globe");
}

// Basic utility module for the whole network.
// Provides utility methods and information used by
// the whole network.
simple BaseWorldUtility like IWorldUtility
{
    parameters:
        @class(BaseWorldUtility);
        double playgroundSizeX @unit(m); // x size of the area the nodes are in (in meters)
        double playgroundSizeY @unit(m); // y size of the area the nodes are in (in meters)
        double playgroundSizeZ @unit(m); // z size of the area the nodes are in (in meters)
        bool useTorus = default(false);  // use the playground as torus?
        @display("i=misc/globe");
}
```

Figura 23 – Código del módulo World

En la figura 23 se puede observar el código del módulo World, que utiliza tres parámetros double para controlar el tamaño del área de los nodos de la red: "playgroundSizeZ", que indica en metros el eje x de dicha área; "playgroundSizeY", que indica en metros el eje y del área y "playgroundSizeZ", que indica en metros el eje z de esa misma área. Además, hace usa de una última variable de tipo boolean: "useTorus", que indica si el área tiene forma de toro.

5.2.4 TEST DEVICE

En este apartado se mostrarán cómo están organizados y configurados los dispositivos de la red, independientemente de si actúan como maestros o como esclavos. Estos dispositivos, además de tener los elementos necesarios para funcionar correctamente, tienen una serie de módulos incorporados de la librería del módulo BLE para configurar ciertos parámetros de la comunicación BLE. El esquema de la red y el código es igual si el dispositivo es maestro o esclavo, por lo que tan solo lo mostraré una vez.

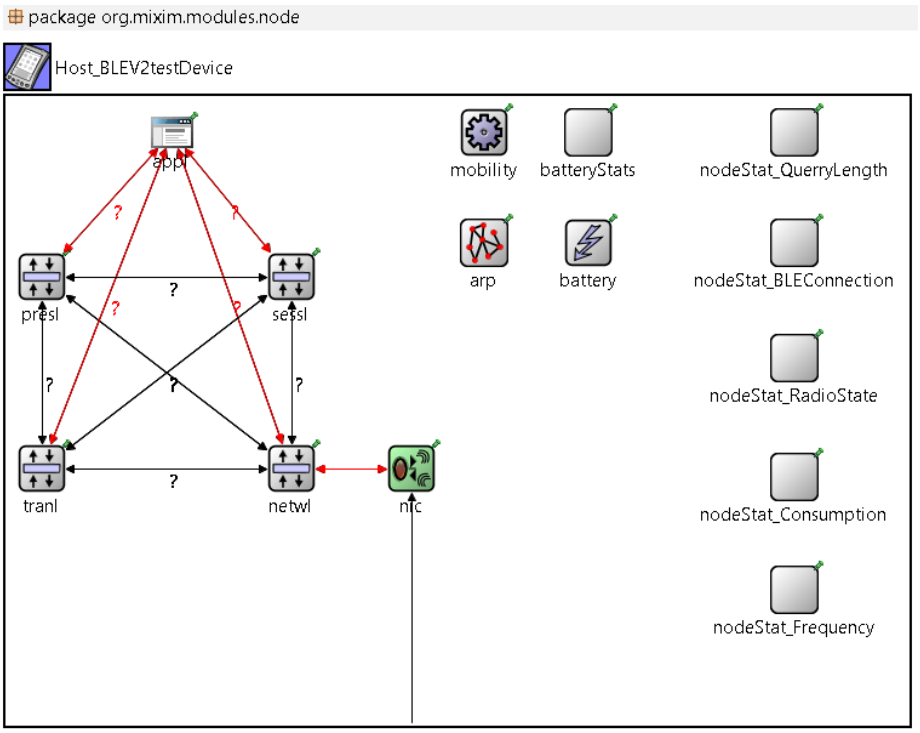


Figura 24 – Organización de los elementos de un dispositivo en nuestra red

En la figura 24 podemos ver cómo están organizados los elementos dentro de la topología del dispositivo. En primer lugar, voy a explicar el papel que desempeñan los módulos de la capa superior (la columna de la derecha en la imagen). "NodeStat_Frequency" es el módulo que permite el registro de la frecuencia usada por todos los canales. "NodeStat_Consumption" es el módulo que proporciona algunas funcionalidades adicionales para obtener el vector de consumo de energía adecuado en cada caso. El resto de los módulos también permiten registrar ciertos valores que influyan en la comunicación. Todos estos módulos tienen un solo parámetro tipo boolean que indica si estará activo o no, por defecto no están activos estos módulos.

En la siguiente capa se incorporan dos módulos más relacionados con la batería del dispositivo. El primero de ellos, denominado "battery Stats" es el módulo que recoge todas las estadísticas e información de la batería del dispositivo. Los parámetros que incorpora son cuatro booleans, donde se destaca el poder elegir escribir o no cierta información en los archivos escalares y vectoriales resultantes de la simulación. El segundo de ellos se llama "battery", y es un simple módulo de batería obtenido de la librería de módulos de MiXiM. De entre sus parámetros destaca la capacidad de batería nominal (mAh), la capacidad de batería normal (mAh), el voltaje nominal (V), el tiempo de resolución (s) y el número de dispositivos que tiene la comunicación.

En la tercera y última capa se añaden dos módulos más. "Mobility" es el módulo relacionado con la movilidad, como su propio nombre indica, de la comunicación y "arp" es el módulo responsable de la resolución de direcciones dentro de la comunicación. Incluye algunos parámetros tipo boolean indicando si el dispositivo se verá afectado por el estado del host y si se piensa realizar una depuración o no.

Por último, además del módulo "nic" que veremos en el siguiente apartado, hay que destacar los cuatro módulos pertenecientes a la capa de red ("netwl, "tranl", "sessl" y "presl") y el módulo de la capa de aplicación "app". Todos ellos están conectados entre sí y al módulo nic.

En cuanto al código de todo lo anterior comentado, vamos a observar el de las tres primeras capas antes mencionadas, viendo cómo se relacionan los parámetros de dicho código con los módulos de la capa en cuestión.

```
package org.mixim.modules.node;

import org.mixim.modules.utility.Stat_ConsumptionChart;
import org.mixim.modules.utility.Stat_FrequencyHops;
import org.mixim.modules.utility.Stat_RadioState;
import org.mixim.modules.utility.Stat_QuerryData;
import org.mixim.modules.utility.Stat_BLEConnection;

module Host_BLEV2testDevice extends Host_BLEV2
{
    parameters:
        bool LogCurrent;
        bool LogFrequency;
        bool LogRadioState;
        bool LogQuerryLgth;
        bool LogBLEConnection;

        @display("bgb=746,519");
    submodules:
        nodeStat_Consumption: Stat_ConsumptionChart {
            Log = LogCurrent;
            @display("p=651,314");
        }
        nodeStat_Frequency: Stat_FrequencyHops {
            Log = LogFrequency;
            @display("p=651,407");
        }
        nodeStat_RadioState: Stat_RadioState {
            Log = LogRadioState;
            @display("p=651,216");
        }
        nodeStat_QuerryLength: Stat_QuerryData {
            Log = LogQuerryLgth;
            @display("p=651,30");
        }
        nodeStat_BLEConnection: Stat_BLEConnection {
            Log = LogBLEConnection;
            @display("p=651,121");
        }
}
```

Figura 25 – Código de la primera capa de Test Device

En la figura 25 podemos ver como los cinco parámetros tipo boolean que hay, son los mismos que incluía cada uno de los módulos de esta capa respectivamente. En el código podemos ver cada uno de estos cinco submódulos y como están ubicados en el display.

```
package org.mixim.modules.node;

import org.mixim.modules.node.WirelessNodeBatteryCurrentChart;

module Host_BLEV2 extends WirelessNodeBatteryPlusTranCurrentChart
{
    parameters:
        applicationType = default("SensorApplLayer");
        transportType = default("Aggregation");

        nicType = "NicBLEV2";
        arpType = default("org.mixim.modules.netw.ArpHost");
        batteryStats.detail = default(false);
        batteryStats.timeSeries = default(false);

        battery.nominal = default(2000 mAh);
        battery.capacity = default(battery.nominal);
        battery.voltage = default(3 V);
        battery.resolution = default(60 s);
        battery.publishDelta = default(1);
        battery.publishTime = default(battery.resolution);
        battery.numDevices = default(1);

        @display("bgb=748,521");
    submodules:
        //ConsumptionProfile: Stat_ConsumptionChart{
        //    Log=true;
        //}

        //RadioStateProfile: Stat_RadioState{
        //    Log=true;
        //}
}
```

Figura 26 – Código de la segunda capa de Test Device

En la figura 26 podemos ver la asignación de los valores por defecto de los parámetros del submódulo de batería que comentamos en el apartado anterior como la capacidad, el voltaje, la resolución o el número de dispositivos. Además, hay un par de parámetros relacionados con el submódulo “battery Stats”, que son los mismos que vimos al comentar dicho submódulo.

```
package org.mixim.modules.node;

import org.mixim.modules.power.battery.BatteryStats;
import org.mixim.modules.power.battery.SimpleBatteryCurrentChart;

module WirelessNodeBatteryCurrentChart extends WirelessNode
{
    parameters:
        applicationType = default("BurstApplLayerBattery");  //type of the application layer
         //@display("p=140,310");
        @display("bgb=746,450");
    submodules:
        batteryStats: BatteryStats {
            //@display("p=140,240;i=block/table,#FF8040");
            @display("p=481,30");
        }
        battery: SimpleBatteryCurrentChart {
            //@display("p=140,170;i=block/plug,#FF8000");
            @display("p=481,121");
        }
}

module WirelessNodeBatteryNetwlCurrentChart extends WirelessNodeBatteryCurrentChart
{
    parameters:
        transportType = default("");    //type of the transport layer
        sessionType = default("");      //type of the session layer
        presentationType = default(""); //type of the presentation layer
        applicationType = default("");  //type of the application layer

    connections allowunconnected:
}

module WirelessNodeBatteryPlusTranCurrentChart extends WirelessNodeBatteryCurrentChart
{
    parameters:
        sessionType = default("");      //type of the session layer
        presentationType = default(""); //type of the presentation layer
        applicationType = default("");  //type of the application layer
}
```

Figura 27 – Código de la tercera capa de Test Device

En la figura 27 tenemos el código de la tercera capa de Test Device, donde por un lado podemos ver dos submódulos relacionados también con la batería y una serie de parámetros dentro de dos de los módulos principales que hacen referencia a las distintas capas: transporte, sesión, presentación y aplicación.

5.2.5 MÓDULO NIC

Nic es la abreviatura de "Network interface controller" y como su propio nombre indica es una tarjeta de red que se encarga de controlar los recursos de los que dispone un dispositivo y compartirlos con otros. El framework MiXiM ya traía incorporado este módulo Nic para diversos tipos de redes, sin embargo, no estaba el de una comunicación BLE. De todas formas, esa librería la implementamos desde el módulo BLE adquirido previamente desde internet. En el capítulo siguiente veremos que desempeña un papel fundamental durante una comunicación BLE entre varios dispositivos. En este apartado nos centraremos únicamente en comentar sus elementos y sus parámetros de configuración.

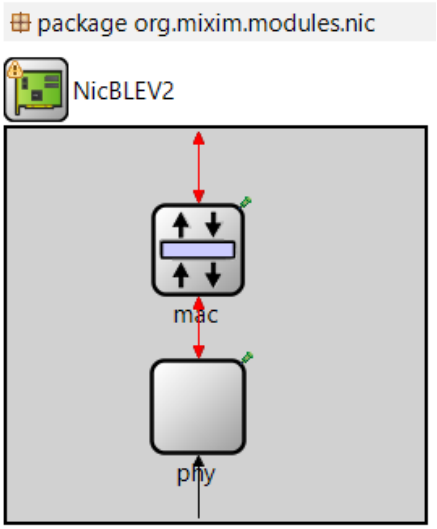


Figura 28 – Elementos de la tarjeta de red de un dispositivo en nuestra red

En la figura 28 podemos ver dos elementos claves para el correcto funcionamiento de la tarjeta de red de un dispositivo. Uno de ellos es "phy", el módulo de la capa física inalámbrica del dispositivo. De entre sus parámetros destacan una serie de tiempos tipo double relacionados con el envío y recepción de datos de conmutación, así como el tiempo de espera del mismo. Así como otros que indican la longitud de la cabecera, la sensibilidad, la máxima potencia de transmisión o el ruido ocasionado.

El segundo de sus elementos es "mac", que es el módulo de todas las capas mac participantes en la comunicación. Tiene una serie de puertas de entrada y salida, algunas conectadas con el módulo "phy" y otras conectadas a otros elementos de control del dispositivo. En el capítulo siguiente veremos como desempeña un papel importante. Por otro lado, tiene tan solo dos parámetros, uno tipo int que indica la longitud de la cabecera del paquete MAC y otro de tipo string que hace referencia a la dirección MAC generada para el dispositivo.

```
package org.mixim.modules.nic;

import org.mixim.modules.mac.BLEMacV2;
import org.mixim.modules.phy.PhyLayer;
import org.mixim.modules.utility.Stat_FrequencyHops;
//"Ideal" BLE transceiver
module NicBLEV2 extends WirelessNicBattery
{
    parameters:
        @display("i=block/ifcard;bgb=258,243");
        macType = "BLEMacV2";

        mac.queueLength = default(20);
        mac.bitrate = default(1E+6bps);
        mac.autoBitrate = default(false);
        // values if no fading is modelled, gives at most 1% packet error rate
        mac.snr2Mbit = default(1.46dB);
        mac.snr5Mbit = default(2.6dB);
        mac.snr11Mbit = default(5.68dB);
        mac.neighborhoodCacheSize = default(30);
        mac.neighborhoodCacheMaxAge = default(100s);
        mac.txPower = default(10.0mW);
        mac.llDataHeaderLengthBits = default(16bits);


        phy.headerLength = default(64bit);//Preamble+Access Address + CRC according to BLE spec. 4.1
        phy.sensitivity = default(-70dBm);//minimum sensitivity according to BLE spec. 4.1
        phy.decider = default(xmldoc("NicBLE_Decider.xml"));
        phy.initialRadioChannel = default(1);
        phy.nbRadioChannels = default(40); //

        phy.timeRXToTX = 0.000150s;//
        phy.timeRXToSleep = 0s;
        phy.timeTXToRX = 0.000150s;
        phy.timeTXToSleep = 0s;
        phy.timeSleepToRX = 0.000004s;
        phy.timeSleepToTX = 0.000004s;
}
```

Figura 29 – Código del módulo NIC

En la figura 29 vemos el código de la tarjeta red de nuestro dispositivo en donde se pueden distinguir dos tipos de parámetros, unos hacen referencia al módulo mac y otros al módulo phy. Todos ellos están configurados de la manera deseada, añadiendo por ejemplo todos los tiempos y datos como la cabecera o la sensibilidad en el módulo ble, o todos los valores por defecto relacionados con ciertas atenuaciones, longitudes o velocidad de transmisión en el módulo mac.

5.2.6 MÓDULO MAC

El módulo Mac no solo es importante a la hora de asignar la dirección MAC de cada dispositivo, sino que también incorpora ciertos parámetros para controlar los tiempos de conmutación y es importante en los procesos de inicialización, conexión y pruebas tras la simulación. Digamos que es uno de los módulos más importantes utilizados. Al igual que el módulo Nic del apartado anterior, este ya viene incorporado en la librería de MiXiM, sin embargo, el módulo BLE adquirido por internet ya traía incorporada una librería con varios módulos, entre ellos este que estoy comentando.

En este apartado mostraré el código de este módulo y, debido a la gran cantidad de parámetros, me limitaré a comentarlos muy por encima. En el apartado de inicialización veremos los valores asignados a estos parámetros y en el capítulo de resultados nos daremos cuenta de su gran importancia en la comunicación.

```
package org.mixim.modules.mac;

import org.mixim.base.modules.BaseMacLayer;

simple BLEMacV2 extends BaseMacLayer
{
    parameters:
        @class(BLEMacV2);
        // debug switch
        bool debug = default(false);
        bool stats = default(true);
        bool trace = default(false);

        double bitrate @unit(bps) = default(1000000 bps);
        // length of MAC header - not really used by BLE (BLE uses llDataHeaderLengthBits instead)
        headerLength @unit(bit) = default(72 bit);

        //desired tx power [mW]
        double txPower @unit(mW) = default(1mW); //i.e., 0 dbm
        int Initial_State = default(0) ;

//BLE standard stuff
        int llDataHeaderLengthBits @unit(bit)= default(16 bit);//according to the standard - see p. 2511 of spec v4.1
        int llAdvHeaderLengthBits @unit(bit)= default(16 bit);//according to the standard -
        double llIFSDeviation @unit(s) = default(0.000002 s);//according to the standard - see p. 2524 of spec v4.1
        double llIFS @unit(s) = default(0.000150 s);//duration of IFS - according to the standard
        double llmaxAdvPDUduration @unit(s) = default(0.000376 s);//according to the standard
        double llmaxDataPDUduration@unit(s) = default(0.000328 s);//according to the standard
        double llminDataPDUduration@unit(s) = default(0.000088 s);//according to the standard
        int llmaxDataPDUPayloadBytes @unit(byte)= default(27 byte);//according to the standard

//HW stuff required for correct timing
        //Switching times
        double Time_llSLEEPtoTX @unit(s) = default(0.0 s);//
        double Time_llSLEEPtoRX @unit(s) = default(0.0 s);//
        double Time_llTXtoRX@unit(s) = default(0.000150 s);//should be less or equal to IFS (otherwise MIGHT not work)
        double Time_llRXtoTX@unit(s) = default(0.000150 s);//should be less or equal to IFS (otherwise MIGHT not work)
        double Time_llTXtoSLEEP@unit(s) = default(0.0 s);//
        double Time_llRXtoSLEEP@unit(s) = default(0.0 s);//
```

Figura 30 – Primera parte del código del módulo MAC

```
//CONNECTION
    //for initialization
        int transmitWindowOffset = default(0); //in 1.25ms units, range [0, connInterval], see p. 2540 of spec v4.1
        int transmitWindowSize = default(1);//in 1.25ms units, range [1.25 ms, min(10 ms, (connInterval-1.25ms))]

    //for connection
        int advertisingAddr = default(-1);
        int connAccessAddress = default(777);
        int connStartingChannel = default(0);//
        int nodeSCA = default(0);

        string slave_beacon_ReplyPolicy = default("ondata");//defines the policy of the slave's reply in case if there
                                                            //is no MD set in a beacon and if slave has no data to TX
                                                            //Can be one of: "ondata","always"

//TESTING and debug parameters
        int Init_DataQueryLgth @unit(byte)= default(0 byte);//initial nuber of bytes in the LL data buffer
        bool TST_stopSimulation_ConnectionBreak = default(false);
        bool TST_NoRandomAdvInt = default(false);//true = do not use random component of AdvInterval
        int  TST_ForceHop = default(-1);//force specific hop value


        bool TST_CollisionMonitoringMechanism = default(false);//true - use connection monitoring mechanism
        string TST_CMM_DetectionPolicy  = default("DropTimeInRangeRelative");
        int TST_CMM_NumPoints = default(3);
        double TST_CMM_Par1 = default(0.5);
        double TST_CMM_Par2 = default(0.1);
        double TST_CMM_Offset_Const_Relative = default(1);
        double TST_CMM_Offset_Random_Relative = default(0);

//HW params
        int macBufferSize @unit(byte) = default(1000 byte);

}
```

Figura 31 – Segunda parte del código del módulo MAC

En las figuras 30 y 31 tenemos el código de este módulo, donde podemos ver algunos parámetros relacionados con el debug y los tiempos de conmutación, siendo estos últimos muy importantes para que todo funcione correctamente. Por otro lado, también se asignan valores importantes en la comunicación como son la velocidad de transmisión y la potencia transmitida. Nos encontramos también con una serie de parámetros sobre tamaños de la cabecera MAC y sobre otros tiempos de la PDU. Por último, podemos ver que se configura lo necesario para que las conexiones fluyan de la manera correcta, siendo importantes los parámetros de inicialización, conexión y pruebas.

5.2.7 MÓDULO NETWORK

El módulo Network, Basic Network o Red en español, extiende el estándar de la capa de red para proporcionar soporte a controles de potencia dentro de la comunicación. Este es el último de los tres grandes módulos que veremos que se utilizan con mayor frecuencia para que funcione la simulación. Al igual que los anteriores, este módulo venía incorporado en el módulo BLE que obtuvimos por internet, tan solo tuvimos que importarlo a la librería de MiXiM de nuestro simulador.

```
package org.mixim.modules.netw;
import org.mixim.base.modules.BaseNetwLayer;

simple BLE_BasicNwk extends BaseNetwLayer
{
    parameters:
        // debug switch
        bool debug = default(true);
        bool trace = default(false);
        bool useSimTracer = default(false);

         //TODO: remove
        int NWK_DEBUG_ROLE = default(0);

        int connAdvertiseChannelMap_0to7 = default(0x00);
        int connAdvertiseChannelMap_8to15 = default(0x00);
        int connAdvertiseChannelMap_16to23 = default(0x00);
        int connAdvertiseChannelMap_24to31 = default(0x00);
        int connAdvertiseChannelMap_32to39 = default(0xE0);

        int connDataChannelMap_0to7 = default(0x00);
        int connDataChannelMap_8to15 = default(0xFE);
        int connDataChannelMap_16to23 = default(0x00);
        int connDataChannelMap_24to31 = default(0x00);
        int connDataChannelMap_32to39 = default(0x00);

        int advInterval = default(160);//in 0.626 ms units,
        int scanInterval = default(160);//in 0.626 ms units, Range: 0x0004 â€“ 0x4000, see p.1261 of Spec v4.1
        int scanWindow = default(160);//in 0.626 ms units, Range: 0x0004 â€“ 0x4000, see p.1261 of Spec v4.1
        int connInterval = default(8);//in 1.25 ms units
        int connLatency = default(0);//SHOULD BE 0
        int supervision_Timeout = default(8); //in 10 ms units

        double nodeStartupDelay @unit(s)= default(0 s);

        int initialData @unit(byte) = default(1000 byte);

        bool useSuggestedFastIntervalSturtupMechanism = default(false);//activate startup with fast intervals

        @display("i=block/fork");
        @class(BLE_BasicNwk);
}
```

Figura 32 – Código del módulo Network

En la figura 32 nos encontramos con el código del módulo en cuestión, donde vemos que hay dos grandes grupos de parámetros para configurar los dispositivos dentro de la comunicaicón. Estos son los “AdvertiseChannel” y los “DataChannel”, que traducidos literalmente serían canales de publicidad y canales de datos. Los parámetros de canales de datos sólo se asignan a los dispositivos cuyo rol es el de iniciador o maestro (otro de los parámetros consiste en la asginación de este rol para cada dispositivo: maestro o esclavo), mientras que los de canales de publicidad se asignan a la red en general. Por otro lado, podemos ver otros parámetros relacionados con algunos intervalos, tiempos y retrasos en la comunicació, así como otro que incica la cantidad de datos (en bytes) iniciales que tiene cada dispositivo y uno último de tipo boolean que menciona si se emplearán intervalos rápidos o no.

5.3 INICIALIZACIÓN

En una simulación con Omnet ++ siempre hay un archivo de inicialización con extensión ".ini" que otorga valores a los parámetros que se quieran cambiar (si no se tocan, se asignan los por defecto) para correr el programa de la manera deseada. Nuestro fichero de inicialización tiene una gran cantidad de asignación de parámetros que dividimos en secciones según el papel que desempeñen en la comunicación. Todos estos parámetros son extraídos de ciertos elementos participantes en la comunicación en más o menos alto nivel. Sin embargo, hay otros que son extraídos de ciertos módulos BLE implementados, como pueden ser el "BLEMacV2", el "BLENicV2" o el "BLE_BasicNwk" (Módulos Mac, Nic y Network).

Por otro lado, en esta apartado también comentaremos alguno de estos parámetros y su función en la comunicación BLE. Más adelante, en el capítulo de resultados, sería buena idea ir cambiando el valor de alguno de estos parámetros para ver como influye en los resultados finales de la simulación y obtener unas conclusiones más sólidas.

```
[General]
experiment-label = "BLEV2_TEST"
network = BLEV2_testnwk
**.vector-recording = true

############################################################
################### Display ################################
############################################################
## BLEV2_testnwk.ned -> BaseNetwork.ned

**.playgroundSizeX = 1000 m # x size of the area the nodes are in (in meters)
**.playgroundSizeY = 1000 m # y size of the area the nodes are in (in meters)
**.playgroundSizeZ = 1000 m # z size of the area the nodes are in (in meters)


############################################################
################### Channel parameters #####################
############################################################
## BLEV2_testnwk.ned -> ConnectionManager.ned

**.connectionManager.sendDirect = false           # send directly to the node or create separate gates for every connection
**.connectionManager.pMax = 10 mW                 # maximum sending power used for this network [mW]
**.connectionManager.sat = -90 dBm                # minimum signal attenuation threshold [dBm]
**.connectionManager.alpha = 2.0                  # minimum path loss coefficient
**.connectionManager.carrierFrequency = 2.45e+9 Hz # minimum carrier frequency of the channel [Hz]


############################################################
################### Battery ################################
############################################################
## BLEV2_testmwk.ned -> Host_BLEV2testDevice.ned -> SimpleBatteryCurrentChart.ned -> SimpleBattery.ned

**.battery.nominal = 2000 mAh  # nominal battery capacity
**.battery.capacity = 2000 mAh # battery capacity
**.battery.resolution = 30 s   # capacity is updated at least every resolution time
**.battery.publishDelta = 0    # (0..1): capacity is published each time it is observed to have changed by publishDelta * nominal_capacity
**.battery.publishTime = 30 s  # if > 0, capacity is published to the BB each publishTime interval
**.battery.voltage =3 V        # nominal voltage
```

Figura 33 – Primera parte del código del fichero de inicialización

```
############################################################
################### Phy Layer parameters ###################
############################################################
## Modules -> NicBLEV2.ned -> IWirelessPhy.ned

**.nic.phy.usePropagationDelay = false          # should transmission delay be simulated?
**.nic.phy.thermalNoise = -100 dBm              # the strength of the thermal noise [dBm]
**.nic.phy.useThermalNoise = true               # should thermal noise be considered?
**.nic.phy.analogueModels = xmldoc("config.xml") # specification of the analogue models to use and their parameters
**.nic.phy.decider = xmldoc("config.xml")       # specification of the decider to use and its parameters
**.nic.phy.initialRadioState = 2                # state the radio is initially in (0=RX, 1=TX, 2=Sleep)
**.nic.phy.sensitivity = -93 dBm                # the sensitivity of the physical layer [dBm]
**.nic.phy.maxTXPower = 10.0 mW                 # the maximum transimission power of the physical layer [mW]
**.nic.phy.nbRadioChannels = 40                 # number of available radio channels. Defaults to single channel radio.

############################################################
################### Current consumption ####################
############################################################
## Modules -> NicBLEV2.ned -> WirelessNicBattery.ned

**.nic.phyType = "org.mixim.modules.phy.PhyLayerBattery"
**.nic.sleepCurrent = 0.0235 mA    #CC2540 datasheet (SWRS084F) p.4
**.nic.rxCurrent = 22.1 mA         #CC2540 datasheet (SWRS084F) p.5 (High-gain mode)
**.nic.decodingCurrentDelta = 0 mA
**.nic.txCurrent = 27mA            #CC2540 datasheet (SWRS084F) p.6 for 0dbm
**.nic.setupRxCurrent = 11.05 mA   #just half of rxCurrent
**.nic.setupTxCurrent = 13.5 mA    #just half of txCurrent
**.nic.rxTxCurrent = 13.5 mA       #just half of txCurrent
**.nic.txRxCurrent = 11.05 mA      #just half of rxCurrent

############################################################
################### Switching times ########################
############################################################
## Modules -> BLEMacV2.ned

**.mac.Time_llSLEEPtoTX = 0.000004 s
**.mac.Time_llSLEEPtoRX = 0.000004 s
**.mac.Time_llTXtoRX = 0.000150 s
**.mac.Time_llRXtoTX = 0.000150 s
**.mac.Time_llTXtoSLEEP = 0 s
**.mac.Time_llRXtoSLEEP = 0 s
**.mac.TST_stopSimulation_ConnectionBreak = false
```

Figura 34 – Segunda parte del código del fichero de inicialización

```
#################################################################
#################### Mobility ###################################
#################################################################
## BLEV2_testnwk.ned -> Host_BLEV2testDevice.ned -> Host_BLEV2.ned -> WirelessNodeBatteryCurrentChart.ned -> WirelessNode.ned

**.mobilityType = "StationaryMobility"


#################################################################
#################### NWK & APP pars #############################
#################################################################
## BLEV2_testnwk.ned -> Host_BLEV2testDevice.ned -> Host_BLEV2.ned -> WirelessNodeBatteryCurrentChart.ned -> WirelessNode.ned

**.networkType = "BLE_BasicNwk"

## BLEV2_testnwk.ned -> Host_BLEV2testDevice.ned -> IBaseNetwLayer.ned

**.netwl.headerLength = 0 bit

## BLEV2_testnwk.ned -> Host_BLEV2testDevice.ned -> IBaseApplLayer.ned

**.appl.headerLength = 0 byte


#################################################################
#################### Some MAC consts ############################
#################################################################
## Modules -> BLEMacV2.ned

**.nic.mac.llDataHeaderLengthBits = 16 bits
**.nic.mac.llIFSDeviation = 0.0000002 s
**.nic.mac.llIFS = 0.000150 s
**.nic.mac.llmaxAdvPDUduration = 0.000376s     # maximum duration of a advertisement channel packet
**.nic.mac.txPower = 1mW                       # 0dbm - desired TX power
**.nic.mac.slave_beacon_ReplyPolicy = "ondata" # "ondata" or "always"
```

Figura 35 – Tercera parte del código del fichero de inicialización

```
#################################################################
#################### Output data ################################
#################################################################
## BLEV2_testnwk.ned -> Host_BLEV2testDevice.ned

**.LogCurrent=true
**.LogFrequency=true
**.LogRadioState=true
**.LogQuerryLgth=true
**.LogBLEConnection=true

## BLEV2_testnwk.ned -> Host_BLEV2testDevice.ned -> IBaseNetwLayer.ned

**.netwl.stats = false
**.tranl.stats = false

## BLEV2_testnwk.ned -> Host_BLEV2testDevice.ned -> IBaseApplLayer.ned

**.appl.stats = false


#################################################################
#################### Test/debug stuff ###########################
#################################################################
## Modules -> BLEMacV2.ned

**.TST_NoRandomAdvInt=false
**.TST_ForceHop=-1          # if >=0 will be used
**.coreDebug = false
**.debug = false


#################################################################
#################### Devices ####################################
#################################################################
## BLEV2_testnwk.ned

**.NumMasters = ${1}
**.NumSlaves = ${1}
```

Figura 36 – Cuarta parte del código del fichero de inicialización

```
## Modules -> BLEMacV2.ned

**master[*].nic.mac.Init_DataQueryLgth = 0 byte # initial nuber of bytes in the LL TX data buffer
**slave[*].nic.mac.Init_DataQueryLgth = 0 byte  # initial nuber of bytes in the LL data buffer
**.mac.advertisingAddr = -1                     # -1 - advertising address equals to index
**.mac.transmitWindowOffset = 0
**.mac.transmitWindowSize = 1
**.mac.connAccessAddress = 777

## BLEV2_testnwk.ned -> BLETestNetworkStatistics.ned

**NwkStats[*].StopSimulationOnNoData = false #stop simulation once the LL TX data buffer of all nodes is empty

## Modules -> BLE_BasicNwk.ned

**master[*].netwl.NWK_DEBUG_ROLE = 2 # initiator/master
**slave[*].netwl.NWK_DEBUG_ROLE = 1  # advertiser/slave

**master[*].netwl.connDataChannelMap_0to7 = 0x03
**master[*].netwl.connDataChannelMap_8to15 = 0x00
**master[*].netwl.connDataChannelMap_16to23 = 0x00
**master[*].netwl.connDataChannelMap_24to31 = 0x00
**master[*].netwl.connDataChannelMap_32to39 = 0x00

**netwl.connAdvertiseChannelMap_0to7 = 0x00
**netwl.connAdvertiseChannelMap_8to15 = 0x00
**netwl.connAdvertiseChannelMap_16to23 = 0x00
**netwl.connAdvertiseChannelMap_24to31 = 0x00
**netwl.connAdvertiseChannelMap_32to39 = 0xE0 # default 0xE0

**netwl.advInterval = 160                       # in 0.626 ms units
**netwl.scanInterval = 160                      # in 0.626 ms units, Range: 0x0004 â€“ 0x4000, see p.1261 of Spec v4.1
**netwl.scanWindow = 160                        # in 0.626 ms units, Range: 0x0004 â€“ 0x4000, see p.1261 of Spec v4.1
**netwl.connInterval = ${connIntervalVal=800}   # in 1.25 ms units
**netwl.connLatency = 0                         # SHOULD BE 0
**netwl.supervision_Timeout = ${connIntervalVal} # in 10 ms units, 8*connInterval

**netwl.nodeStartupDelay = 0 s

**master[*].netwl.initialData = 0 byte
**slave[*].netwl.initialData = 40 byte
**netwl.useSuggestedFastIntervalSturtupMechanism = true # activate startup with fast intervals
```

Figura 37 – Quinta parte del código del fichero de inicialización

En las figuras 33, 34, 35, 36 y 37 vemos el código del fichero de inicialización. Como ya he dicho antes, este fichero lo incluyen todos los proyectos simulados en Omnet ++ y consiste en asignar un valor a los parámetros pertinentes siempre que no se desee que reciban su valor por defecto. A la hora de compilar el proyecto, es este el archivo que hay que ejecutar dentro del simulador. Como se puede ver en las cinco imágenes del código, este se ha dividido en secciones con parámetros comunes. En la mayor parte de los casos, estos grupos de parámetros se encuentran dentro del mismo elemento de la red o del mismo módulo, pero pueden darse excepciones. En este último apartado del capítulo del módulo BLE comentaré dicho código antes de pasar al capítulo de resultados, donde este fichero de inicialización tiene una grandísima importancia.

En el primer grupo de parámetros posterior a indicar la red que se está simulando y el nombre que se le va a dar al experimento, podemos ver los parámetros del display, ubicados en la "Base Network" dentro de la red global. Estos simplemente indican el tamaño que debe poseer en los ejes X, Y y Z, la ventana que muestra la topología de la red durante la simulación. Los valores asignados son de 1000 m cada uno para que no falte espacio en ningún caso.

El segundo grupo se centra en los parámetros del canal, que están ubicados en el "Connection Manager". "sendDirect" indica que se van a crear puertas de entrada o salida diferentes para cada una de las conexiones. Los demás parámetros tan solo indican el máximo de potencia enviada usada por la red (10 mW), el mínimo límite de atenuación que tendrá la señal (-90 dBm), el mínimo coeficiente de pérdidas (2.0) y la mínima frecuencia de carga del canal (2.45e + 9 Hz).

El tercer grupo de parámetros se encuentra, de menor a mayor nivel, en el "SimpleBattery", dentro del "SimpleBatteryCurrentChart", dentro de la red de un dispositivo ubicado en la red global. Básicamente se encuentra en el módulo de batería que posee un dispositivo. Podemos observar que tanto la capacidad nominal como la real de la batería es de 2000 mAh, el tiempo de resolución es de 30 segundos, la delta de publicidad tiene el valor 0, el tiempo de publicidad es también de 30 segundos y el voltaje nominal de la batería es de 3 V.

El cuarto grupo de parámetros incluye los de la capa física, y se encuentra dentro de los módulos de la librería de MiXiM, más concretamente en el "IWirelessPhy", dentro del módulo NicBLEV2. Viendo estos parámetros podemos decir que el retraso en la transmisión no debería de ser simulado, la fuerza del ruido térmico de la señal es de -100 dBm, dicho ruido térmico sí debería de ser considerado, la radio está inicialmente en estado de reposo, la sensibilidad de la capa física es de -93 dBm, la máxima potencia de transmisión de la capa física será de 10 mW y habrá 40 canales de radio disponibles. Por último, se indica que los modelos análogos que se usarán se extraen del archivo "config.xml".

El quinto grupo de parámetros están relacionados con el consumo de corriente. Se encuentran también en el módulo NicBLEV2 de la librería de MiXiM, pero esta vez en el "WirelessNicBattery". En primer lugar, se indica el "phyType", que en nuestro caso será el "PhyLayerBattery". El resto de los parámetros son valores de intensidad asignados: La corriente en reposo es de 0.0235 mA, la corriente de recepción es de 22.1 mA, la corriente de decodificación es de 0 mA, la corriente de transmisión es de 27 mA, la corriente de la configuración de la recepción es de 11.05 mA, la de la transmisión es de 13.5 mA, la corriente de recepción-transmisión es de 13.5 mA y la opuesta es de 11.05 mA.

El sexto grupo de parámetros incluye todos los tiempos de conmutación necesarios para que la comunicación funcione como es debido. Estos se encuentran en el módulo BLEMacV2, dentro de la librería de módulos del espacio de trabajo del simulador. El tiempo que pasa desde el estado de reposo hasta el de transmisión es de 0.000004 segundos y también de 0.000004 segundos hasta el de recepción. El tiempo que pasa desde el estado de transmisión hasta el de recepción es de 0.000150 segundos, que es el mismo que el opuesto. El tiempo desde el estado de transmisión hasta el reposo es de 0 segundos, igual al tiempo que pasa desde el estado de recepción hasta el reposo.

El séptimo grupo de parámetros tan solo incluye uno y está relacionado con la movilidad. Este parámetro se encuentra, de menor a mayor nivel, en el "WirelessNode", que está dentro del "WirelessNodeBatteryCurrentChart", dentro de "Host_BLEV2", dentro del dispositivo ubicado en el mayor nivel de la red de esta comunicación. El parámetro indica el tipo de módulo de movilidad que se elige y, en este caso, es el de "StationaryMobility".

El octavo grupo de parámetros están relacionados con la red interna de los dispositivos y alguna de sus capas, incluyendo la de aplicación que es la más elevada. Estos parámetros son tres, uno de ellos se encuentra en la misma ubicación que el parámetro de movilidad del párrafo anterior, e indica el tipo de capa de red que se utiliza, que en este caso es una red básica BLE. Los otros dos se encuentran en las capas de aplicación y de red, dentro del dispositivo ubicado en la red global. Estos indican, respectivamente, el tamaño de la cabecera de la capa de red y de aplicación, ambos de 0 bits.

El noveno grupo de parámetros incluye algunas constantes importantes del módulo BLEMacV2, ubicado en la librería de módulos de MiXiM. La longitud de la cabecera de los datos es de 16 bits, la desviación IFS es de 0.0000002 segundos y la propia IFS es de 0.000150 segundos. La duración máxima de los canales de paquetes de anuncio es de 0.000376 segundos y la potencia de transmisión es de 1 mW. Por último, hay una variable relacionada con la política de beacons, que puede ser "ondata" o "always". En nuestro caso esta variable está en modo "ondata", que es sólo cuando llegue un dato.

El décimo grupo de parámetros abarca los datos de salida de la simulación. Estos se dividen a su vez en tres subgrupos. En primer lugar, nos encontramos con un conjunto de parámetros tipo boolean ubicados en el más alto nivel de los dispositivos dentro de la red global. Estos únicamente nos dicen si se utilizarán dichos elementos en la comunicación o no y, como podemos observar, los cinco elementos (Corriente, frecuencia, estado de radio, longitud de la cola y conexión BLE) están activos. En segundo y tercer lugar, nos encontramos con tres parámetros tipo boolean que mencionan si estarán activas las estadísticas de computación en ciertos elementos de la red. Dos de ellos se encuentran en la "IBaseNetwLayer" y el último en la "IBaseApplLayer" dentro de un dispositivo.

El undécimo grupo muestra una serie de parámetros sobre la depuración de la simulación. Estos están ubicados en el módulo BLEMacV2 y son cuatro. Se indica que se usarán componentes aleatorios de intervalos de publicidad y que no se forzará el salto entre ellos. Los otros dos parámetros están relacionados con la depuración, y están puestos sus valores por defecto, por lo que no cambia absolutamente nada.

El duodécimo y último grupo de parámetros sirve para configurar los dispositivos que participen en la comunicación BLE. Al ser tantos, no voy a pararme en cada uno de los subgrupos para indicar su ubicación en la red física, sino que los voy a analizar todos como un conjunto. De todas formas, hay que destacar que todos ellos están localizados en ciertos elementos de la red y en alguno de los tres módulos importantes ya antes mencionados, siendo el mayor número de los parámetros del módulo BLE_BasicNwk, el único que aún no había proporcionado variables de configuración.

En nuestro caso tendremos un dispositivo que actúa como maestro y otro que lo hace como esclavo, sin embargo, en el capítulo de resultados veremos cómo cambian las gráficas si añadiésemos más dispositivos esclavos. El número inicial de bytes del búfer de datos de ambos dispositivos es de 0 bytes, la dirección de publicidad es la misma que la del índice, "transmitWindowOffset" = 0, "transmitWindowSize" = 1 y "connAccessAddress = 777. Por otro lado, la simulación no terminará, aunque el búfer de datos de todos los nodos esté vacío. El dispositivo maestro tendrá el rol 2 (iniciador) y el dispositivo esclavo tendrá el rol 1 (esclavo). Después hay dos grandes grupos de parámetros relacionados con las conexiones de los canales de datos y los canales de publicidad, estando estos últimos asignados a toda la red y los primeros únicamente al dispositivo maestro. El intervalo de publicidad y escaneo y la ventana de escaneo son de 100.16 ms, el intervalo de conexión es de 800 * "connIntervalVal", la latencia de conexión es igual a cero y el tiempo de supervisión es igual al "connIntervalVal". El retraso de los nodos es de 0 segundos, el dispositivo maestro comienza con 0 bytes de datos dentro de la red, mientras que el dispositivo esclavo lo hace con 40 bytes. Y, por último, hay que mencionar que se activa la opción de intervalos rápidos.

Tras concluir este importante capítulo centrado en la explicación de cada uno de los elementos del módulo BLE, de sus librerías y del fichero de inicialización de la simulación, pasaremos en el siguiente capítulo a explicar y comentar los resultados obtenidos de esta simulación. Hay que destacar que se podrán realizar algunos cambios en ciertos parámetros ya antes comentados para obtener unos resultados más sólidos. De todas formas, todo esto ya lo comentaré en mayor profundidad en el siguiente apartado de resultados.

Capítulo 6. RESULTADOS

6.1 SIMULACIÓN

En este apartado del capítulo de resultados de la memoria, comenzaré a explicar como se compila y se simula un proyecto en nuestro simulador y comentaré los resultados visuales obtenidos durante la simulación para dos configuraciones distintas de los dispositivos.

6.1.1 COMPILACIÓN

En primer lugar, antes de poder ejecutar cualquier proyecto, como es obvio, es necesario instalar la versión que se desee de Omnet++ y del framework MiXiM. No me pararé a explicar como se realiza su instalación ya que es muy sencillo y no hay apenas pasos a seguir. Cuando tengamos ambos instalados, será necesario importar este espacio de trabajo dentro del simulador.

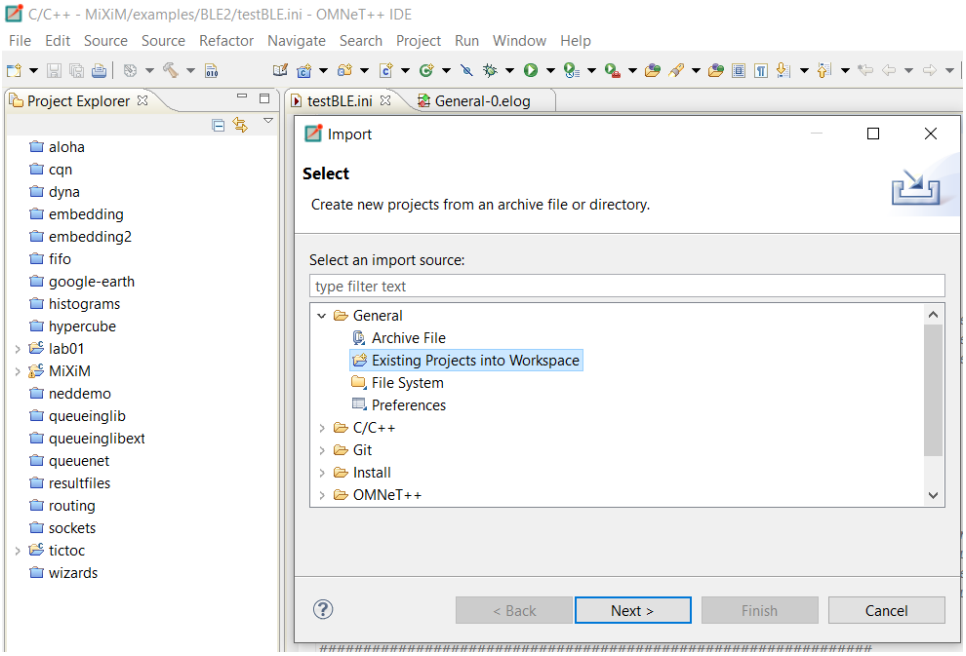


Figura 38 – Importación de MiXiM en el simulador

En la figura 38 vemos cómo se va a importar el framework dentro del simulador. Para ello, estando en el simulador, habrá que hacer click en: File -> Import… -> General -> Existing Proyects into Workspace. Esto será necesario para buscar e incorporar ciertos proyectos y frameworks al Workspace.

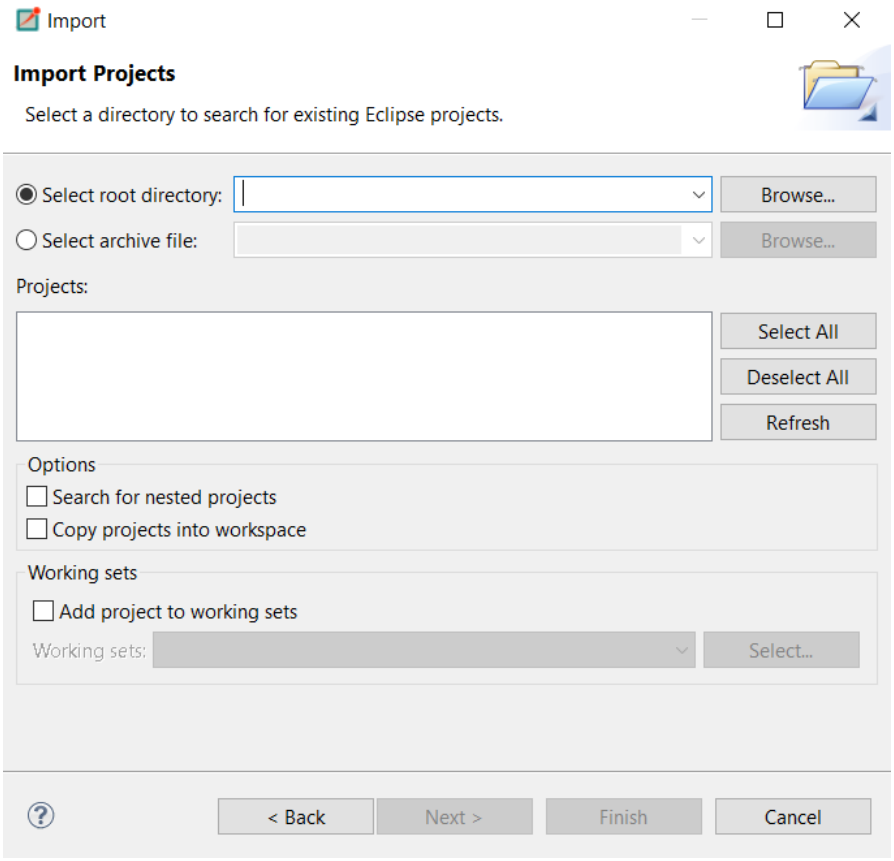


Figura 39 – Ventana de importación de proyectos existentes

En la figura 39 podemos ver la ventana de importación de proyectos en el Workspace. En ella, tan solo hay que indicar la ubicación del MiXiM dentro de la caja de texto "Browse…", además de marcar la casilla de "Copy proyects into workspace", para que así todos los proyectos que incluya el framework que importemos, se añadan también a los demás proyectos del simulador.

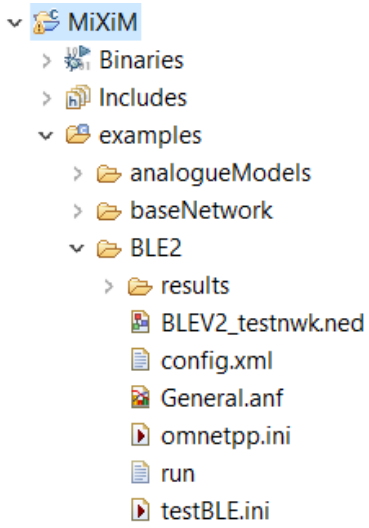


Figura 40 – Proyecto de BLE2 dentro de MiXiM

En la figura 40 vemos como el framework de MiXiM está correctamente importado en el simulador y como el proyecto de BLE2 esta también incorporado a dicho framework. Para incorporar nuestro proyecto de BLE a MiXiM, bastaba con copiar las carpetas pertinentes en sus respectivas ubicaciones dentro del framework: el proyecto global dentro de la carpeta de "examples" y los demás módulos en sus respectivas ubicaciones dentro de src -> modules.

Llegado a este punto ya tendremos todo listo para comenzar a trabajar en nuestro proyecto y realizar las simulaciones necesarias. Cuando ya tengamos nuestro proyecto terminado, necesitaremos generar un fichero de inicialización, con extensión ".ini", donde poder controlar todas las variables que participen en nuestro proyecto.

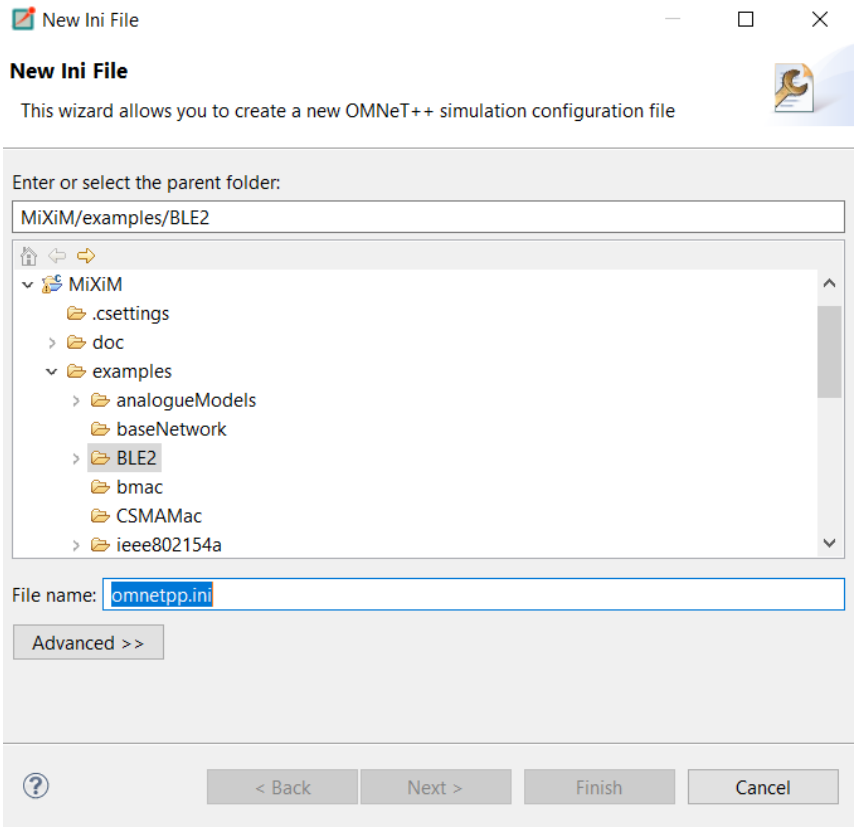


Figura 41 – Creación del fichero de inicialización

En la figura 41 vemos la ventana que nos saldrá al darle a crear un nuevo fichero de inicialización (Click derecho donde se encuentran todos los proyectos del simulador -> New -> Initialization File (ini)). Basta con que nos ubiquemos en el proyecto al que deseemos añadir dicho fichero y ponerle el nombre que deseemos.

Después de esto, ya tendremos creado nuestro fichero de inicialización donde poder controlar todas las variables de nuestro proyecto. Además, será este fichero el que se ejecutará para poder simular nuestro proyecto. A continuación, veremos como se lleva a cabo este proceso y mostraremos los resultados oportunos.

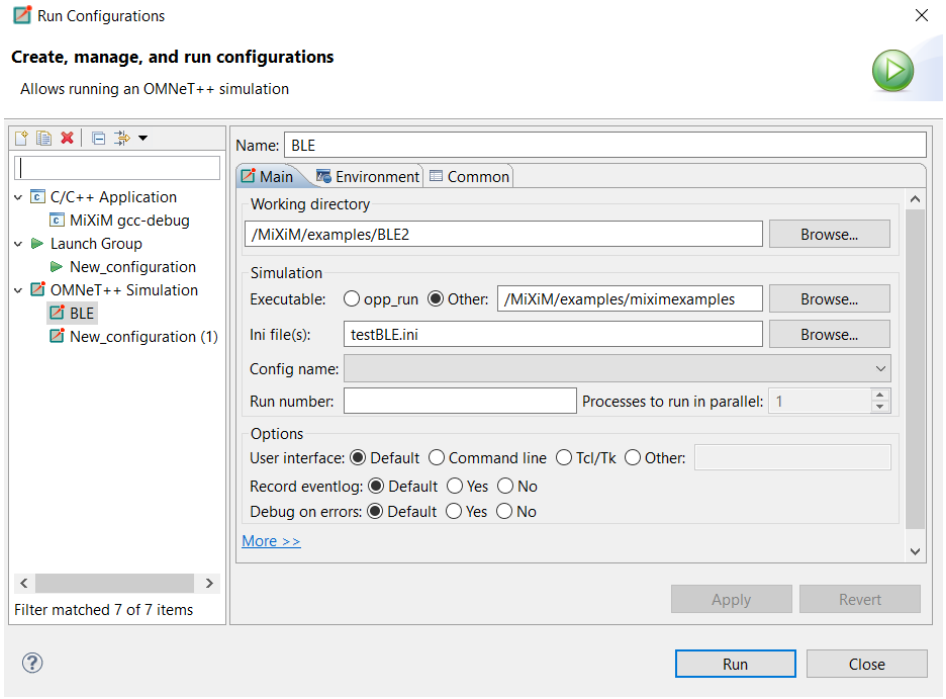


Figura 42 – Ventana de elección de simulación

En la figura 42 podemos ver la ventana que se muestra antes de simular un proyecto (Click derecho donde se encuentran todos los proyectos del simulador -> Run as -> Run Configurations…) En esta ventana comenzaremos por asignarle un nombre a la simulación y así poder volver a seleccionarla más adelante. En el "Working directory" añadiremos la ubicación del proyecto que estamos simulando, en la siguiente caja de texto añadiremos la ubicación por defecto de los ejemplos de nuestro framwork, MiXiM en este caso, y en la siguiente tan solo tendremos que escribir el nombre del fichero de inicialización que queramos ejecutar. Lo siguiente sería pulsar el botón de "Apply" y de "Run". Si todo ha salido correctamente, lo siguiente que veremos será la ventana de simulación del Omnet++, que ya comentaré en los apartados siguientes.

6.1.2 MAESTRO-ESCLAVO

En este apartado mostraré la simulación de una comunicación BLE en donde intervienen dos dispositivos: uno maestro y otro esclavo. Pero antes, comentaré algunas utilidades a tener en cuenta de esta herramienta de simulación.

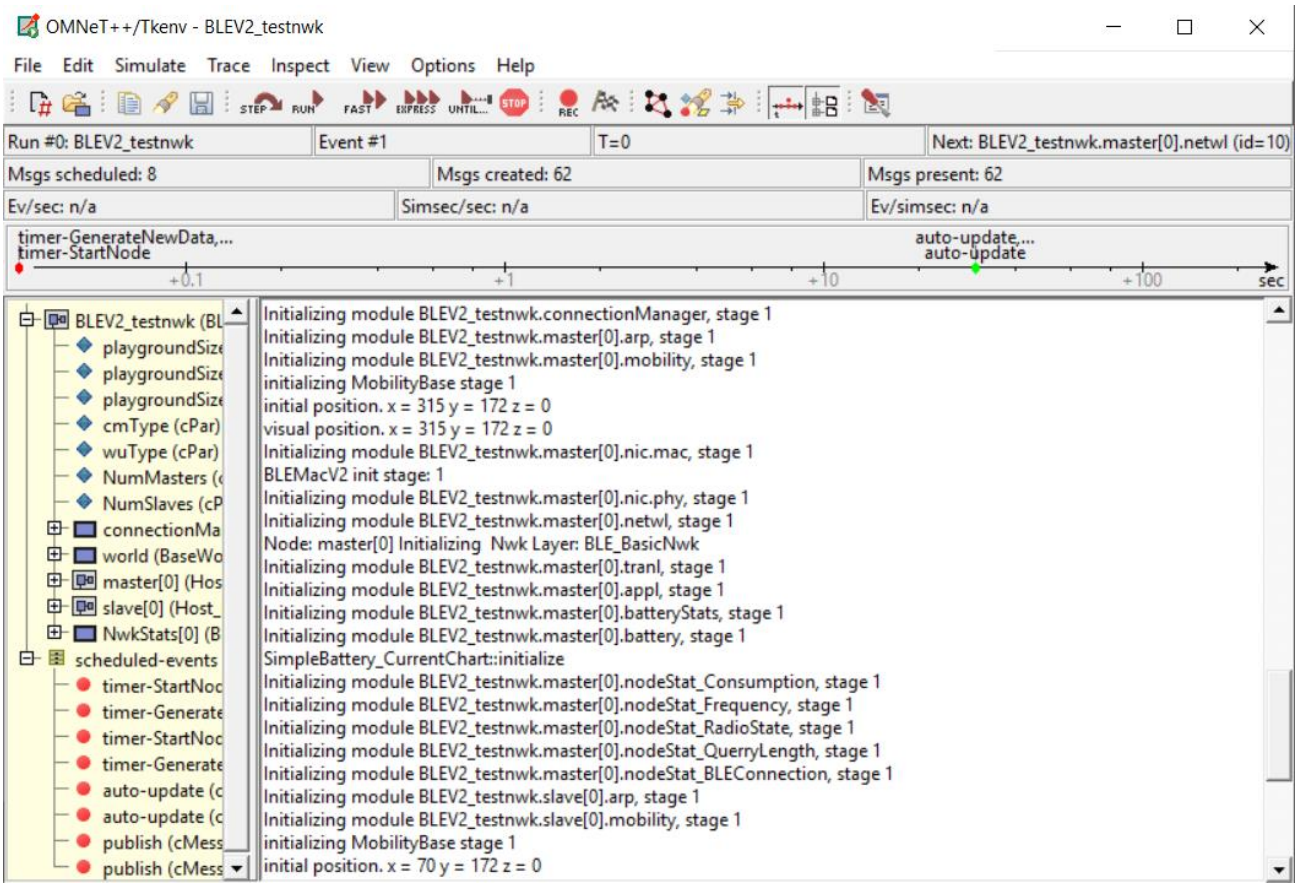


Figura 43 – Ventana de simulación del Omnet++

En la figura 43 vemos que en la parte superior de la ventana nos encontramos con el botón de REC, que sirve para grabar la simulación y así poder visualizar más adelante desde el simulador alguna de las gráficas que veremos en apartados posteriores. Es importante hacer siempre click en este botón antes de comenzar la simulación para obtener mejores resultados.

Por otro lado, nos encontramos con una serie de botones que controlan los tiempos reales de la simulación. Me refiero al botón de comenzar la simulación (RUN), al de pararla (STOP) y al de simular más rápido (FAST). Además de un último botón llamado "UNTIL…", que puede venir muy bien si lo que queremos es simular nuestro proyecto hasta que llegue un evento en concreto o hasta que llegue un preciso instante de tiempo. En mi caso, los botones que pulso son el de REC, el de RUN y el de STOP, en ese orden y dejando un cierto tiempo entre los dos últimos, para poder analizar la mayor cantidad de resultados posibles.

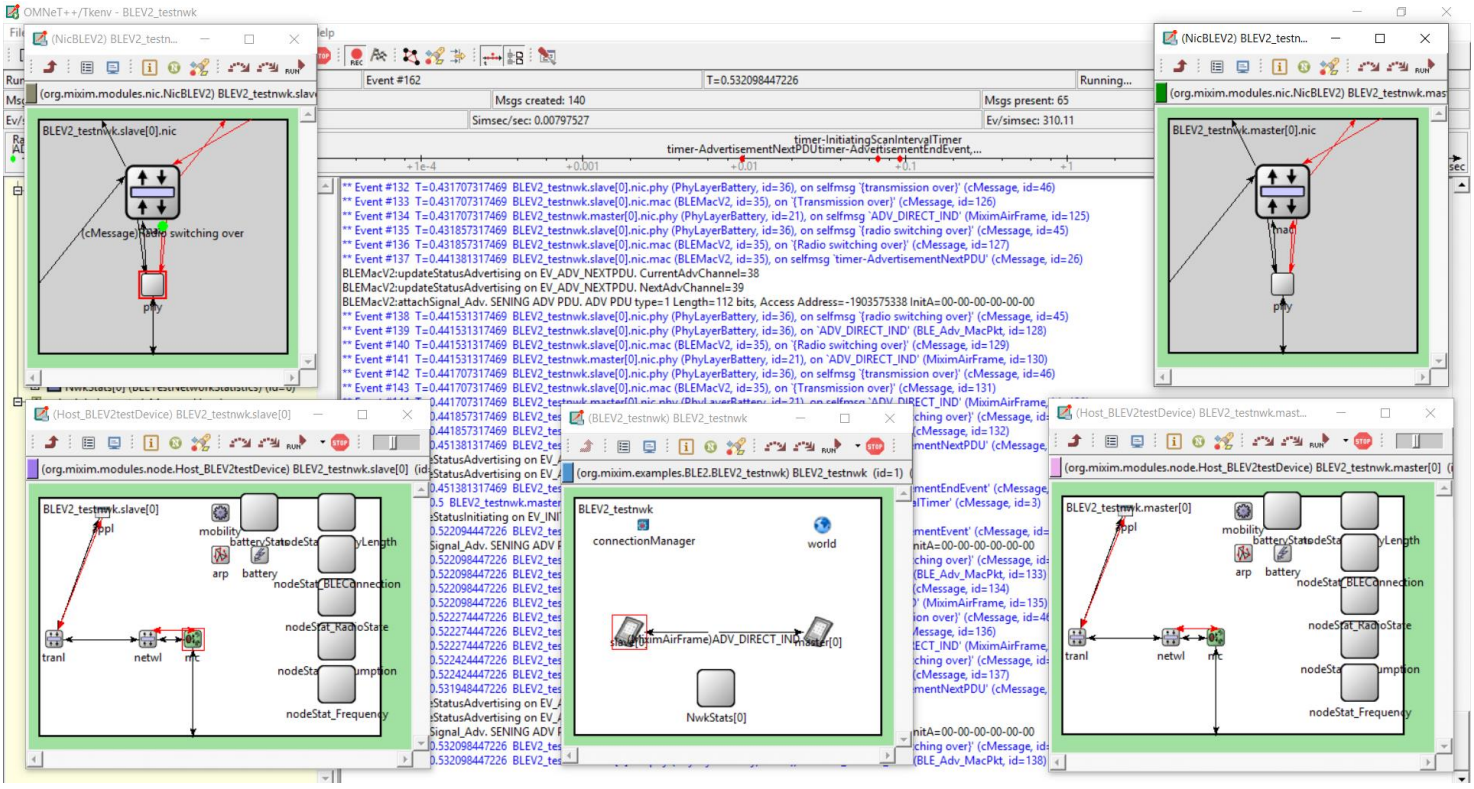


Figura 44 – Simulación de una comunicación Maestro-Esclavo

En la figura 44 se pueden ver varios elementos importantes de la red que participan enteramente en la comunicación BLE mediante el envío y recepción de mensajes. En el centro nos encontramos con el "BLEV2_testnwk", que es el más alto nivel de nuestra red, y en donde podemos diferenciar dos dispositivos, el maestro y el esclavo. Abajo a la derecha podemos ver el "Host_BLEV2testDevice", más concretamente el "BLEV2_testnwk.master[0]", es decir, el dispositivo maestro, con todos sus elementos. Del mismo modo, abajo a la izquierda se ubica el "BLEV2_testnwk.slave[0], también con todos sus elementos, que son los mismos que los del maestro. Arriba a la derecha está el "NicBLEV2", o "BLEV2_testnwk.master[0].nic. Este es el módulo NIC o la interfaz de la tarjeta de red del dispositivo maestro. Por otro lado, tenemos arriba a la izquierda este mismo módulo, pero para el dispositivo esclavo ("BLEV2_testnwk.slave[0].nic").

En el momento de comenzar a simular, estos elementos mencionados se comunican entre sí mediante el envío y recepción de ciertos mensajes que ya comentaremos en los siguientes apartados. En el momento exacto de la captura, podemos ver un mensaje llamado "(MiximAirFrame)ADV_DIRECT_IND" que está siendo enviado desde el módulo PHY de la tarjeta de red del dispositivo esclavo al mismo módulo de la tarjeta de red del dispositivo maestro.

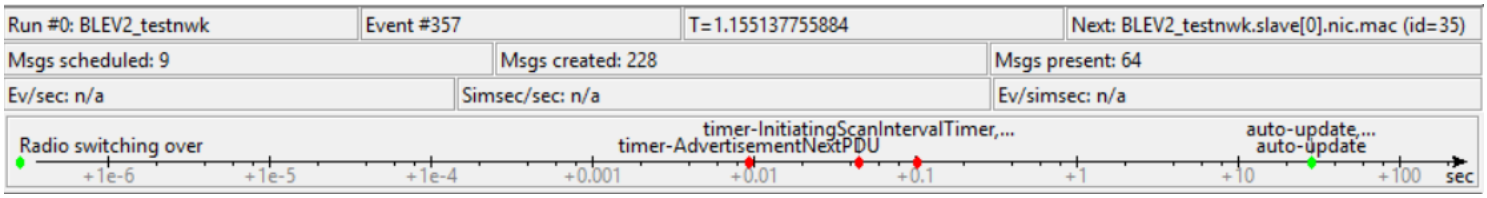


Figura 45 – Algunos datos de la simulación

En la figura 45 podemos ver algunos datos de la simulación como el tiempo que lleva activa, el número de eventos que se han generado, los eventos por segundo o los mensajes creados, así como el id y el nombre del siguiente elemento que participará en la comunicación.

6.1.3 MAESTRO-ESCLAVO-ESCLAVO

En este apartado mostraremos lo mismo que en el anterior, solo que esta vez habrá dos dispositivos esclavos y un dispositivo maestro con el que se comunican. El esquema es muy similar, salvo que ahora tenemos un dispositivo esclavo más, junto con su propia tarjeta de red también mostrada.

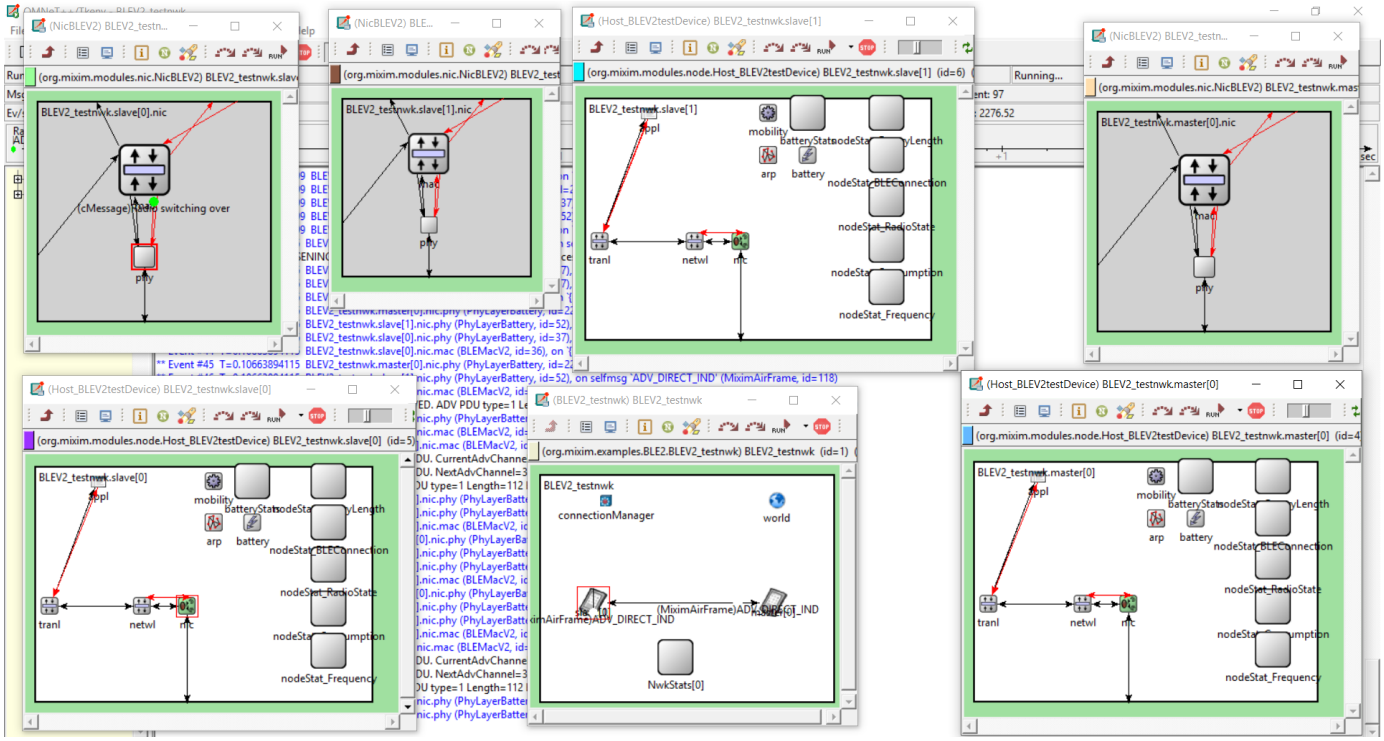


Figura 46 – Simulación de una comunicación Maestro-Esclavo-Esclavo

En la figura 46, podemos ver la simulación de esta comunicación ya antes comentada. Como ya se ha mencionado, la única diferencia es la presencia de un segundo dispositivo esclavo y de su tarjeta de red mostrada en la figura. En el caso anterior, el de la comunicación Maestro-Esclavo, los módulos PHY y MAC de la tarjeta de red del dispositivo esclavo se intercambian mensajes entre sí, hasta que le envían un mensaje al dispositivo maestro.

Por otro lado, en esta nueva simulación comienzan los módulos de uno de los dispositivos esclavos a intercambiarse paquetes entre ellos, hasta que le envía un mensaje tanto al otro dispositivo esclavo, como al dispositivo maestro. Acto seguido, son los módulos del otro dispositivo esclavo los que se intercambian mensajes entre sí, hasta enviar otro mensaje al primer dispositivo esclavo y al dispositivo maestro. Esto sucede indefinidamente hasta que el programador decida detener la simulación.

De todas formas, estas dos simulaciones se mostrarán en vivo en la presentación de este proyecto, para verlas en movimiento y entender mejor la comunicación. Además, en los apartados siguientes mostraremos una serie de gráficas y trazas obtenidas de esta simulación realizada anteriormente, por lo que será mucho más sencillo entender este proceso de comunicación entre un dispositivo maestro y uno o varios dispositivos esclavos.

6.2 COMUNICACIÓN

En este importante apartado, se van a explicar una serie de gráficas que se obtienen de la simulación de la propia comunicación BLE. Para que los resultados sean más sólidos, terminaremos por cambiar algunos parámetros importantes y observaremos las diferencias obtenidas en las gráficas y en las trazas.

6.2.1 ELOG

En este primer apartado de gráficas, voy a mostrar gráficamente cómo se comporta la comunicación. El archivo ".elog" se genera cuando pulsamos el botón de REC antes de comenzar la simulación y muestra que ha sucedido en ella. Para obtener resultados más variados, cambiaré algunos parámetros como el número de dispositivos maestros o esclavos y algunos intervalos. Cabe recalcar que, en todo momento, realizaré la simulación para que la comunicación persista durante tan solo 1 segundo, tiempo más que suficiente para poder contemplar lo que sucede.

Antes de comenzar a mostrar resultados, es importante comprender como se comportan los tres intervalos principales de la comunicación. El intervalo de publicidad o "advertising Interval" es el tiempo en el que los dispositivos podrán enviar paquetes de publicidad en cada uno de los canales de la comunicación. En el caso de BLE, estos canales serán el 37, el 38 y el 39 y lo ideal es que dicho intervalo abarque estos tres canales. Por otro lado, el intervalo de escaneo y la ventana de escaneo serán iguales para que se comunique el 100% del tiempo. Este intervalo (lo mencionaremos como uno sólo ya que tienen el mismo valor) debería de ser menor o igual que el intervalo de publicidad para que no se solapen y mayor de lo que tarda el dispositivo esclavo en comunicarse con el maestro por uno de los tres canales para que no se interrumpa dicho proceso.

En primer lugar, mostraré algunos fragmentos de la comunicación cuando los parámetros comentados anteriormente tienen los siguientes valores:

- Número de maestros: 1
- Número de esclavos: 1
- Intervalo de publicidad: 100 ms
- Intervalo de escaneo: 100 ms
- Ventana de escaneo: 100 ms

Antes de comenzar, mencionaré que al ser este el primer caso mostrado, incorporaré un par de figuras más para que se vea más de cerca lo que sucede. En el resto de los casos, tan solo mostraré una captura alejada de lo que está sucediendo en la comunicación para que se aprecie todo en una sola imagen.

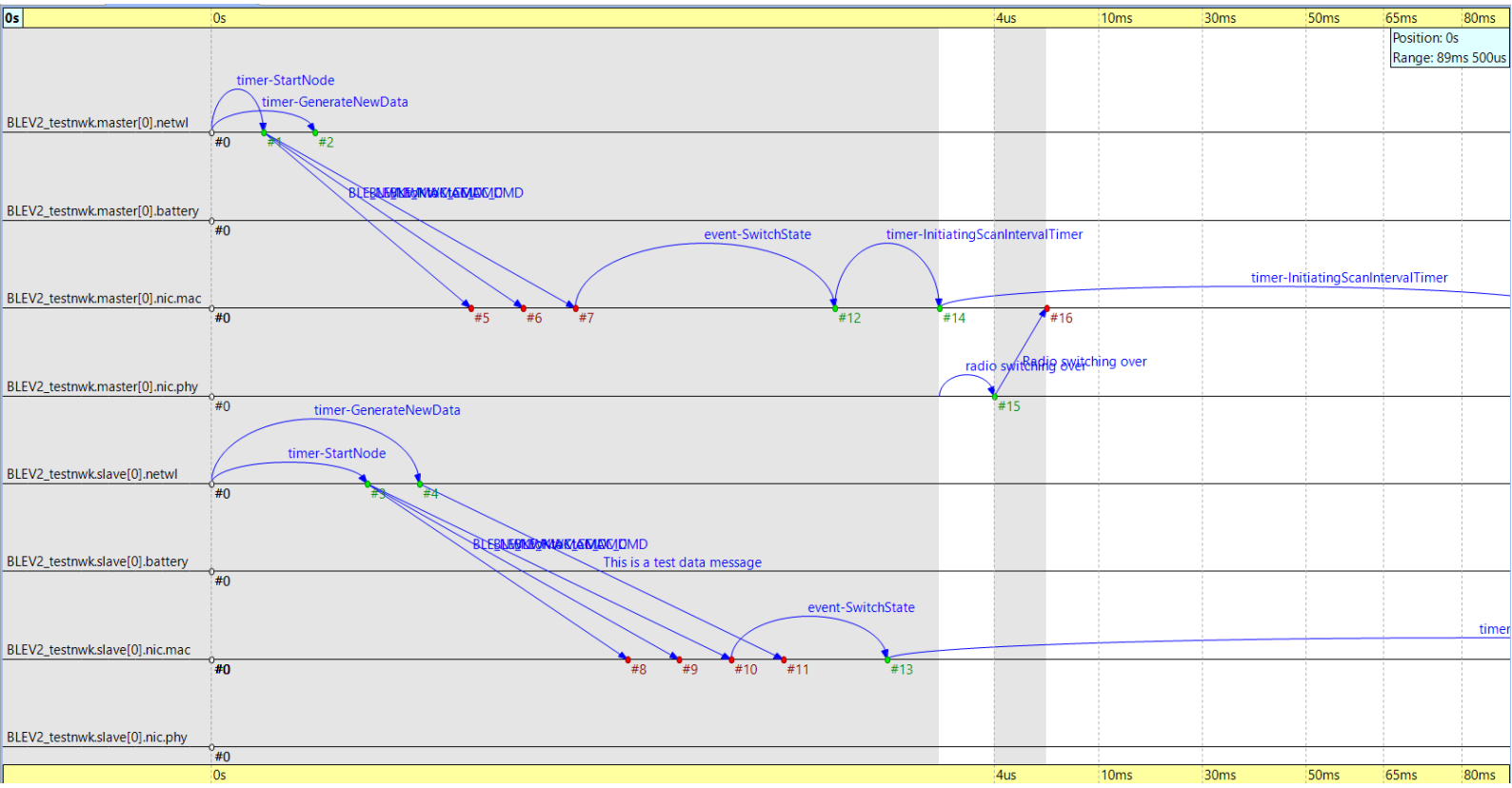


Figura 47 – Inicialización de los dispositivos

En la figura 47 nos encotramos con la inicialización del dispositivo maestro y del esclavo. Cada uno de ellos tiene cuatro módulos importantes que son los que realizan las tareas más importantes en la comunicación: El módulo de red (netwl), el módulo de batería (battery), el módulo MAC (nic.mac) y el módulo físico (nic.phy). El primer paso es la inicialización del módulo de red con los mensajes "timer-StartNode" y "timer-GenerateNewData", de ahí se envían una serie de paquetes con datos del dispositivo al módulo mac. Estos paquetes se llaman "BLE_NWKtoMAC_CMD" y un cuarto mensaje de texto que solo se envía en el dispositivo esclavo. Dentro de ese mismo módulo, y en ambos dispositivos, se produce un cambio de estado ("event-SwitchState") previo a la inicialización del intervalo de escaneo (dispositivo maestro) y del intervalo de publicidad (dispositivo esclavo). El "Radio switching over" tan solo indica que ha habido un cambio de estado en la radio del dispositivo.

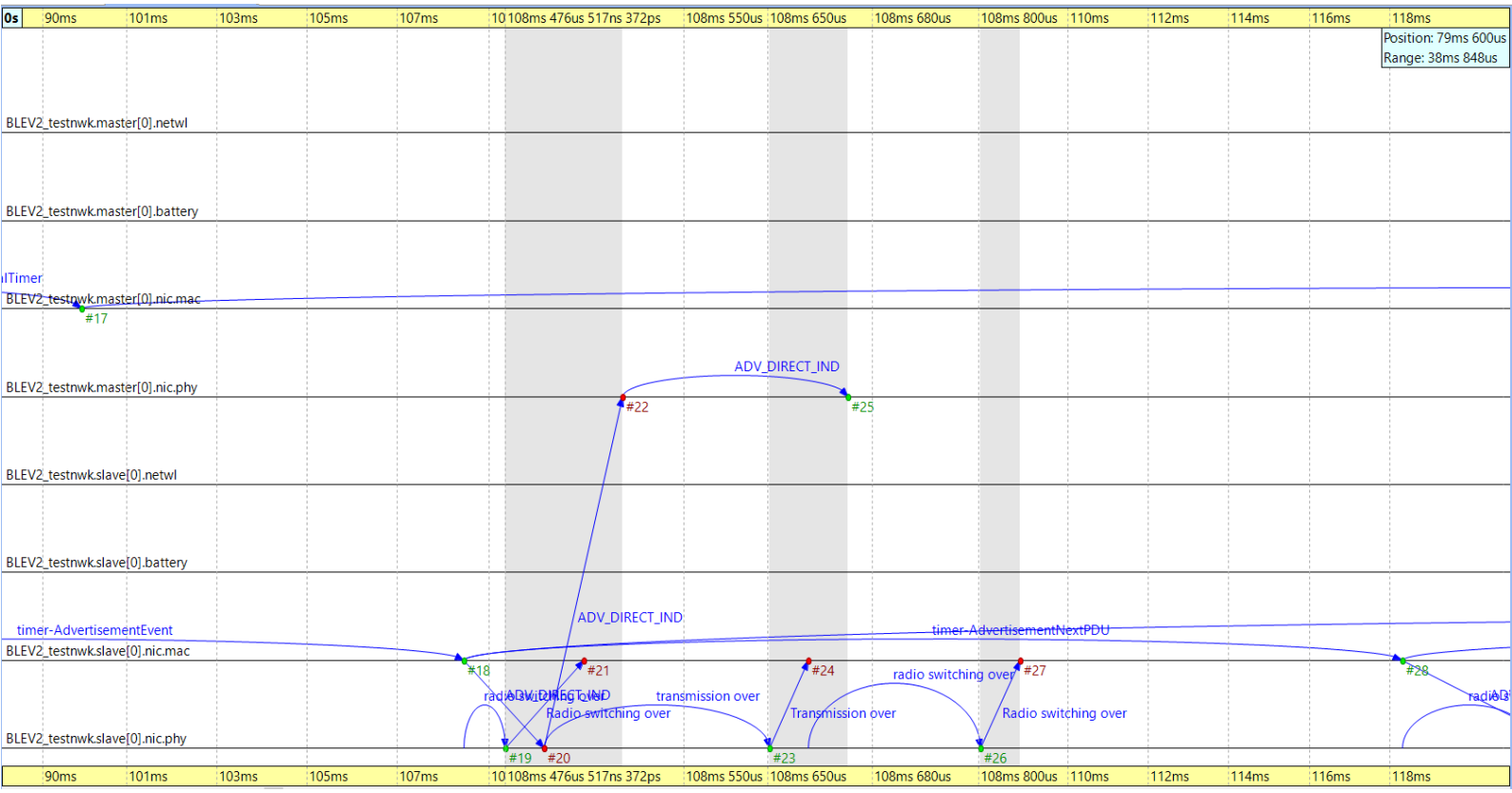


Figura 48 – Comunicación entre el dispositivo esclavo y el maestro en uno de los canales de publicidad

En la figura 48 vemos como se comunican el dispositivo esclavo y el maestro en el primero de los canales. Se puede observar que al llegar el mensaje “timer-AdvertisementEvent”, la radio cambia de estado para comenzar a retransmitir por el canal 37. El mensaje “ADV_DIRECT_IND” que primero pasa del módulo MAC al módulo físico del dispositivo esclavo y luego se envía al módulo físico del dispositivo maestro, es la solicitud de conexión por parte del esclavo hacia el maestro. Al terminar esta solicitud se vuelve a realizar un cambio de estado en la transmisión (“transmission over”), seguido de un cambio en el estado de la radio. Después de esto, llega le mensaje “timer-AdvertisementNextPDU” que da por terminada la transmisión en este canal para poder seguir transmitiendo en el siguiente, el 38 en este caso.

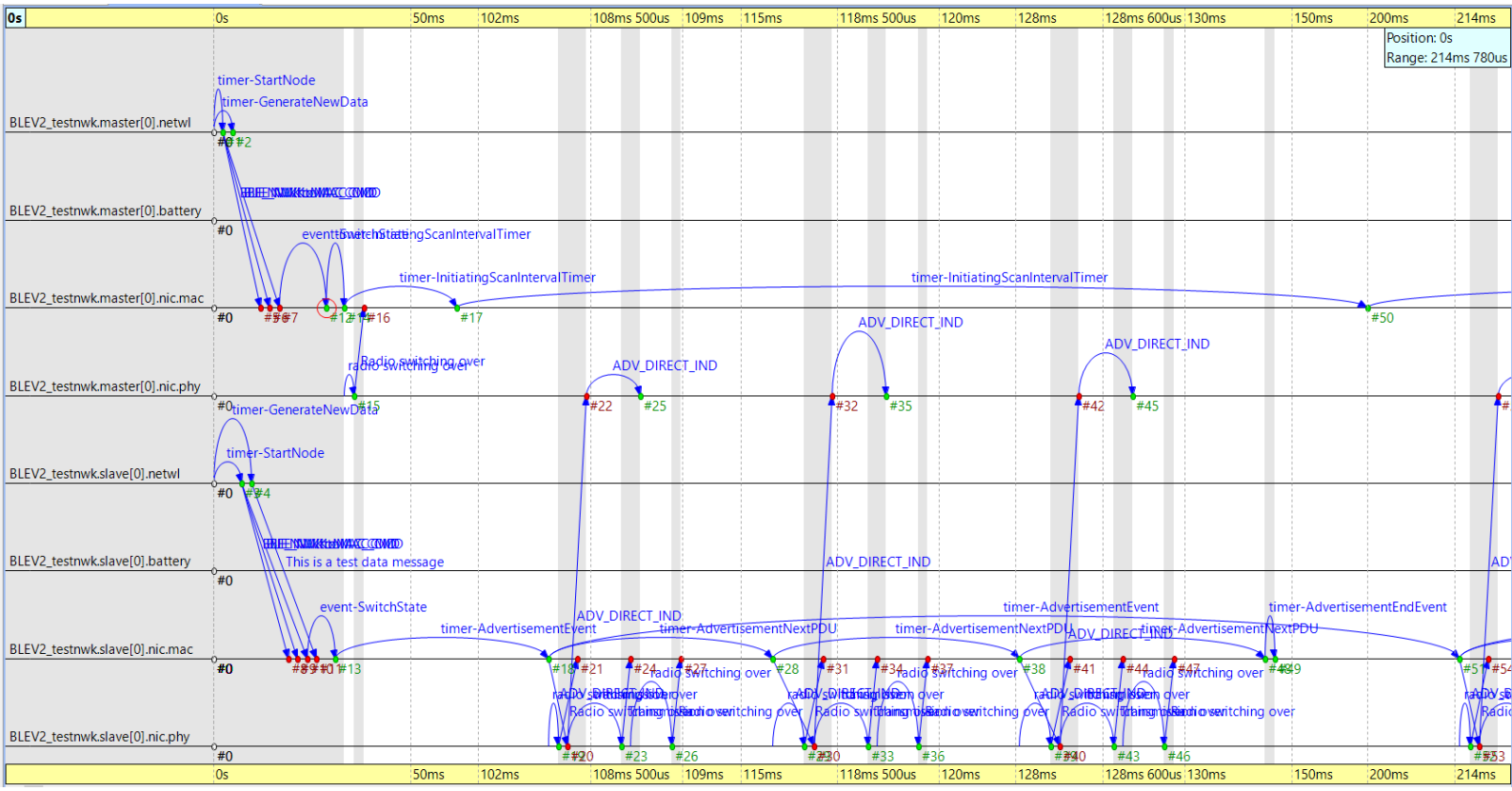


Figura 49 – Comunicación entre un esclavo y un maestro con todos los intervalos iguales a 100 ms

En la figura 49 vemos la comunicación entre un dispositivo maestro y un esclavo cuando el intervalo de publicidad y el de escaneo son iguales a 100 ms. Podemos ver como, tras inicializarse ambos dispositivos, comienzan a comunicarse en cada uno de los tres canales. Este proceso se repite indefinidamente hasta que se decida terminar la simulación. El tiempo que tarda el dispositivo esclavo en comunicarse con el maestro nos lo muestra el recorrido del mensaje "timer-AdvertisementNextPDU", que dura aproximadamente unos 10 ms. Por lo tanto, se retransmitirá por los tres canales (37, 38 y 39) en unos 40 ms, si tenemos en cuenta los tiempos que pasan antes y despues de dicho proceso. Nuestros intervalo de publicidad y de escano son de 100 ms, por lo que no hay problemas para retransmitirse los tres canales dentro de un mismo intervalo. Por último, hay que destacar que cuando se termina de transmitir por los tres canales se recibe un mensaje en el módulo MAC del escalvo inciando el fin del evento ("timer-AdvertisementEndEvent").

En segundo lugar, mostraré otro caso en donde aumentamos el número de dispositivos esclavos y mantenemos constantes los intervalos:

- Número de maestros: 1
- Número de esclavos: 2
- Intervalo de publicidad: 100 ms
- Intervalo de escaneo: 100 ms

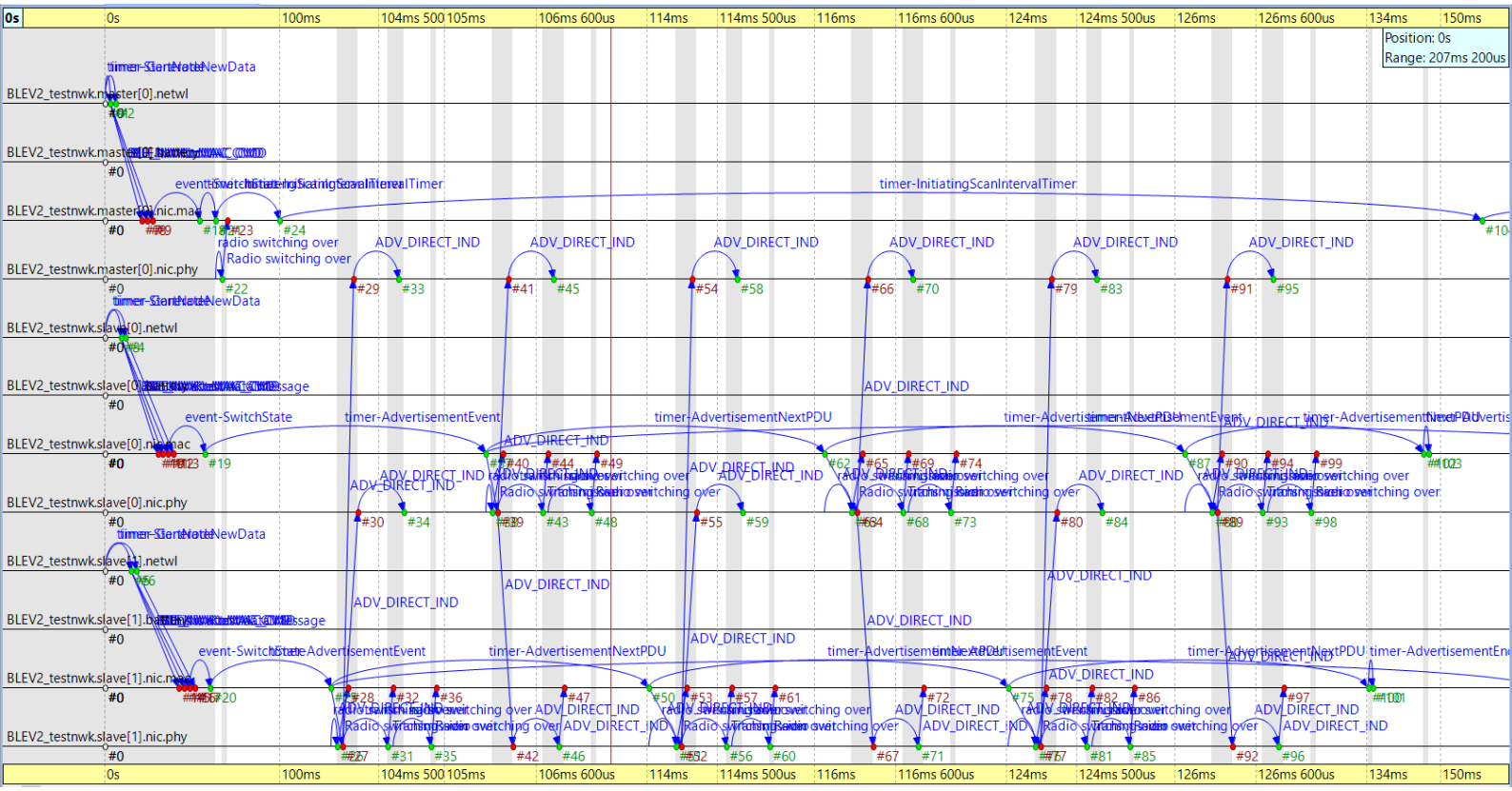


Figura 50 – Comunicación entre dos esclavos y un maestro con todos los intervalos iguales a 100 ms

En la figura 50 podemos ver cómo se comunican dos dispositivos esclavos con uno maestro. En primer lugar, se inicializan los tres dispositivos de la misma manera que comentamos anteriormente. Ahora, al haber el doble de dispositivos esclavos, se tardará el doble de tiempo en comunicarse por los tres canales. Estos dos dispositivos se van turnando vara comunicarse con el dispositivo maestro. Además, mientras uno de los esclavos está transmitiendo información por un canal, no solo se comunica con el dispositivo maestro,

sino que también lo hace con el otro esclavo. De todas formas, el proceso es muy similar, al comentado anteriormente con un dispositivo maestro y otro esclavo, solo que tarda más tiempo en transmitirse la información. Hay que destacar también, que en este caso el dispositivo maestro consume ligeramente una mayor cantidad de energía que antes y cada uno de los esclavos consume algo menos, ya que se reparten la carga de trabajo.

En tercer lugar, vamos a ver como afecta la comunicaión el bajar el intervalo de publicidad al mínmo posible sin que se ocasionen errores. Este intervalo debe ser el mínimo posible que abarque la transmisión por los tres canales:

- Número de maestros: 1
- Número de esclavos: 1
- Intervalo de pubicidad: 20 msaefa
- Intervalo de escaneo: 100 ms

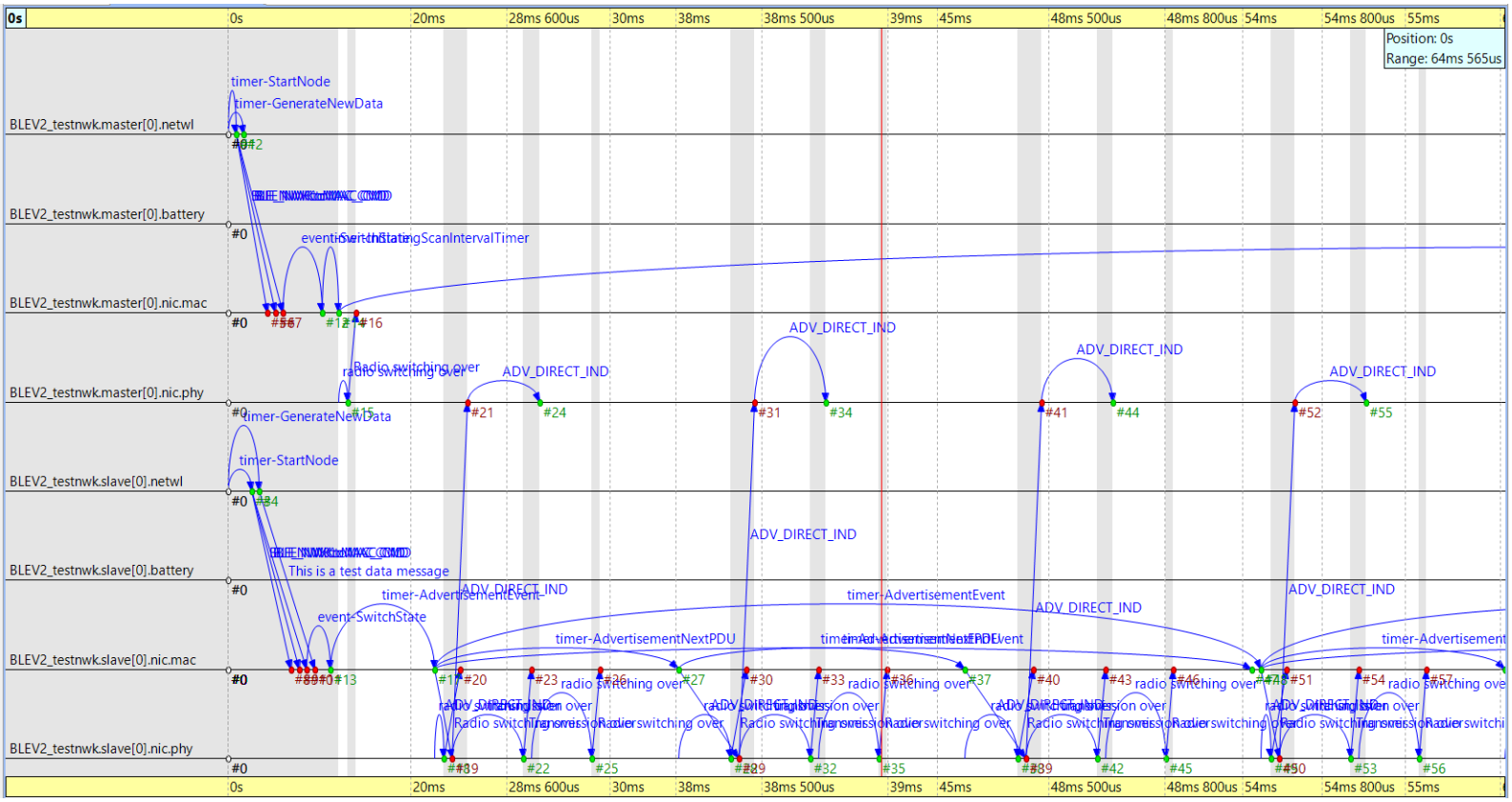


Figura 51 – Comunicación entre dos dispositivos con el intervalo de publicidad a 20 ms

En la figura 51 podemos ver esta misma comunicación entre un esclavo y un maestro cuando el intervalo de publicidad se ha reducido al mínimo posible (20 ms). Cada evento de publicidad sucede en un canal distinto y dura una pequeña cantidad de milisegundos. Al haber reducido tanto este intervalo, ahora el intervalo de scaneo, que sigue siendo de 100 ms, podrá abarcar una mayor cantidad de eventos de publicidad.

Por último, mostraré el caso contrario que sucedería al mantener constante a 100 ms el intervalo de publicidad y reducir el intervalo de escaneo:

- Número de maestros: 1
- Número de esclavos: 1
- Intervalo de publicidad: 100 ms
- Intervalo de escaneo: 10 ms

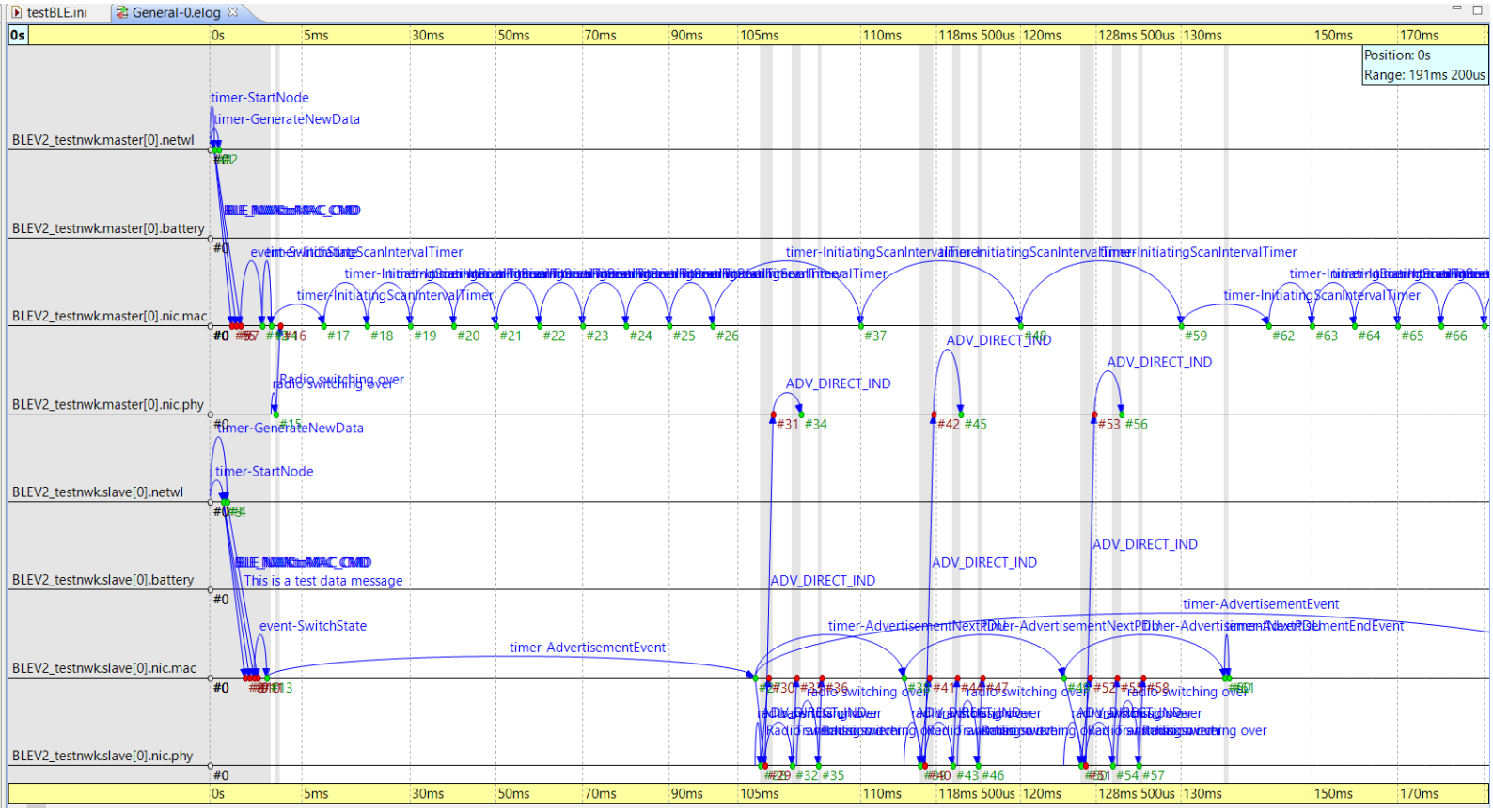


Figura 52 – Comunicación entre dos dispositivos con el intervalo de escaneo a 10 ms

En la figura 52 vemos la comunicación entre un esclavo y un maestro cuando el intervalo de escaneo se reduce a 10 ms y el de publicidad se mantiene constante a 100 ms. En este caso, el intervalo de publicidad no tiene problemas en abarcar los tres eventos de publicidad transmitiendo en cada uno de los tres canales. Por otro lado, al ser tan pequeño el intervalo de escaneo, concretamente la décima parte del de publicidad, se repite 10 veces por cada una del "advertising interval". De todas formas, sigue siendo ligeramente mayor de lo que dura un evento de publicidad, por lo que cada "scan interval" puede abarcar sin problemas cada uno de estos eventos. En cuanto al consumo que ya veremos en apartados siguientes, el aumento o la disminución de este intervalo prácticamente no afecta al consumo ni del dispositivo maestro ni del esclavo.

6.2.2 TRAZA

En este subapartado se va a comentar la traza que se obtiene de la simulación. Este fichero muestra lo mismo que se ve en las gráficas .elog anteriores, pero en texto. La traza completa se mostrará en el Anexo del final de la memoria, aquí se comentará un pequeño fragmento de la misma. La configuración que se ha utilizado es:

- 1 dispositivo Maestro
- 1 dispositivo Esclavo
- Tadv, TscanWindow, TscanInterval = 100 ms.

Antes de comenzar, se mostrará una tabla con todos los eventos de la simulación hasta llegar a unos 0.3 segundos, tiempo más que suficiente para observar y comprender lo sucedido en más detalle que con las gráficas del apartado anterior.

EVENTO	TIEMPO (seg)	FROM	TO	MENSAJE
#1	0	BLEV2_testnwk	Master.NETWL	Timer-StartNode
#2	0	BLEV2_testnwk	Master.NETWL	Timer-GenerateNewData
#3	0	BLEV2_testnwk	Slave.NETWL	Timer-StartNode
#4	0	BLEV2_testnwk	Slave.NETWL	Timer-GenerateNewData
#5	0	Master.NETWL	Master.NIC.MAC	BLE_NWKtoMAC_CMD
#6	0	Master.NETWL	Master.NIC.MAC	BLE_NWKtoMAC_CMD
#7	0	Master.NETWL	Master.NIC.MAC	BLE_NWKtoMAC_CMD
#8	0	Slave.NETWL	Slave.NIC.MAC	BLE_NWKtoMAC_CMD
#9	0	Slave.NETWL	Slave.NIC.MAC	BLE_NWKtoMAC_CMD
#10	0	Slave.NETWL	Slave.NIC.MAC	BLE_NWKtoMAC_CMD
#11	0	Slave.NETWL	Slave.NIC.MAC	This is a test data message
#12	0	Master.NIC.MAC	Master.NIC.MAC	event-SwitchState
#13	0	Slave.NIC.MAC	Slave.NIC.MAC	event-SwitchState
#14	0	Master.NIC.MAC	Master.NIC.MAC	timer-InitiatingScanIntervalTimer
#15	0.000004	Master.NIC.MAC	Master.NIC.PHY	radio switching over
#16	0.000004	Master.NIC.PHY	Master.NIC.MAC	Radio switching over
#17	0.1	Master.NIC.MAC	Master.NIC.MAC	timer-InitiatingScanIntervalTimer
#18	0.108472	Slave.NIC.MAC	Slave.NIC.MAC	timer-AdvertisementEvent
#19	0.108476	Slave.NIC.MAC	Slave.NIC.PHY	radio switching over
#20	0.108476	Slave.NIC.MAC	Slave.NIC.PHY	ADV_DIRECT_IND
#21	0.108476	Slave.NIC.PHY	Slave.NIC.MAC	Radio switching over
#22	0.108476	Slave.NIC.PHY	Master.NIC.PHY	ADV_DIRECT_IND
#23	0.108652	Slave.NIC.PHY	Slave.NIC.PHY	transmission over
#24	0.108652	Slave.NIC.PHY	Slave.NIC.MAC	Transmission over
#25	0.108652	Master.NIC.PHY	Master.NIC.PHY	ADV_DIRECT_IND
#26	0.108802	Slave.NIC.MAC	Slave.NIC.PHY	radio switching over
#27	0.108802	Slave.NIC.PHY	Slave.NIC.MAC	Radio switching over
#28	0.118326	Slave.NIC.MAC	Slave.NIC.MAC	timer-AdvertisementNextPDU
#29	0.118476	Slave.NIC.MAC	Slave.NIC.PHY	radio switching over
#30	0.118476	Slave.NIC.MAC	Slave.NIC.PHY	ADV_DIRECT_IND
#31	0.118476	Slave.NIC.PHY	Slave.NIC.MAC	Radio switching over
#32	0.118476	Slave.NIC.PHY	Master.NIC.PHY	ADV_DIRECT_IND
#33	0.118652	Slave.NIC.PHY	Slave.NIC.PHY	transmission over
#34	0.118652	Slave.NIC.PHY	Slave.NIC.MAC	Transmission over
#35	0.118652	Master.NIC.PHY	Master.NIC.PHY	ADV_DIRECT_IND
#36	0.118802	Slave.NIC.MAC	Slave.NIC.PHY	radio switching over
#37	0.118802	Slave.NIC.PHY	Slave.NIC.MAC	Radio switching over
#38	0.128326	Slave.NIC.MAC	Slave.NIC.MAC	timer-AdvertisementNextPDU
#39	0.128476	Slave.NIC.MAC	Slave.NIC.PHY	radio switching over
#40	0.128476	Slave.NIC.MAC	Slave.NIC.PHY	ADV_DIRECT_IND
#41	0.128476	Slave.NIC.PHY	Slave.NIC.MAC	Radio switching over
#42	0.128476	Slave.NIC.PHY	Master.NIC.PHY	ADV_DIRECT_IND
#43	0.128652	Slave.NIC.PHY	Slave.NIC.PHY	transmission over
#44	0.128652	Slave.NIC.PHY	Slave.NIC.MAC	Transmission over
#45	0.128652	Master.NIC.PHY	Master.NIC.PHY	ADV_DIRECT_IND
#46	0.128802	Slave.NIC.MAC	Slave.NIC.PHY	radio switching over
#47	0.128802	Slave.NIC.PHY	Slave.NIC.MAC	Radio switching over
#48	0.138326	Slave.NIC.MAC	Slave.NIC.MAC	timer-AdvertisementNextPDU

#49	0.138326	Slave.NIC.MAC	Slave.NIC.MAC	timer-AdvertisementEndEvent
#50	0.2	Master.NIC.MAC	Master.NIC.MAC	timer-InitaitingScanIntervalTimer
#51	0.214708	Slave.NIC.MAC	Slave.NIC.MAC	timer-AdvertisementEvent
#52	0.214712	Slave.NIC.MAC	Slave.NIC.PHY	radio switching over
#53	0.214712	Slave.NIC.MAC	Slave.NIC.PHY	ADV_DIRECT_IND
#54	0.214712	Slave.NIC.PHY	Slave.NIC.MAC	Radio switching over
#55	0.214712	Slave.NIC.PHY	Master.NIC.PHY	ADV_DIRECT_IND
#56	0.214888	Slave.NIC.PHY	Slave.NIC.PHY	transmission over
#57	0.214888	Slave.NIC.PHY	Slave.NIC.MAC	Transmission over
#58	0.214888	Master.NIC.PHY	Master.NIC.PHY	ADV_DIRECT_IND
#59	0.215038	Slave.NIC.MAC	Slave.NIC.PHY	radio switching over
#60	0.215038	Slave.NIC.PHY	Slave.NIC.MAC	Radio switching over
#61	0.224562	Slave.NIC.MAC	Slave.NIC.MAC	timer-AdvertisementNextPDU
#62	0.224712	Slave.NIC.MAC	Slave.NIC.PHY	radio switching over
#63	0.224712	Slave.NIC.MAC	Slave.NIC.PHY	ADV_DIRECT_IND
#64	0.224712	Slave.NIC.PHY	Slave.NIC.MAC	Radio switching over
#65	0.224712	Slave.NIC.PHY	Master.NIC.PHY	ADV_DIRECT_IND
#66	0.224888	Slave.NIC.PHY	Slave.NIC.PHY	transmission over
#67	0.224888	Slave.NIC.PHY	Slave.NIC.MAC	Transmission over
#68	0.224888	Master.NIC.PHY	Master.NIC.PHY	ADV_DIRECT_IND
#69	0.225038	Slave.NIC.MAC	Slave.NIC.PHY	radio switching over
#70	0.225038	Slave.NIC.PHY	Slave.NIC.MAC	Radio switching over
#71	0.234562	Slave.NIC.MAC	Slave.NIC.MAC	timer-AdvertisementNextPDU
#72	0.234712	Slave.NIC.MAC	Slave.NIC.PHY	radio switching over
#73	0.234712	Slave.NIC.MAC	Slave.NIC.PHY	ADV_DIRECT_IND
#74	0.234712	Slave.NIC.PHY	Slave.NIC.MAC	Radio switching over
#75	0.234712	Slave.NIC.PHY	Master.NIC.PHY	ADV_DIRECT_IND
#76	0.234888	Slave.NIC.PHY	Slave.NIC.PHY	transmission over
#77	0.234888	Slave.NIC.PHY	Slave.NIC.MAC	Transmission over
#78	0.234888	Master.NIC.PHY	Master.NIC.PHY	ADV_DIRECT_IND
#79	0.235038	Slave.NIC.MAC	Slave.NIC.PHY	radio switching over
#80	0.235038	Slave.NIC.PHY	Slave.NIC.MAC	Radio switching over
#81	0.244562	Slave.NIC.MAC	Slave.NIC.MAC	timer-AdvertisementNextPDU
#82	0.244562	Slave.NIC.MAC	Slave.NIC.MAC	timer-AdvertisementEndEvent
#83	0.3	Master.NIC.MAC	Master.NIC.MAC	timer-InitaitingScanIntervalTimer
#84	0.318551	Slave.NIC.MAC	Slave.NIC.MAC	timer-AdvertisementEvent

Tabla 3 – Eventos simulación Maestro - Esclavo

En la tabla 3 podemos observar lo ya comentado anteriormente, los eventos de una simulación con un dispositivo maestro y otro esclavo, y con todos los intervalos iguales a 100 ms. Se han dividido por colores para diferenciar mejor los conjuntos de eventos con un mismo papel:

- **Gris:** Eventos de inicialización.
- **Rosa:** Eventos de comienzo del intervalo de publicidad.
- **Verde:** Eventos de transmisión por el canal 37.
- **Azul:** Eventos de transmisión por el canal 38.
- **Morado:** Eventos de transmisión por el canal 39
- **Naranja:** Eventos de fin del intervalo de publicidad.

En el apartado anterior, me centré en hablar de los mensajes y entre qué dos módulos se enviaban. Sin embargo, en este apartado voy a hablar más en profundidad de los tiempos en los que suceden todos estos eventos. En primer lugar, podemos ver cómo todos los eventos de inicialización suceden en el segundo cero, exceptuando los cambios en la radio que se efectúan pasados 0.000004 segundos. Este mensaje es el que dará paso, casi 100 ms después (tiempo que vale el Tadv, TscanWindow y TscanInterval) al comienzo del intervalo de publicidad y de escaneo.

En segundo lugar, nos encontramos con el mensaje que inicializa el intervalo de escaneo exactamente en 0.1 segundos, así como el que inicializa el intervalo de publicidad unos milisegundos después. Este pequeño retraso se debe al tiempo de descubrimiento, que será ligeramente superior al intervalo de publicidad (Tadv = 100 ms, en este caso).

En tercer lugar, nos encontramos con una serie de eventos que suceden para que el dispositivo esclavo se comunique con el maestro a través del primero de los canales de publicidad, el 37. Este mismo conjunto de eventos se repite otras dos veces, una para transmitir por el canal 38 y otra para hacerlo por el canal 39. Además, podemos ver que el tiempo que tarda en transmitirse por un solo canal en muy similar en todos los casos, unos 10 ms.

En cuarto lugar, tras pasar los tres conjuntos de eventos correspondientes a la transmisión por los tres canales de publicidad, nos encontramos con el mensaje que indica que ha finalizado el intervalo de publicidad. Es decir, indica que se ha terminado de transmitir con éxito por el último de los canales.

Por último, hay que destacar que todo lo sucedido hasta ahora se repite las veces que nosotros queramos hasta que finalicemos la simulación. Podemos ver como el siguiente mensaje que inicializa los intervalos de escaneo y de publicidad (ya que son iguales en este caso) no llegará hasta el segundo 0.2, exactamente 100 ms después de que se recibiese el anterior mensaje de inicialización del intervalo. De aquí en adelante todo se repite de la misma forma que hemos visto antes.

Antes de finalizar con este apartado, es interesante mencionar que la transmisión por los tres canales de publicidad dura aproximadamente 30 ms. Este tiempo se podría alargar hasta casi 40 ms si incluimos los tiempos que pasan antes y después de transmitir, así como los tiempos entre canales. Este tiempo de 40 ms es muy significativo porque es el tiempo mínimo que debería de tener el intervalo de publicidad para que no se pierdan paquetes y el dispositivo esclavo se pueda comunicar con el maestro por los tres canales de publicidad. En este ejemplo el tiempo de publicidad es de 100 ms, por lo que no habrá problema. Además, el intervalo de escaneo y la ventana de escaneo (iguales para perder el mínimo tiempo posible y transmitir el 100 % del tiempo), también tendrán el valor de 100 ms para que tanto el intervalo de publicidad como el de escaneo comiencen y terminen a la vez y se transmita toda la información sin problemas.

6.3 CONSUMO

En este apartado se va a analizar el consumo en los dispositivos participantes en diferentes tipos de comunicaciones, cambiando alguno de sus parámetros. Los tres consumos que se van a mostrar son:

- Consumo medio de potencia del dispositivo Esclavo.
- Consumo medio de potencia del dispositivo Maestro.
- Consumo medio de corriente del dispositivo Maestro.

Para ver cómo cambian estos consumos, iremos probando con distintas configuraciones de parámetros. La mayor parte de las gráficas que se mostrarán serán resultado de introducir todos los consumos en Excel y ver cómo fluctúan estos valores para cada una de las configuraciones.

6.3.1 VARIACIÓN EN LOS TIEMPOS

En este apartado vamos a ver cómo se comporta el consumo de potencia de los dos dispositivos participantes en la comunicación. En nuestro caso, emplearemos un dispositivo Esclavo y otro Maestro y variaremos los intervalos conjuntamente. Estos intervalos serán el de publicidad (Tadv), el de escaneo (TscanInterval) y la ventana de escaneo (TscanWindow). El valor de estos tres intervalos siempre coincidirá e irá aumentando hasta llegar a los 1000 ms. Además, para que todas las simulaciones se puedan comparar, se han llevado a cabo ejecutando el programa hasta el evento número 314. Si lo hubiésemos ejecutado hasta una misma cantidad de segundos no tendría sentido, ya que, al aumentar los intervalos, aumentaría de por sí el tiempo de transmisión, por lo que no se enviaría la misma cantidad de información.

T (ms)	Consumo medio Maestro (mA)	Consumo medio Maestro (mW)	Consumo medio Esclavo (mW)
50	15,711507	66,299746	33,877651
100	15,555063	66,299863	18,232198
150	15,555063	66,299906	12,486333
200	15,555063	66,299929	9,502352
250	15,555063	66,299942	7,674767
300	15,555063	66,299952	6,440474
350	15,555063	66,299958	5,550916
400	15,555063	66,299963	4,879635
450	15,555063	66,299967	4,354427
500	15,555063	66,299971	3,932815
550	15,555063	66,299973	3,586755
600	15,555063	66,299975	3,297609
650	15,555063	66,299977	3,052403
700	15,555063	66,299979	2,841829
750	15,555063	66,299980	2,659034
800	15,555063	66,299981	2,498861
850	15,555063	66,299982	2,357354
900	15,555063	66,299983	2,231432
950	15,555063	66,299984	2,118653
1000	15,555063	66,299985	2,017063

Tabla 4 – Cambio en los consumos en función de los tiempos

En la tabla 4 podemos observar el cambio en los consumos de los dispositivos participantes en la comunicación al cambiar los tiempos. A primera vista no se notan grandes cambios en el consumo del Scanner, ni de corriente ni de potencia. Sin embargo, en el consumo de potencia del Advertiser sí que se aprecia una notable caída en picado a medida que se aumentan los tiempos ya antes mencionados.

Para ver mejor qué está sucediendo vamos a plasmar estos datos en distintas tablas con ayuda del Excel.

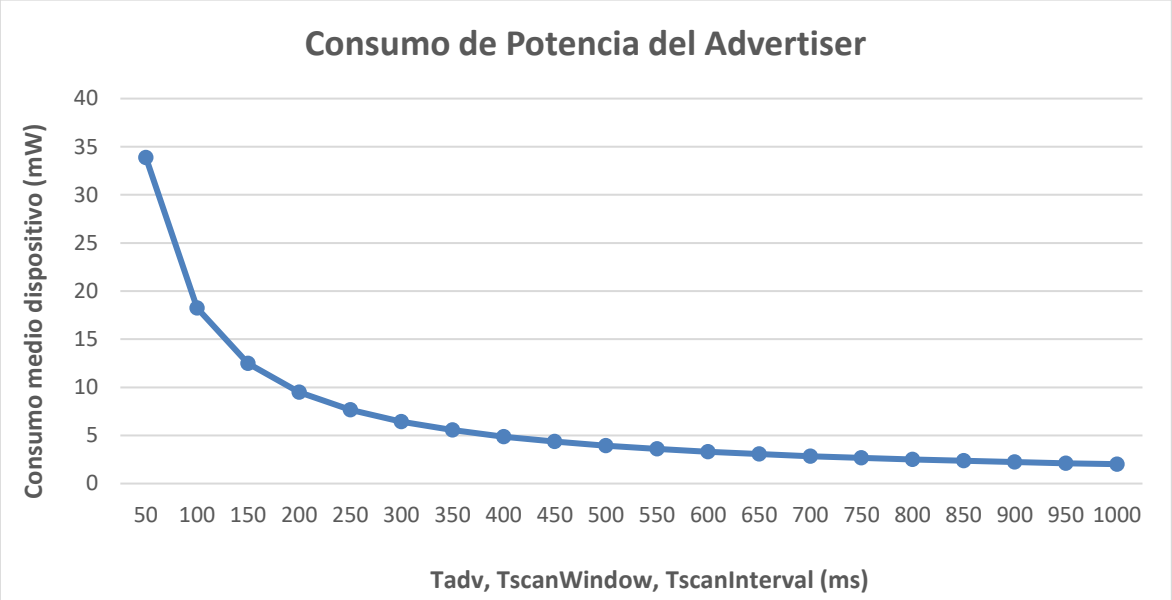


Figura 53 – Consumo de potencia del Advertiser para diferentes tiempos

En la figura 53 podemos ver el consumo de potencia del dispositivo Esclavo o Advertiser, cambiando los intervalos ya antes mencionados. La primera conclusión que se puede sacar es que, al aumentar los tiempos, disminuye este consumo medio de potencia en el dispositivo. Además, es interesante que sufre una considerable caída en picado al aumentar el intervalo de 50 a 100 ms. En ese punto, el consumo en el Advertiser disminuye unos 15 mW y luego continúa disminuyendo de una manera menos agresiva.

Este resultado tiene bastante sentido, ya que cuanto mayor sean los intervalos de publicidad y de escaneo, más tiempo tiene el Advertiser en transmitir información al Scanner por cada uno de los tres canales.

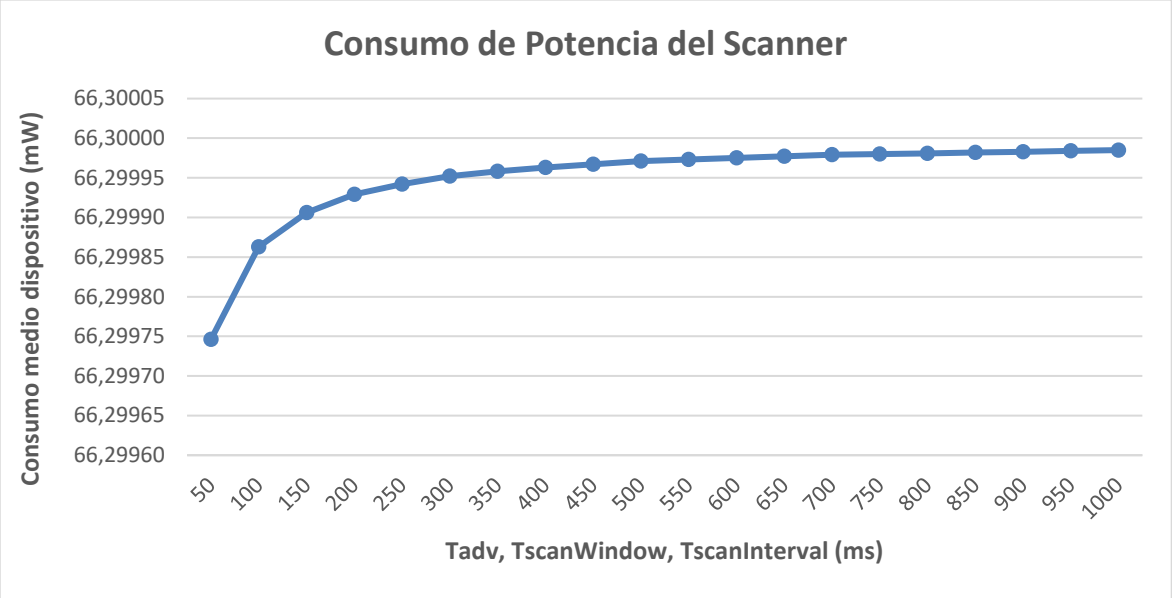


Figura 54 - Consumo de potencia del Scanner al cambiar los tiempos

En la figura 54 vemos el mismo consumo que antes, solo que para el dispositivo Maestro o Scanner. En primera instancia, veíamos en la tabla que no parecía cambiar el valor de este consumo. Sin embargo, viendo la gráfica generada, se ve que aumenta muy levemente dicha potencia consumida. En el caso del Advertiser, veíamos que el consumo se llegaba a reducir hasta 30 mW, pero en este caso podemos observar que aumenta hasta 0.2 micro Watios. El aumento es casi imperceptible, por lo que se podría concluir que un aumento del intervalo de publicidad y de escaneo, no afecta al consumo de potencia del dispositivo Maestro.

Se puede entender que, por mucho que aumenten estos intervalos, el dispositivo Esclavo siempre transmitirá la misma cantidad de información al dispositivo Maestro. Por lo tanto, el consumo de este último no debería de variar pese al aumento de los tiempos.

En cuanto a la corriente consumida por el Scanner, no se incluirá la gráfica generada por no proporcionar unas conclusiones lo suficientemente sólidas, seguramente debido a ciertos fallos en el módulo BLE.

6.3.2 VARIACIÓN EN EL NÚMERO DE DISPOSITIVOS

Al igual que en el apartado anterior, en este veremos como cambia el consumo de potencia y de corriente de los dispositivos Scanner y Advertiser, solo que ahora cambiaremos el número de estos. En este caso y, para que todas las mediciones sean lo más parecidas posibles, se ha mantenido constante el valor de los tiempos: Tadv, TscanInterval y TscanWindow = 100 ms. Además, a diferencia del apartado anterior, donde simulábamos hasta el evento 314, ahora simularemos hasta el segundo 1, ya que los intervalos se han mantenido constantes y necesitamos que todos los dispositivos, en cualquiera de los casos, envíen el mismo número de paquetes.

Nº Maestros	Nº Esclavos	Consumo medio Nodo (mA)	Consumo medio un Maestro (mA)	Consumo medio un Esclavo (mA)
1	**1**	15,555063	66,299863	18,232198
1	**2**	15,607606	66,299865	17,978530
1	**3**	15,625393	66,299864	18,102601
1	**4**	15,634340	66,299866	17,878428
1	**5**	15,639724	66,299866	17,901800
1	**6**	15,643321	66,299865	18,024822
1	**7**	15,645893	66,299864	18,082962
1	**8**	15,647825	66,299864	18,141269
1	**9**	15,649328	66,299866	17,904580
1	**10**	15,650531	66,299865	17,969009
1	1	15,555063	66,299863	18,232198
2	1	15,453622	66,299866	17,886871
3	1	15,356656	66,299863	18,286326
4	1	15,263875	66,299863	18,215807
5	1	15,175015	66,299865	18,036765
6	1	15,089832	66,299864	18,152275
7	1	15,008103	66,299864	18,089149
8	1	14,929620	66,299864	18,112241
9	1	14,854196	66,299865	18,009710
10	1	14,781654	66,299866	17,864943

Tabla 5 – Cambios en los consumos en función del número de dispositivos

En la tabla 5, podemos ver cómo cambian algunos consumos importantes si parametrizamos el número de Advertiser y de Scanners. La tabla y los experimentos se van a dividir en dos grupos: Uno en el dejamos 1 Scanner y vamos aumentando el número de Advertisers y otro en el que nos quedamos con 1 Advertiser y vamos aumentando el número de Scanners. A simple vista no se pueden extraer conclusiones claras, así que pasemos a representarlo gráficamente y veamos qué sucede.

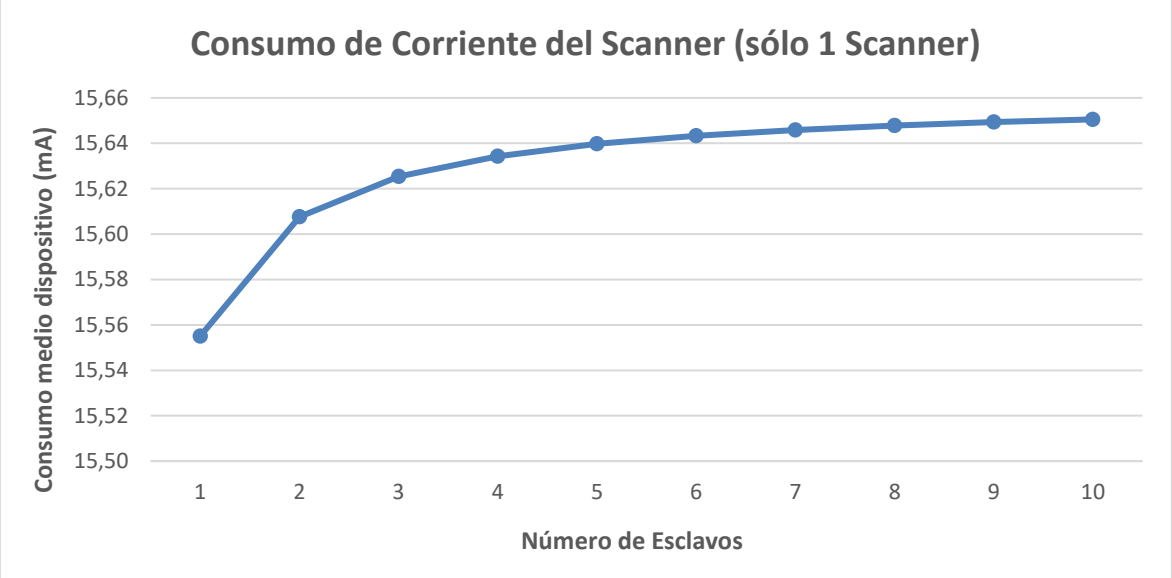


Figura 55 – Consumo de corriente del Scanner al cambiar el número de Advertisers

En la figura 55 vemos el cambio en el consumo de la corriente del dispositivo Maestro o Scanner a medida que aumentamos el número de Advertisers o dispositivos Esclavos, todo ello empleando siempre un solo Scanner. Podemos observar que aumenta ligeramente hasta alcanzar un valor casi constante. El aumento es muy leve, de unos 0.1 mA, pero aumenta, al fin y al cabo.

Por otro lado, tiene sentido que aumente el consumo ya que cuantos más Advertisers estén conectándose con el Scanner, mayor será la corriente que consuma para poder devolver información a todos los dispositivos.

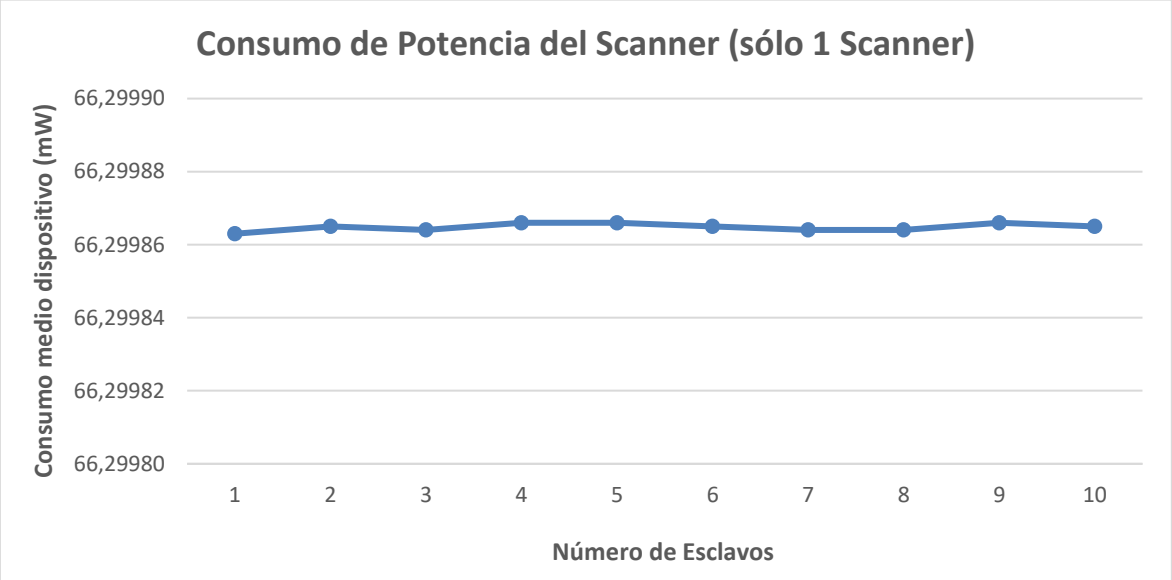


Figura 56 – Consumo de potencia del Scanner al cambiar el número de Advertisers

En la figura 56 podemos observar la potencia consumida por el dispositivo Maestro o Scanner a medida que aumentamos el número de Advertisers. Es interesante fijarse también en la gráfica anterior, en donde existían valores con los mismos parámetros y se estaba midiendo también el consumo del Scanner, solo que la corriente. En dicha gráfica, la corriente consumida aumentaba ligeramente a medida que aumentaban los Advertisers en la comunicación. Ese aumento tan leve de corriente consumida por el Scanner se transmite en un consumo de potencia casi constante, como vemos en esta gráfica.

Para entender mejor este valor constante pongamos un ejemplo: Si tenemos una comunicación en la que participa un dispositivo Maestro y tres Esclavos, y todos los intervalos (publicidad y escaneo) son iguales a 100 ms, en cada canal de publicidad habrá tres eventos clave. En cada uno de ellos, será uno de los Advertisers el que haga broadcast en la red, enviando información al Maestro y a los otros dos Eslavos. En definitiva, el tiempo en el que el Maestro está recibiendo información es casi el mismo, por lo que el consumo será casi idéntico al aumentar el número de Esclavos.

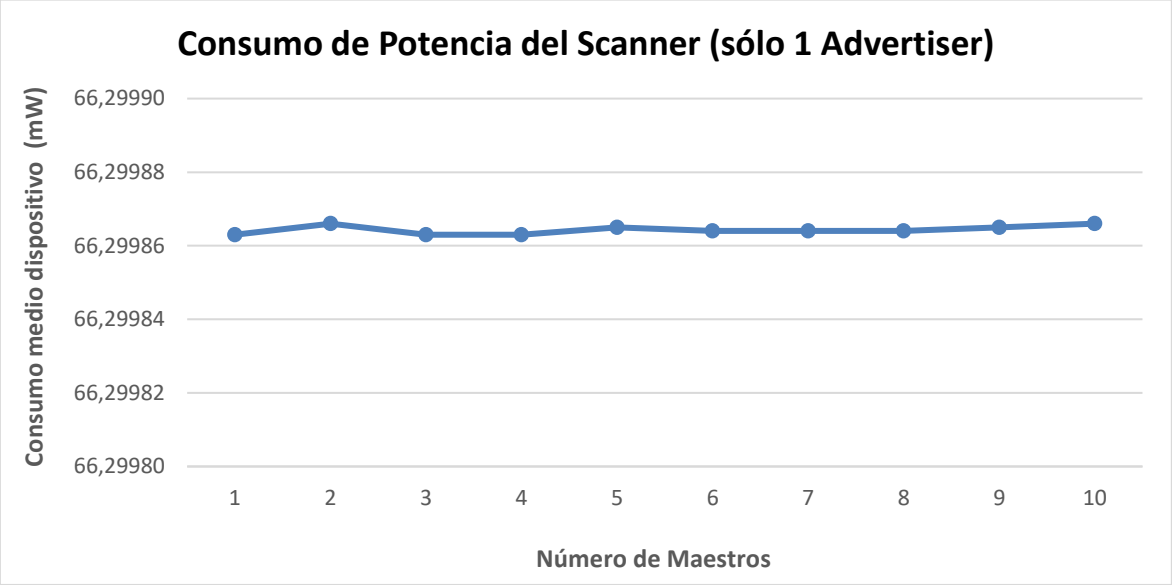


Figura 57 – Consumo de potencia del Scanner al cambiar el número de Scanners

En la figura 57 podemos ver como varía el consumo de la potencia del dispositivo maestro en función del número de Scanners y con un solo Advertiser. Al igual que en el caso anterior, esta potencia casi no varía, oscila en un rango de 0.1 micro Watios. Este consumo casi constante de la potencia por parte del dispositivo Maestro es bastante lógico. Si en la comunicación hay un solo dispositivo esclavo y varios dispositivos maestros, este esclavo se comunicará con todos y cada uno de los maestros por igual, pese a que tarde un poco más de tiempo. Por lo tanto, el consumo de uno de los Scanners participantes no se ve prácticamente afectado.

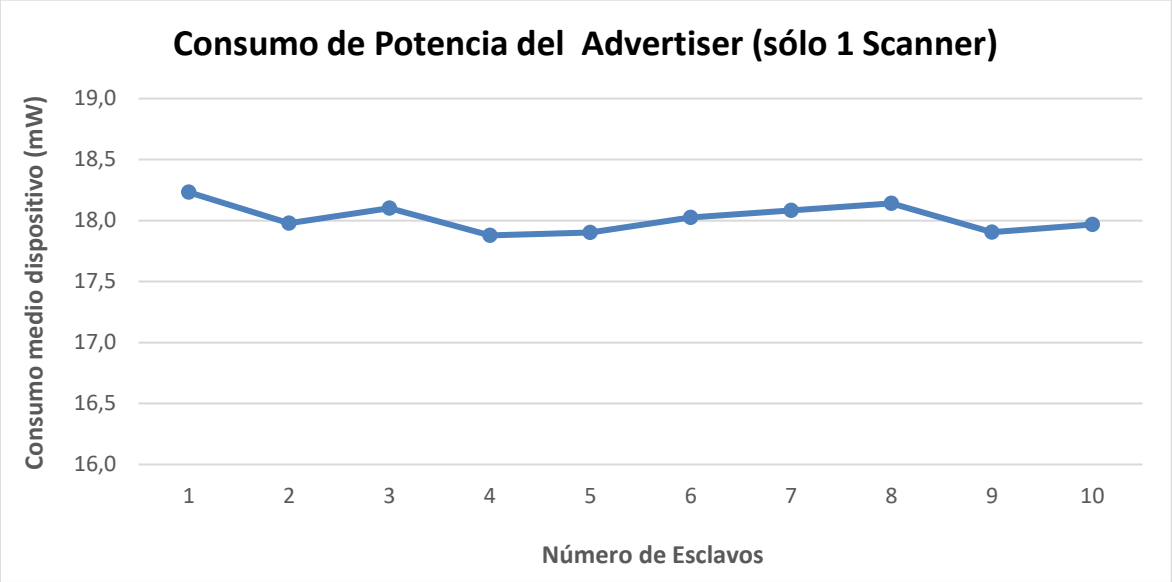


Figura 58 – Consumo de potencia del Advertiser al cambiar el número de Advertisers

En la figura 58 nos encontramos con el cambio en la potencia consumida del dispositivo Esclavo o Advertiser si aumentamos el número de ellos, manteniendo constante un solo dispositivo Maestro o Scanner en la comunicación. A diferencia de con las últimas gráficas, ahora el orden de magnitudes es algo mayor, por lo que los cambios en el consumo son algo más notables. Sin embargo, se puede seguir viendo que el consumo es casi constante por la misma razón que antes. Si aumentamos el número de Advertisers, cada uno de ellos se comunicará por turnos con el Scanner, por lo que seguirán consumiendo más o menos lo mismo cada uno de ellos (recordemos que la gráfica representa el valor medio).

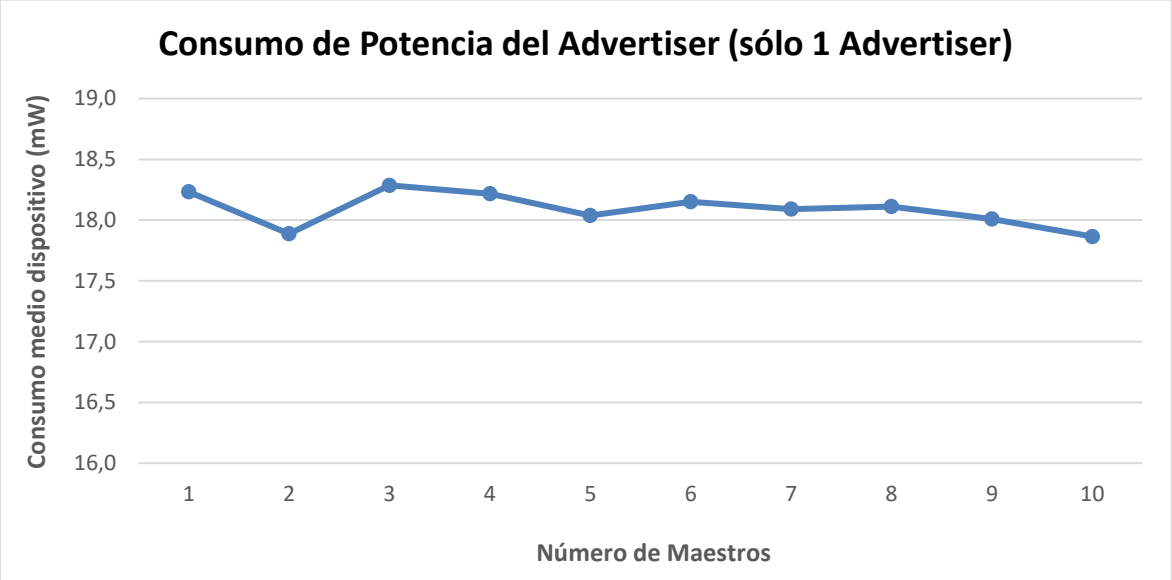


Figura 59 – Consumo de potencia del Advertiser al cambiar el número de Scanners

En la figura 59 mostramos la última de las gráficas de este apartado, en la cual podemos observar el cambio en la potencia consumida por parte del dispositivo Esclavo o Advertiser en función del número de Scanners. Unas páginas más atrás, vimos el consumo de uno de los Scanners en este mismo caso, y veíamos que el consumo era casi constante. En este caso sucede algo parecido, si un solo Advertiser se quiere comunicar con varios Scanners, se ajustarán los tiempos para que pueda conectarse con todos prácticamente al mismo tiempo, por lo que, aunque aumente el número de maestros, el consumo siempre será mas o menos el mismo.

6.4 RADIO

En este último apartado se van a analizar los cambios de estado de la radio, así como los cambios entre los diferentes canales de publicidad de la radio. Las gráficas que se mostrarán son resultado de la simulación y no se ven afectadas por cambios de parámetros ya que, en BLE siempre se transmite por los tres últimos canales (37, 38 y 39) y los cambios en el estado de la radio suceden de manera periódica en la comunicación.

Por lo tanto, debido a que las variaciones en los parámetros no afectan a estas gráficas, la configuración que se seguirá será la misma que la del apartado 5.2.2 (Traza), y así podremos ver cómo concuerdan todos los tiempos.

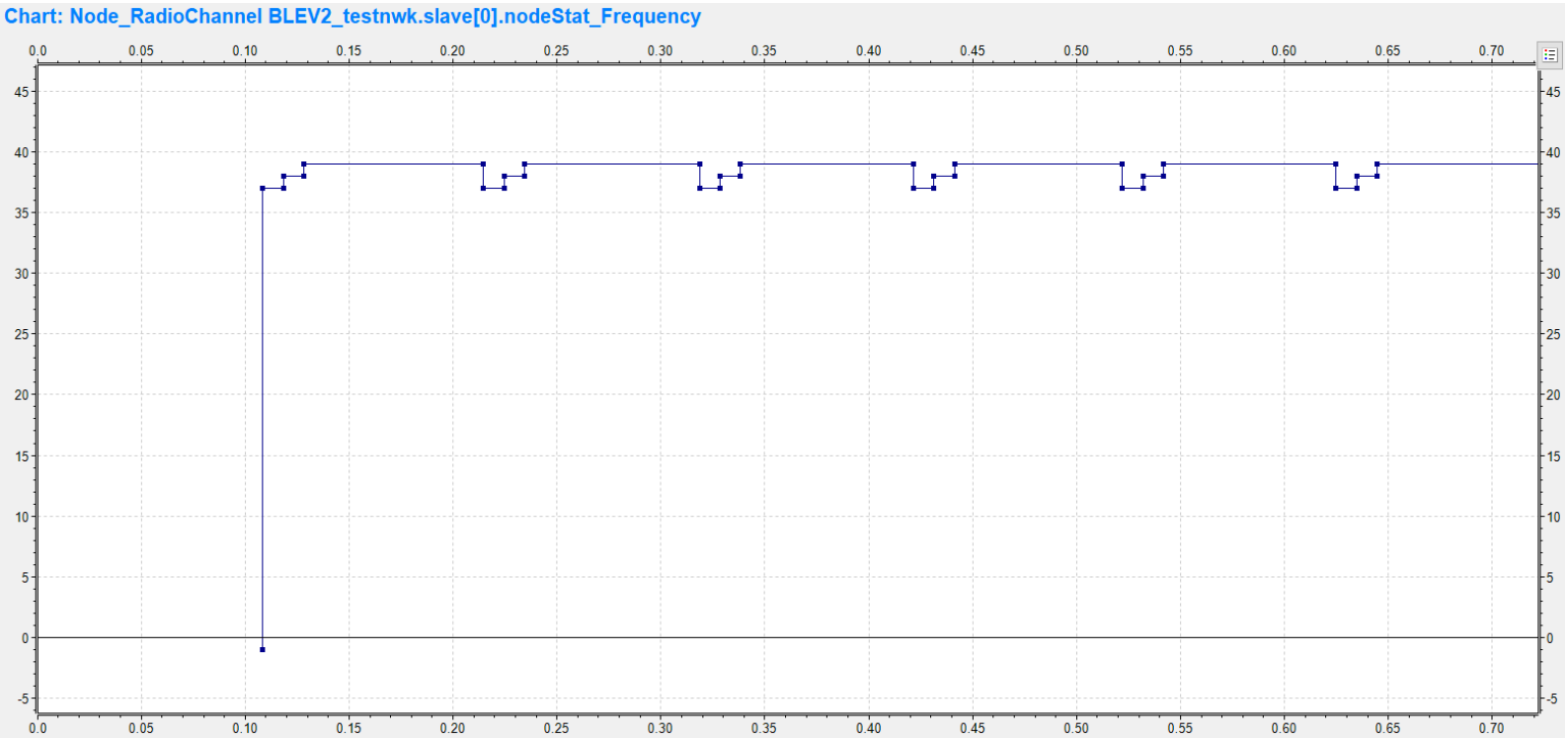


Figura 60 – Cambios en los canales de la radio

En la figura 60 podemos ver los cambios mencionados en los canales de publicidad de la radio. En primer lugar, la radio se encuentra sin transmitir (canal con valor -1), pero inmediatamente comienza a transmitir en el canal 37 un cierto tiempo, más tarde pasa al 38 y finalmente al 39. El tiempo que pasa hasta que vuelve a transmitir en el canal 37 es lo que falta para que finalice el tiempo de publicidad asignado (100 ms) y esto se repite indefinidamente.

Veamos más en detalle estos tiempos en los que cambia el canal de la radio, haciendo referencia a los tiempos vistos en la traza del apartado 5.2.2.

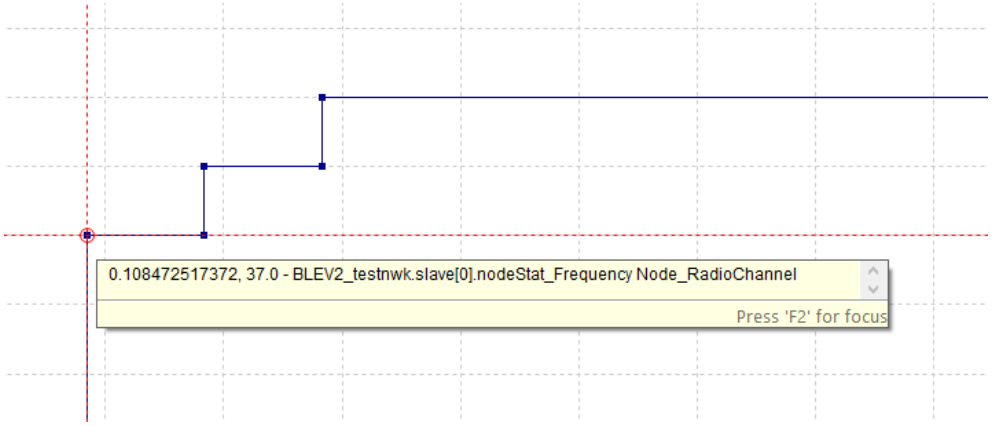


Figura 61 – Comienzo de la radio a transmitir en el canal 37

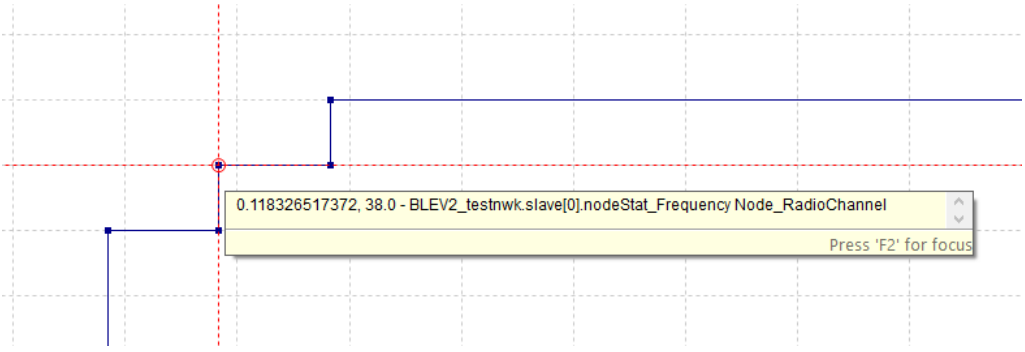


Figura 62 – Comienzo de la radio a transmitir en el canal 38

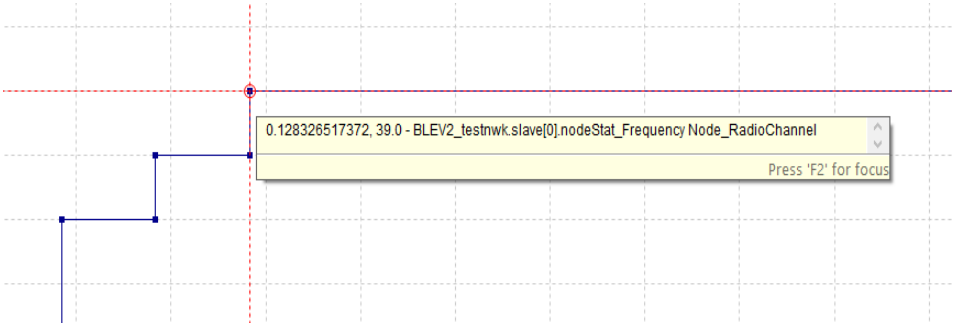


Figura 63 – Comienzo de la radio a transmitir en el canal 39

En las figuras 61, 62 y 63 vemos el momento exacto en el que la radio comienza a transmitir por primera vez en cada uno de los tres canales de publicidad mencionados de la comunicación BLE. Si nos fijamos, el eje Y que muestra el canal de la radio corresponde con el canal de publicidad en el que se está transmitiendo en ese momento. Para ello hay que fijarse en el tiempo exacto que marca el eje X:

- Canal 37: 0.108472 segundos.
- Canal 38: 0.118326 segundos.
- Canal 39: 0.128326 segundos

Estos tiempos concuerdan casi a la prefección con los vistos en los eventos de la traza del apartado 5.2.2.

En este apartado, como ya se ha mencionado antes, también se va a mostrar gráficamente cómo cambia el estado de la radio a medida que pasa el tiempo en la simulación. Al igual que en el cambio del canal de publicidad, estos tiempos van a concordar con los eventos donde la radio cambiaba de estado en el apartado 5.2.2.

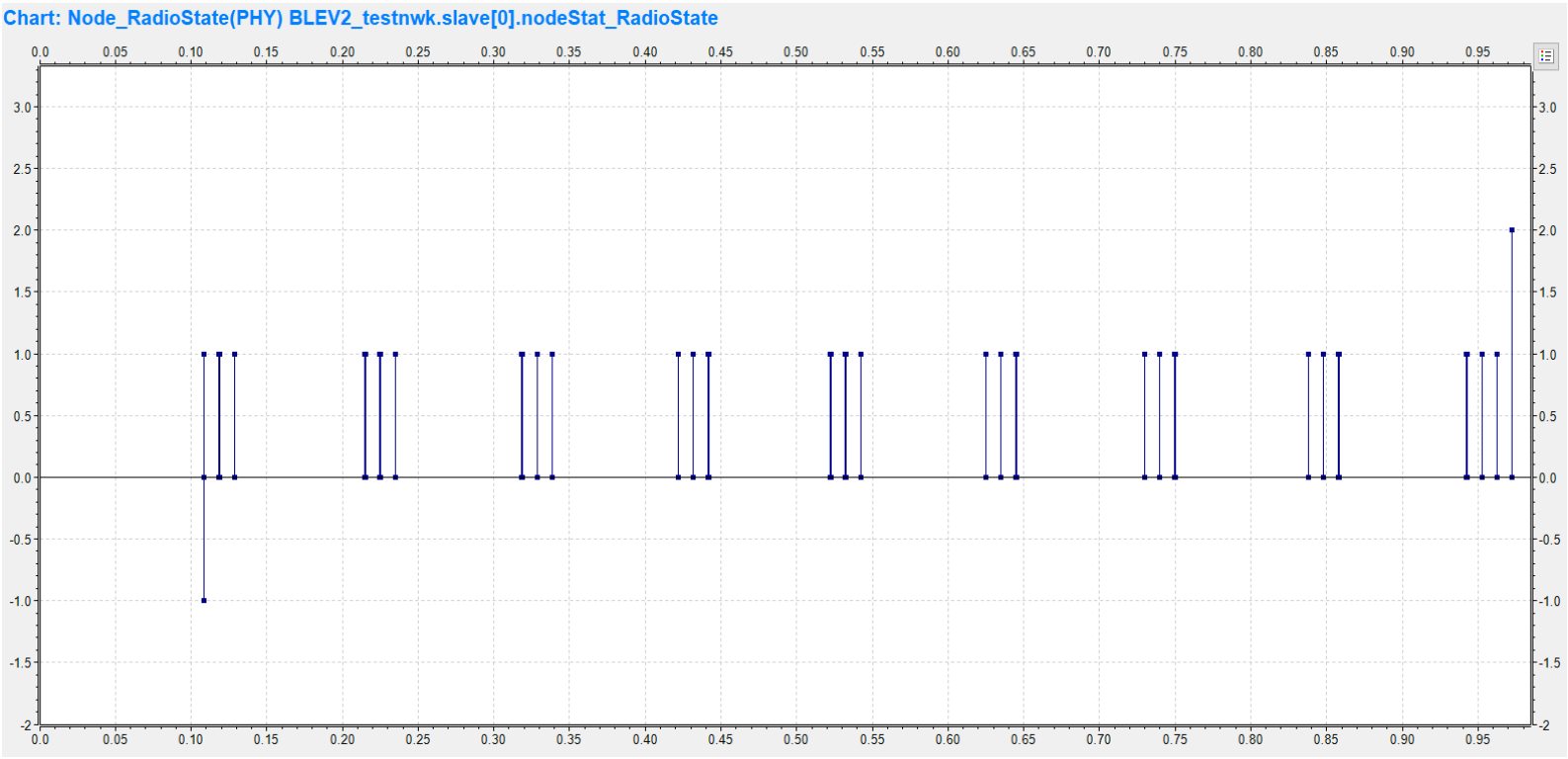


Figura 64 – Cambios en el estado de la radio

En la figura 64 podemos ver estos cambios de estado de la radio ya antes comentados. Al igual que sucedía en los cambios de canal, el valor de inicialización de dicho estado es de -1 y luego va alternando entre 1 y 0. Se puede observar como cada 100 ms suceden una serie de cambios de estado debidos a los cambios en los canales de publicidad.

En la gráfica, el estado de la radio se encuentra en 1 cuando la radio comienza a transmitir por un nuevo canal de publicidad y baja a 0 tras el envío de solicitud de conexión del dispositivo esclavo al maestro. Los tiempos exactos de dicha gráfica no los voy a mostrar pero concuerdan a la perfección con los mostrados en los eventos del apartado 5.2.2, en donde se recibía el mensaje “radio swtiching over”, indicando el ya mencionado cambio de estado de la radio.

Capítulo 7. CONCLUSIONES Y TRABAJOS FUTUROS

El objetivo principal de este proyecto era el de conocer mejor el funcionamiento de la tecnología de comunicaciones inalámbricas BLE gracias a la ayuda de entornos simulados y experimentales. En primera instancia, como ya se ha comentado, se iba a utilizar el simulador de red NS-3 pero, debido a sus problemas para trabajar con módulos BLE, se decidió continuar con el Omnet ++, el cual sí que proporcionaba resultados algo más concluyentes.

La tecnología BLE es conocida por su bajo consumo, por lo que el parámetro principal a estudiar a lo largo del proyecto ha sido este, el consumo de los dispositivos participantes en una comunicación simulada (Maestros y Esclavos). Esto se ha realizado cambiando una serie de parámetros muy relevantes en una comunicación BLE como los intervalos o el número de dispositivos. Todos estos resultados se muestran, como es lógico, en el capítulo de Resultados de este documento, mientras que en los capítulos previos se habla de todo lo demás relacionado con este proyecto: arquitectura de BLE, módulo utilizado para su estudio y simuladores empleados a lo largo del tiempo de vida del proyecto.

Por último, hay que mencionar que por falta de tiempo no se pudo probar esta misma comunicación utilizando el hardware específico, todas las pruebas han sido realizadas con el simulador de red Omnet ++. Además, el módulo BLE empleado en el simulador tenía algunas carencias y el autor de dicho módulo se ha desentendido, por lo que no hfue posible contactar con él. De todas formas, se han obtenido algunos resultados muy interesantes en cuanto al consumo, pero no son tan sólidos como se podía esperar, por lo que en un futuro se podría mejorar este módulo BLE empleado para mejorar las conclusiones respecto a esta tecnología. Además, también se podría comparar este mismo estudio en entornos experimentales con otro en entornos reales, para así conseguir una mejor y más eficiente conclusión sobre la tecnología de comunicaciones inalámbricas BLE.

Capítulo 8. BIBLIOGRAFÍA

[1] "Introduction to Bluetooth Low Energy (BLE) v4.0". Argenox Technologies LLC. http://www.argenox.com/bluetooth-low-energy-ble-v4-0development/library/introduction-to-bluetooth-low-energy-v4-0/

[2] "Diseño del Bluetooth de baja energía 101: Desde chipsets hasta pilas de protocolo hasta módulos". Digi-Key Electronics. https://www.digikey.es/es/articles/techzone/2016/jul/bluetooth-low-energy-design-101-from-chipsets-to-protocol-stacks-to-modules

[3] "BLE-Stack User's Guide for Bluetooth 4.2". Texas Instruments. http://dev.ti.com/tirex/content/simplelink_cc2640r2_sdk_1_40_00_45/docs/blestack/ble_user_guide/html/cc2640/architecture.html#

[4] "Bluetooth Low Energy – Part 1: Introduction to BLE". MicroElektronika. https://www.mikroe.com/blog/bluetooth-low-energy-part-1-introduction-ble

[5] "Bluetooth Low Energy". Wikipedia The Free Encyclopedia. https://en.wikipedia.org/wiki/Bluetooth_Low_Energy

[6] "NS-3: Network Simulator 3". Gustavo J. A. M. Carneiro, INESC Porto. https://www.nsnam.org/tutorials/NS-3-LABMEETING-1.pdf

[7] "Ns (simulator)" Wikipedia The Free Encyclopedia. https://en.wikipedia.org/wiki/Ns_(simulator)

[8] "Building a bus network topology". NS-3 A Discrete Event Simulator. https://www.nsnam.org/docs/tutorial/html/building-topologies.html

[9] "Adding a new module to ns-3". NS-3 A Discrete Event Simulator. https://www.nsnam.org/docs/release/3.26/manual/html/new-modules.html

[10] "Using the tracing system". NS-3 A Discrete Event Simulator. https://www.nsnam.org/docs/release/3.9/tutorial/tutorial_23.html

[11] "BLE Module". Zzma, GitHub. https://github.com/zzma/bleats/tree/master/ns-3.24/ns-3.24/src/ble

[12] "Tracing tutorial". NS-3 A Discrete Event Simulator. https://www.nsnam.org/docs/tutorial/html/tracing.html

[13] "Capítulo 4: Simulación en OMNeT++". Catarina Udlap. http://catarina.udlap.mx/u_dl_a/tales/documentos/lem/deschamps_e_me/capitulo4.pdf

[14] "Capítulo 3: Simulador OMNET++". Levy Bunan, Upcommons. https://upcommons.upc.edu/bitstream/handle/2099.1/.../Levy_Bunan_memòria_3.pdf

[15] "Introducción a OMNeT++". Javier Armendáriz Silva, Universidad Pública de Navarra. http://www.tlm.unavarra.es/research/seminars/slides/20090306_Javi_OMNeT.pdf

[16] "Tag Archives: OMNet++". Saurabh Kulkarni, All about wireless sensor networks. https://allaboutwirelesssensornetworks.wordpress.com/tag/omnet/

[17] "BLE images", ResearchGate. https://www.researchgate.net/figure/BLE-stack-architecture_fig3_259333192?_sg=uM9lARDyzIMww6oRHjqVXfYz8xdv3sh3sR13X1-XvFqef8U_qM-5pv2gEEJh5aVJ57pJF2QUzlVMN-Mf3t_Ssg

[18] "Generic Access Profile (GAP)". Texas Instruments. http://dev.ti.com/tirex/content/simplelink_cc2640r2_sdk_1_40_00_45/docs/blestack/ble_user_guide/html/ble-stack-3.x/gap.html

[19] "Gatt Overview". Bluetooth SIG. https://www.bluetooth.com/specifications/gatt/generic-attributes-overview

[20] "Generic Attribute Profile (GATT)". Texas Instruments. http://dev.ti.com/tirex/content/simplelink_cc2640r2_sdk_1_35_00_33/docs/ble5stack/ble_user_guide/html/ble-stack/gatt.html

[21] "Logical Link Control and Adaptation Layer Protocol (L2CAP)". Texas Instruments. http://dev.ti.com/tirex/content/simplelink_cc2640r2_sdk_1_35_00_33/docs/ble5stack/ble_user_guide/html/ble-stack/l2cap.html

[22] "Bluetooth Pairing Part 1 Pairing Feature Exchange". Bluetooth SIG. https://blog.bluetooth.com/bluetooth-pairing-part-1-pairing-feature-exchange

[23] "Host Controller Interface (HCI)". Texas Instruments. http://dev.ti.com/tirex/content/simplelink_cc2640r2_sdk_1_35_00_33/docs/ble5stack/ble_user_guide/html/ble-stack/hci.html

[24] "Diseño del Bluetooth de baja energía 101: desde chipsets y pilas de protocolo hasta módulo". Digi-Key Electronics. https://www.digikey.es/es/articles/techzone/2016/jul/bluetooth-low-energy-design-101-from-chipsets-to-protocol-stacks-to-modules

[25] "Bluetooth low energy (BLE) enabled devices market volume worldwide, from 2013 to 2020 (in million units)". Satista. https://www.statista.com/statistics/750569/worldwide-bluetooth-low-energy-device-market-volume/

[26] "Bluetooth low energy (Bluetooth smart) network simulator tool". Konstantin Mikhaylov, University of Oulu. http://cc.oulu.fi/~kmikhayl/BLE.html

[27] "A BLE Advertising Primer". Argenox Technologies LLC. http://www.argenox.com/a-ble-advertising-primer/

ANEXO

OMNETPP.INI

```
[General]
experiment-label = "BLEV2_TEST"
network = BLEV2_testnwk
ned-path = ../../src;..
tkenv-image-path = ../../images;


**.playgroundSizeX = 1000 m # x size of the area the nodes are in (in meters)
**.playgroundSizeY = 1000 m # y size of the area the nodes are in (in meters)
**.playgroundSizeZ = 1000 m # z size of the area the nodes are in (in meters)
#**.NumNodes = 2

############################################################
################ Channel parameters ####################

**.connectionManager.sendDirect = false             # send directly to the node or
create separate gates for every connection
**.connectionManager.pMax = 10 mW                   # maximum sending power used
for this network [mW]
**.connectionManager.sat = -90 dBm                  # minimum signal attenuation
threshold [dBm]
**.connectionManager.alpha = 2.0                    # minimum path loss
coefficient
**.connectionManager.carrierFrequency = 2.45e+9 Hz # minimum carrier frequency of
the channel [Hz]

############################################################
################ PhyLayer parameters ####################

**.nic.phy.usePropagationDelay = false            # should transmission delay be
simulated?
**.nic.phy.thermalNoise = -100 dBm                # the strength of the thermal
noise [dBm]
**.nic.phy.useThermalNoise = true                 # should thermal noise be
considered?
**.nic.phy.analogueModels = xmldoc("config.xml") # specification of the analogue
models to use and their parameters
**.nic.phy.decider = xmldoc("config.xml")         # specification of the decider
to use and its parameters

**.nic.phy.initialRadioState = 2 # state the radio is initially in (0=RX, 1=TX,
2=Sleep)
**.nic.phy.sensitivity = -93 dBm # the sensitivity of the physical layer [dBm]
```

```
**.nic.phy.maxTXPower = 10.0 mW  # the maximum transimission power of the
physical layer [mW]
**.nic.phy.nbRadioChannels = 40   # number of available radio channels. Defaults
to single channel radio.

############################################################
################ Current consumption ######################

**.nic.sleepCurrent = 0.0235 mA    #CC2540 datasheet (SWRS084F) p.4
**.nic.rxCurrent = 22.1 mA         #CC2540 datasheet (SWRS084F) p.5 (High-gain
mode)
**.nic.decodingCurrentDelta = 0 mA
**.nic.txCurrent = 27mA            #CC2540 datasheet (SWRS084F) p.6 for 0dbm
**.nic.setupRxCurrent = 11.05 mA   #just half of rxCurrent
**.nic.setupTxCurrent = 13.5 mA    #just half of txCurrent
**.nic.rxTxCurrent = 13.5 mA       #just half of txCurrent
**.nic.txRxCurrent = 11.05 mA      #just half of rxCurrent

############################################################
################ Battery ##################################

**.nic.phyType = "org.mixim.modules.phy.PhyLayerBattery"
**.battery.nominal = 2000 mAh  # nominal battery capacity
**.battery.capacity = 2000 mAh # battery capacity
**.battery.resolution = 30 s   # capacity is updated at least every resolution
time
**.battery.publishDelta = 0    # (0..1): capacity is published each time it is
observed to have changed by publishDelta * nominal_capacity
**.battery.publishTime = 30 s  # if > 0, capacity is published to the BB each
publishTime interval
**.battery.voltage =3 V        # nominal voltage

############################################################
## Switching times - SHOULD BE THE SAME AS FOR PHY LAYER! ##
#I do not want to change the basic files and there is no mechnism to get those
data to MAC level otherwise
# (for correct timing we need to have those on LL!)
#note - the TX-RX and RX-TX times should be equal to IFS (or at least not exceed
those!)
**.nic.mac.Time_llSLEEPtoTX = 0.000004 s
**.nic.mac.Time_llSLEEPtoRX = 0.000004 s
**.nic.mac.Time_llTXtoRX = 0.000150 s
**.nic.mac.Time_llRXtoTX = 0.000150 s
**.nic.mac.Time_llTXtoSLEEP = 0 s
**.nic.mac.Time_llRXtoSLEEP = 0 s

############################################################
################ MOBILITY ##################################

**.mobilityType = "StationaryMobility"
#**.mobilityType = "ConstSpeedMobility"
#**.master[*].mobility.initFromDisplayString = false
#**.slave[*].mobility.initFromDisplayString = false
```

```
#**.mobility.speed = 0mps
#**.master[0].mobility.initialX = 30m
#**.master[0].mobility.initialY = 30m
#**.slave[0].mobility.initialX = 30m
#**.slave[0].mobility.initialY = 40m

###################################################################
################ NWK & APP pars (those layer ARE NOT USED!!!) ####

**.networkType = "BLE_BasicNwk"
**.netwl.headerLength = 0bit
**.appl.headerLength = 0byte


**.nic.mac.TST_stopSimulation_ConnectionBreak = false
#**.nic.mac.connAccessAddress = -1 #if set to negative value - the index of the
device will be used as an access address

############################
####Some MAC consts #######

**.nic.mac.llDataHeaderLengthBits = 16 bits
**.nic.mac.llIFSDeviation = 0.0000002 s
**.nic.mac.llIFS = 0.000150 s
**.nic.mac.llmaxAdvPDUduration = 0.000376s #maximum duration of a advertisement
channel packet
**.nic.mac.txPower = 1mW #0dbm - desired TX power
**.nic.mac.slave_beacon_ReplyPolicy = "ondata" # "ondata" or "always" (what shall
slave do when it receives a beacon with no MD and it does not have any data to
send?)

############################
####Output data ###########

**.vector-recording=true

**.netwl.stats = false
**.appl.stats = false
**.tranl.stats = false
**.mobility.stats = false

**.LogCurrent=true
**.LogFrequency=true
**.LogRadioState=true
**.LogQuerryLgth=true
**.LogBLEConnection=true



############################
###test/debug stuff ######
**.TST_NoRandomAdvInt=false
**.TST_ForceHop=-1 #if >=0 will be used
```

```
**.coreDebug = false
**.debug = false

##############################################################################
##############################################################################

[Config Channel_Hopping_Chart]
description="Channel Hopping chart"

**.NumMasters = ${1}
**.NumSlaves = ${1}

**master[*].nic.mac.Init_DataQueryLgth = 0 byte #initial nuber of bytes in the LL
TX data buffer
**slave[*].nic.mac.Init_DataQueryLgth = 0 byte       #initial nuber of bytes in
the LL data buffer
**nic.mac.advertisingAddr = -1 #-1 - advertising address equals to index
**nic.mac.transmitWindowOffset = 0
**nic.mac.transmitWindowSize = 1
**nic.mac.connAccessAddress = 777


**NwkStats[*].StopSimulationOnNoData = false #stop simulation once the LL TX data
buffer of all nodes is empty

**master[*].netwl.NWK_DEBUG_ROLE = 2 #initiator/master
**slave[*].netwl.NWK_DEBUG_ROLE = 1 #advertiser/slave

#**netwl.connDataChannelMap_0to7 = 0xFF
#**netwl.connDataChannelMap_8to15 = 0xFF
#**netwl.connDataChannelMap_16to23 = 0xFF
#**netwl.connDataChannelMap_24to31 = 0xFF
#**netwl.connDataChannelMap_32to39 = 0x1F

**master[0].netwl.connDataChannelMap_0to7 = 0x03
**master[0].netwl.connDataChannelMap_8to15 = 0x00
**master[0].netwl.connDataChannelMap_16to23 = 0x00
**master[0].netwl.connDataChannelMap_24to31 = 0x00
**master[0].netwl.connDataChannelMap_32to39 = 0x00

**master[1].netwl.connDataChannelMap_0to7 = 0x03
**master[1].netwl.connDataChannelMap_8to15 = 0x00
**master[1].netwl.connDataChannelMap_16to23 = 0x00
**master[1].netwl.connDataChannelMap_24to31 = 0x00
**master[1].netwl.connDataChannelMap_32to39 = 0x00

**netwl.connAdvertiseChannelMap_0to7 = 0x00
**netwl.connAdvertiseChannelMap_8to15 = 0x00
**netwl.connAdvertiseChannelMap_16to23 = 0x00
**netwl.connAdvertiseChannelMap_24to31 = 0x00
**netwl.connAdvertiseChannelMap_32to39 = 0xE0 #default 0xE0

**netwl.advInterval = 160
```

```
**netwl.scanInterval = 160
**netwl.scanWindow = 160
**netwl.connInterval = ${connIntervalVal=800}#in 1.25 ms units
**netwl.connLatency = 0
**netwl.supervision_Timeout = ${connIntervalVal} #8*connInterval

**netwl.nodeStartupDelay = 0 s

**master[*].netwl.initialData = 0 byte
**slave[*].netwl.initialData = 40 byte


**netwl.useSuggestedFastIntervalSturtupMechanism = true



##############################################################################
##############################################################################

[Config Test_StartWithMinimum]
description="Test_New_Startup_Sequence"

#**.vector-recording=true
**.vector-recording=false
repeat = 25 #25
seed-set=${repetition}


**.NumMasters = ${NumMasters=1, 2, 4, 8, 16, 32, 64, 128, 256, 512}
#**.NumMasters = ${NumMasters=64}
**.NumSlaves = ${NumMasters}

**master[*].nic.mac.Init_DataQueryLgth = 0 byte #initial nuber of bytes in the LL
TX data buffer
**slave[*].nic.mac.Init_DataQueryLgth = 0 byte       #initial nuber of bytes in
the LL data buffer
**nic.mac.advertisingAddr = -1 #-1 - advertising address equals to index
**nic.mac.transmitWindowOffset = 0
**nic.mac.transmitWindowSize = 1
**nic.mac.connAccessAddress = 777

**NwkStats[*].StopSimulationOnNoData = true #stop simulation once the LL TX data
buffer of all nodes is empty

**master[*].netwl.NWK_DEBUG_ROLE = 2 #initiator/master
**slave[*].netwl.NWK_DEBUG_ROLE = 1 #advertiser/slave

**netwl.connDataChannelMap_0to7 =       ${freqmap     =      255, 255, 15}
**netwl.connDataChannelMap_8to15 =      ${                          255, 255, 0 !
freqmap}
**netwl.connDataChannelMap_16to23 = ${                      255, 0, 0 ! freqmap}
**netwl.connDataChannelMap_24to31 = ${                      255, 0, 0 ! freqmap}
**netwl.connDataChannelMap_32to39 = ${                      15,  0, 0 ! freqmap}
```

```
**netwl.connAdvertiseChannelMap_0to7 = 0x00
**netwl.connAdvertiseChannelMap_8to15 = 0x00
**netwl.connAdvertiseChannelMap_16to23 = 0x00
**netwl.connAdvertiseChannelMap_24to31 = 0x00
**netwl.connAdvertiseChannelMap_32to39 = 0xE0 #default 0xE0

**netwl.advInterval = 160
**netwl.scanInterval = 160
**netwl.scanWindow = 160
**netwl.connInterval = ${connIntervalVal=8, 80, 800}#in 1.25 ms units
**netwl.connLatency = 0
**netwl.supervision_Timeout = ${connIntervalVal} #8*connInterval

**netwl.nodeStartupDelay = 0 s

**master[*].netwl.initialData = 0 byte
**slave[*].netwl.initialData = 1000 byte


**netwl.useSuggestedFastIntervalSturtupMechanism = ${true, false}

##############################################################################
##############################################################################

[Config DEBUG]
description="DEBUG"

**.vector-recording=true
#**.vector-recording=false
repeat = 25 #25S
seed-set=${repetition}


**.NumMasters = ${NumMasters=1} #16
**.NumSlaves = ${NumMasters}

**master[*].nic.mac.Init_DataQueryLgth = 0 byte #initial nuber of bytes in the LL
TX data buffer
**slave[*].nic.mac.Init_DataQueryLgth = 0 byte       #initial nuber of bytes in
the LL data buffer
**nic.mac.advertisingAddr = -1 #-1 - advertising address equals to index
**nic.mac.transmitWindowOffset = 0
**nic.mac.transmitWindowSize = 1
**nic.mac.connAccessAddress = 777

**NwkStats[*].StopSimulationOnNoData = true #stop simulation once the LL TX data
buffer of all nodes is empty

**master[*].netwl.NWK_DEBUG_ROLE = 2 #initiator/master
**slave[*].netwl.NWK_DEBUG_ROLE = 1 #advertiser/slave

**netwl.connDataChannelMap_0to7 =       ${15}
```

```
**netwl.connDataChannelMap_8to15 =      ${0}
**netwl.connDataChannelMap_16to23 = ${0}
**netwl.connDataChannelMap_24to31 = ${0}
**netwl.connDataChannelMap_32to39 = ${0}

**netwl.connAdvertiseChannelMap_0to7 = 0x00
**netwl.connAdvertiseChannelMap_8to15 = 0x00
**netwl.connAdvertiseChannelMap_16to23 = 0x00
**netwl.connAdvertiseChannelMap_24to31 = 0x00
**netwl.connAdvertiseChannelMap_32to39 = 0xE0 #default 0xE0

**netwl.advInterval = 160
**netwl.scanInterval = 160
**netwl.scanWindow = 160
**netwl.connInterval = 8 #in 1.25 ms units
**netwl.connLatency = 0
**netwl.supervision_Timeout = 10 #8*connInterval

**netwl.nodeStartupDelay = 0 s

**master[*].netwl.initialData = 2000 byte
**slave[*].netwl.initialData = 0 byte


#**netwl.useSuggestedFastIntervalSturtupMechanism = false
#**nic.mac.TST_CollisionMonitoringMechanism = false
**netwl.useSuggestedFastIntervalSturtupMechanism = ${false,true}
**nic.mac.TST_CollisionMonitoringMechanism = ${false,true}

**nic.mac.macBufferSize = 5000 byte
**nic.mac.TST_CMM_DetectionPolicy = "DropTimeInRangeRelative"
**nic.mac.TST_CMM_NumPoints = 3
**nic.mac.TST_CMM_Par1 = 0.5
**nic.mac.TST_CMM_Par2 = 0.1
**nic.mac.TST_CMM_Offset_Const_Relative = 1
**nic.mac.TST_CMM_Offset_Random_Relative = 1


##########################################################################################
##########################################################################################

[Config Expirement]
description="Expirement"

**.vector-recording=false
#**.vector-recording=false
repeat = 25 #25S
seed-set=${repetition}


**.NumMasters = ${NumMasters=1, 2, 4, 8, 16, 32, 64}
**.NumSlaves = ${NumMasters}
```

```
**master[*].nic.mac.Init_DataQueryLgth = 0 byte #initial nuber of bytes in the LL
TX data buffer
**slave[*].nic.mac.Init_DataQueryLgth = 0 byte       #initial nuber of bytes in
the LL data buffer
**nic.mac.advertisingAddr = -1 #-1 - advertising address equals to index
**nic.mac.transmitWindowOffset = 0
**nic.mac.transmitWindowSize = 1
**nic.mac.connAccessAddress = 777

**NwkStats[*].StopSimulationOnNoData = true #stop simulation once the LL TX data
buffer of all nodes is empty

**master[*].netwl.NWK_DEBUG_ROLE = 2 #initiator/master
**slave[*].netwl.NWK_DEBUG_ROLE = 1 #advertiser/slave

**netwl.connDataChannelMap_0to7 =       ${freqmap     =      255, 255, 15}
**netwl.connDataChannelMap_8to15 =      ${                          255, 255, 0 !
freqmap}
**netwl.connDataChannelMap_16to23 = ${                      255, 0, 0 ! freqmap}
**netwl.connDataChannelMap_24to31 = ${                      255, 0, 0 ! freqmap}
**netwl.connDataChannelMap_32to39 = ${                      15,  0, 0 ! freqmap}

**netwl.connAdvertiseChannelMap_0to7 = 0x00
**netwl.connAdvertiseChannelMap_8to15 = 0x00
**netwl.connAdvertiseChannelMap_16to23 = 0x00
**netwl.connAdvertiseChannelMap_24to31 = 0x00
**netwl.connAdvertiseChannelMap_32to39 = 0xE0 #default 0xE0

**netwl.advInterval = 160
**netwl.scanInterval = 160
**netwl.scanWindow = 160
**netwl.connInterval = ${connIntervalVal=10, 80, 800}#in 1.25 ms units
**netwl.connLatency = 0
**netwl.supervision_Timeout = ${connIntervalVal} #8*connInterval

**netwl.nodeStartupDelay = 0 s

**master[*].netwl.initialData = 2000 byte
**slave[*].netwl.initialData = 0 byte

#**netwl.useSuggestedFastIntervalSturtupMechanism = false
#**nic.mac.TST_CollisionMonitoringMechanism = false
**netwl.useSuggestedFastIntervalSturtupMechanism = ${ACE = false,true}
**nic.mac.TST_CollisionMonitoringMechanism = ${CM = false,true}

**nic.mac.macBufferSize = 5000 byte
**nic.mac.TST_CMM_DetectionPolicy = "DropTimeInRangeRelative"
**nic.mac.TST_CMM_NumPoints = 3
**nic.mac.TST_CMM_Par1 = 0.5
**nic.mac.TST_CMM_Par2 = 0.1
**nic.mac.TST_CMM_Offset_Const_Relative = 1
**nic.mac.TST_CMM_Offset_Random_Relative = 1
```

TESTBLE.INI

```
[General]
experiment-label = "BLEV2_TEST"
network = BLEV2_testnwk
**.vector-recording = true

############################################################
#################### Display ###############################
############################################################
## BLEV2_testnwk.ned -> BaseNetwork.ned

**.playgroundSizeX = 1000 m # x size of the area the nodes are in (in meters)
**.playgroundSizeY = 1000 m # y size of the area the nodes are in (in meters)
**.playgroundSizeZ = 1000 m # z size of the area the nodes are in (in meters)


############################################################
#################### Battery ###############################
############################################################
## BLEV2_testmwk.ned -> Host_BLEV2testDevice.ned -> SimpleBatteryCurrentChart.ned
-> SimpleBattery.ned

**.battery.nominal = 2000 mAh  # nominal battery capacity ----- Si es mayor o
igual que "capacity", no afecta al consumo. Pero cada 1 mAh que se aleja por
abajo
**.battery.capacity = 2000 mAh # battery capacity         ----- del valor de
"capacity", el consumo aumenta en 11.100 mA -> consumo = 11100*(capacity-nominal)
**.battery.resolution = 30 s   # capacity is updated at least every resolution
time                                                       ----- No afecta al
consumo
**.battery.publishDelta = 0    # (0..1): capacity is published each time it is
observed to have changed by publishDelta * nominal_capacity ----- No afecta al
consumo
**.battery.publishTime = 30 s  # if > 0, capacity is published to the BB each
publishTime interval                                        ----- No afecta al
consumo
**.battery.voltage = 3 V       # nominal voltage          ----- Proporcional al
consumo, si el voltaje se dubplica, también lo hará el consumo


############################################################
#################### Current consumption ###################
############################################################
## Modules -> NicBLEV2.ned -> WirelessNicBattery.ned

**.nic.phyType = "org.mixim.modules.phy.PhyLayerBattery" #
----- No afecta al consumo
**.nic.sleepCurrent = 0.0235 mA    #CC2540 datasheet (SWRS084F) p.4
----- Al aumentar esta corriente, aumenta el consumo en el nodo y en el esclavo
```

```
**.nic.rxCurrent = 22.1 mA          #CC2540 datasheet (SWRS084F) p.5 (High-gain
mode) ----- Al aumentar esta corriente, aumentan los tres consumos: nodo, esclavo
y maestro
**.nic.decodingCurrentDelta = 0 mA #
----- No afecta al consumo
**.nic.txCurrent = 27 mA            #CC2540 datasheet (SWRS084F) p.6 for 0dbm
----- Al aumentar esta corriente, aumenta el consumo en el nodo y en el esclavo
**.nic.setupRxCurrent = 11.05 mA    #just half of rxCurrent
----- Al aumentar esta corriente, aumenta el consumo en el nodo
**.nic.setupTxCurrent = 13.5 mA     #just half of txCurrent
----- Al aumentar esta corriente, aumenta el consumo en el nodo y en el esclavo
**.nic.rxTxCurrent = 13.5 mA        #just half of txCurrent
----- Al aumentar esta corriente, aumenta el consumo en el nodo y en el esclavo
**.nic.txRxCurrent = 11.05 mA       #just half of rxCurrent
----- Al aumentar esta corriente, aumenta el consumo en el nodo y en el esclavo


###############################################################
#################### Switching times ##########################
###############################################################
## Modules -> BLEMacV2.ned

**.mac.Time_llSLEEPtoTX = 0.000004 s              # ----- Al aumentar este
tiempo, aumenta el consumo en el esclavo
**.mac.Time_llSLEEPtoRX = 0.000004 s              # ----- Al aumentar este
tiempo, aumenta el consumo en el esclavo
**.mac.Time_llTXtoRX = 0.000050 s                 # ----- No afecta al consumo
**.mac.Time_llRXtoTX = 0.000150 s                 # ----- Al aumentar este
tiempo, disminuye el consumo en el esclavo y no puede ser < 150
**.mac.Time_llTXtoSLEEP = 0 s                     # ----- No afecta al consumo
**.mac.Time_llRXtoSLEEP = 0 s                     # ----- No afecta al consumo
**.mac.TST_stopSimulation_ConnectionBreak = false # ----- No afecta al consumo


###############################################################
#################### Channel parameters #######################
###############################################################
## BLEV2_testnwk.ned -> ConnectionManager.ned

**.connectionManager.sendDirect = false           # send directly to the node or
create separate gates for every connection ----- No afecta al consumo
**.connectionManager.pMax = 10 mW                 # maximum sending power used
for this network [mW]                          ----- No afecta al consumo
**.connectionManager.sat = -90 dBm                # minimum signal attenuation
threshold [dBm]                                ----- No afecta al consumo
**.connectionManager.alpha = 2.0                  # minimum path loss
coefficient                                       ----- No afecta al consumo
**.connectionManager.carrierFrequency = 2.45e+9 Hz # minimum carrier frequency of
the channel [Hz]                               ----- No afecta al consumo


###############################################################
#################### Phy Layer parameters #####################
```

```
############################################################
## Modules -> NicBLEV2.ned -> IWirelessPhy.ned

**.nic.phy.usePropagationDelay = false           # should transmission delay be
simulated?                                  ----- No afecta al consumo
**.nic.phy.thermalNoise = -100 dBm               # the strength of the thermal
noise [dBm]                                  ----- No afecta al consumo
**.nic.phy.useThermalNoise = true                # should thermal noise be
considered?                                    ----- No afecta al consumo
**.nic.phy.analogueModels = xmldoc("config.xml") # specification of the analogue
models to use and their parameters       ----- No afecta al consumo
**.nic.phy.decider = xmldoc("config.xml")        # specification of the decider
to use and its parameters                  ----- No afecta al consumo
**.nic.phy.initialRadioState = 2                 # state the radio is initially
in (0=RX, 1=TX, 2=Sleep)                   ----- En 0 no funciona y en 1 cambian
varias cosas
**.nic.phy.sensitivity = -93 dBm                 # the sensitivity of the
physical layer [dBm]                          ----- No afecta al consumo
**.nic.phy.maxTXPower = 5.0 mW                   # the maximum transimission
power of the physical layer [mW]             ----- No afecta al consumo
**.nic.phy.nbRadioChannels = 40                   # number of available radio
channels. Defaults to single channel radio. ----- No afecta al consumo y no puede
ser < 40


############################################################
#################### Mobility ##############################
############################################################
## BLEV2_testnwk.ned -> Host_BLEV2testDevice.ned -> Host_BLEV2.ned ->
WirelessNodeBatteryCurrentChart.ned -> WirelessNode.ned

**.mobilityType = "StationaryMobility" # type of the mobility module ----- No
afecta al consumo


############################################################
#################### NWK & APP pars ########################
############################################################
## BLEV2_testnwk.ned -> Host_BLEV2testDevice.ned -> Host_BLEV2.ned ->
WirelessNodeBatteryCurrentChart.ned -> WirelessNode.ned

**.networkType = "BLE_BasicNwk" # type of the network layer ----- No afecta al
consumo

## BLEV2_testnwk.ned -> Host_BLEV2testDevice.ned -> IBaseNetwLayer.ned

**.netwl.headerLength = 0 bit # ----- No afecta al consumo

## BLEV2_testnwk.ned -> Host_BLEV2testDevice.ned -> IBaseApplLayer.ned

**.appl.headerLength = 0 byte # ----- No afecta al consumo
```

```
############################################################
#################### Some MAC consts ########################
############################################################
## Modules -> BLEMacV2.ned

**.nic.mac.llDataHeaderLengthBits = 16 bits     #
----- No afecta al consumo
**.nic.mac.llIFSDeviation = 0.0000002 s         #
----- No afecta al consumo
**.nic.mac.llIFS = 0.000150 s                   #
----- No afecta al consumo
**.nic.mac.llmaxAdvPDUduration = 0.000376 s     # maximum duration of a
advertisement channel packet ----- No afecta al consumo
**.nic.mac.txPower = 1 mW                       # 0dbm - desired TX power
----- No afecta al consumo
**.nic.mac.slave_beacon_ReplyPolicy = "ondata" # "ondata" or "always"
----- No afecta al consumo


############################################################
#################### Output data ############################
############################################################
## BLEV2_testnwk.ned -> Host_BLEV2testDevice.ned

**.LogCurrent = true        # ----- El consumo en el nodo se dispara si se pone en
false
**.LogFrequency = true      # ----- No afecta al consumo
**.LogRadioState = true     # ----- No afecta al consumo
**.LogQuerryLgth = true     # ----- No afecta al consumo
**.LogBLEConnection = true # ----- No afecta al consumo

## BLEV2_testnwk.ned -> Host_BLEV2testDevice.ned -> IBaseNetwLayer.ned

**.netwl.stats = false # ----- No afecta al consumo
**.tranl.stats = false # ----- No afecta al consumo

## BLEV2_testnwk.ned -> Host_BLEV2testDevice.ned -> IBaseApplLayer.ned

**.appl.stats = false # ----- No afecta al consumo


############################################################
#################### Test/debug stuff #######################
############################################################
## Modules -> BLEMacV2.ned

**.TST_NoRandomAdvInt = false # true = do not use random component of AdvInterval
# ----- En true, el consumo en el maestro disminuye levemente y en el esclavo
aumenta
**.TST_ForceHop = -1          # if >=0 will be used
# ----- No afecta al consumo
**.coreDebug = false
# ----- No afecta al consumo
```

```
**.debug = false
# ----- No afecta al consumo


##############################################################
################### Devices ##################################
##############################################################
## BLEV2_testnwk.ned

**.NumMasters = ${1} # ----- Si aumenta el número de maestros, disminuye
ligeramente el consumo en el nodo y en el esclavo y aumenta el consumo en los
maestros
**.NumSlaves = ${1}  # ----- Si aumenta el número de esclavos, aumentan
ligeramente el consumo en el nodo y en el maestro y disminuye el consumo en los
esclavos

## Modules -> BLEMacV2.ned

**master[*].nic.mac.Init_DataQueryLgth = 0 byte # initial nuber of bytes in the
LL TX data buffer ----- Debe de ser 0
**slave[*].nic.mac.Init_DataQueryLgth = 0 byte       # initial nuber of bytes in
the LL data buffer    ----- Debe de ser 0
**.mac.advertisingAddr = -1                       # -1 - advertising address equals
to index        ----- No afecta al consumo
**.mac.transmitWindowOffset = 0                   #
----- No afecta al consumo
**.mac.transmitWindowSize = 1                     #
----- No afecta al consumo
**.mac.connAccessAddress = 777                    #
----- No afecta al consumo

## BLEV2_testnwk.ned -> BLETestNetworkStatistics.ned

**NwkStats[*].StopSimulationOnNoData = false # stop simulation once the LL TX
data buffer of all nodes is empty ----- No afecta al consumo

## Modules -> BLE_BasicNwk.ned

**master[*].netwl.NWK_DEBUG_ROLE = 2 # initiator/master ----- No se debe cambiar
el rol, el maestro siempre será "initiator"
**slave[*].netwl.NWK_DEBUG_ROLE = 1  # advertiser/slave ----- No se debe cambiar
el rol, el esclavo siempre será "advertiser"

**master[*].netwl.connDataChannelMap_0to7 = 0x03   # ----- Ninguno de estos cinco
parámetros afecta la consumo
**master[*].netwl.connDataChannelMap_8to15 = 0x00  # -----
**master[*].netwl.connDataChannelMap_16to23 = 0x00 # -----
**master[*].netwl.connDataChannelMap_24to31 = 0x00 # -----
**master[*].netwl.connDataChannelMap_32to39 = 0x00 # -----

**netwl.connAdvertiseChannelMap_0to7 = 0x00   #              ----- No se debe
cambiar, en BLE siempre se utilizan los cananles 37, 38 y 39.
```

```
**netwl.connAdvertiseChannelMap_8to15 = 0x00  #              ----- En el caso de
que se aumente el número de canales (empezando siempre desde
**netwl.connAdvertiseChannelMap_16to23 = 0x00 #              ----- el canal más
alto, el 39), aumentarán todos los consumos: En el nodo, en el
**netwl.connAdvertiseChannelMap_24to31 = 0x00 #              ----- dispositivo
maestro y en el esclavo.
**netwl.connAdvertiseChannelMap_32to39 = 0xE0 # default 0xE0 -----

**netwl.advInterval = 160                          # in 0.625 ms units
----- Al aumentar el Advertising interval, disminuye mucho el consumo en el
esclavo
**netwl.scanInterval = 160                         # in 0.625 ms units
----- Casi no afecta al consumo
**netwl.scanWindow = 160                           # in 0.625 ms units
----- Casi no afecta al consumo
**netwl.connInterval = ${connIntervalVal=800}     # in 1.25 ms units
----- No afecta al consumo
**netwl.connLatency = 0                            # SHOULD BE 0
----- No afecta al consumo
**netwl.supervision_Timeout = ${connIntervalVal} # in 10 ms units, 8*connInterval
----- No afecta al consumo

**netwl.nodeStartupDelay = 0 s # ----- Si se aumenta, disminuye ligeramente el
consumo en el dispositivo maestro y en el eclavo

**master[*].netwl.initialData = 0 byte                  #
----- No afecta al consumo
**slave[*].netwl.initialData = 40 byte                  #
----- No afecta al consumo
**netwl.useSuggestedFastIntervalSturtupMechanism = true # activate startup with
fast intervals ----- No afecta al consumo
```

TRAZA1SEGUNDO.TXT

```
Initializing channel BLEV2_testnwk.master[0].nic.upperControlOut.channel, stage 0
Initializing channel BLEV2_testnwk.master[0].netwl.lowerControlOut.channel, stage
0
Initializing channel BLEV2_testnwk.master[0].tranl.upperControlOut.channel, stage
0
Initializing channel BLEV2_testnwk.master[0].appl.lowerControlOut.channel, stage
0
Initializing channel BLEV2_testnwk.master[0].nic.mac.upperLayerOut.channel, stage
0
Initializing channel BLEV2_testnwk.master[0].nic.mac.upperControlOut.channel,
stage 0
Initializing channel BLEV2_testnwk.master[0].nic.mac.lowerLayerOut.channel, stage
0
Initializing channel BLEV2_testnwk.master[0].nic.mac.lowerControlOut.channel,
stage 0
Initializing channel BLEV2_testnwk.master[0].nic.phy.upperLayerOut.channel, stage
0
Initializing channel BLEV2_testnwk.master[0].nic.phy.upperControlOut.channel,
stage 0
Initializing channel BLEV2_testnwk.master[0].nic.upperLayerIn.channel, stage 0
Initializing channel BLEV2_testnwk.master[0].nic.upperControlIn.channel, stage 0
Initializing channel BLEV2_testnwk.slave[0].nic.upperControlOut.channel, stage 0
Initializing channel BLEV2_testnwk.slave[0].netwl.lowerControlOut.channel, stage
0
Initializing channel BLEV2_testnwk.slave[0].tranl.upperControlOut.channel, stage
0
Initializing channel BLEV2_testnwk.slave[0].appl.lowerControlOut.channel, stage 0
Initializing channel BLEV2_testnwk.slave[0].nic.mac.upperLayerOut.channel, stage
0
Initializing channel BLEV2_testnwk.slave[0].nic.mac.upperControlOut.channel,
stage 0
Initializing channel BLEV2_testnwk.slave[0].nic.mac.lowerLayerOut.channel, stage
0
Initializing channel BLEV2_testnwk.slave[0].nic.mac.lowerControlOut.channel,
stage 0
Initializing channel BLEV2_testnwk.slave[0].nic.phy.upperLayerOut.channel, stage
0
Initializing channel BLEV2_testnwk.slave[0].nic.phy.upperControlOut.channel,
stage 0
Initializing channel BLEV2_testnwk.slave[0].nic.upperLayerIn.channel, stage 0
Initializing channel BLEV2_testnwk.slave[0].nic.upperControlIn.channel, stage 0
Initializing module BLEV2_testnwk, stage 0
Initializing module BLEV2_testnwk.connectionManager, stage 0
Initializing module BLEV2_testnwk.world, stage 0
Initializing module BLEV2_testnwk.master[0], stage 0
Initializing module BLEV2_testnwk.master[0].arp, stage 0
Initializing module BLEV2_testnwk.master[0].mobility, stage 0
initializing MobilityBase stage 0
Initializing module BLEV2_testnwk.master[0].nic, stage 0
Initializing module BLEV2_testnwk.master[0].nic.mac, stage 0
```

```
BLEMacV2 init stage: 0
Initializing module BLEV2_testnwk.master[0].nic.phy, stage 0
Initializing module BLEV2_testnwk.master[0].netwl, stage 0
Node: master[0] Initializing  Nwk Layer: BLE_BasicNwk
Initializing module BLEV2_testnwk.master[0].tranl, stage 0
Initializing module BLEV2_testnwk.master[0].appl, stage 0
Initializing module BLEV2_testnwk.master[0].batteryStats, stage 0
Initializing module BLEV2_testnwk.master[0].battery, stage 0
SimpleBattery_CurrentChart::initialize
Initializing module BLEV2_testnwk.master[0].nodeStat_Consumption, stage 0
Stat_ConsumptionChart: LogActive=1
Initializing module BLEV2_testnwk.master[0].nodeStat_Frequency, stage 0
Stat_FrequencyHops: LogActive=1
Initializing module BLEV2_testnwk.master[0].nodeStat_RadioState, stage 0
Stat_RadioState: LogActive=1
Initializing module BLEV2_testnwk.master[0].nodeStat_QuerryLength, stage 0
Stat_QuerryData: LogActive=1
Initializing module BLEV2_testnwk.master[0].nodeStat_BLEConnection, stage 0
Stat_BLEConnection: LogActive=1
Initializing module BLEV2_testnwk.slave[0], stage 0
Initializing module BLEV2_testnwk.slave[0].arp, stage 0
Initializing module BLEV2_testnwk.slave[0].mobility, stage 0
initializing MobilityBase stage 0
Initializing module BLEV2_testnwk.slave[0].nic, stage 0
Initializing module BLEV2_testnwk.slave[0].nic.mac, stage 0
BLEMacV2 init stage: 0
Initializing module BLEV2_testnwk.slave[0].nic.phy, stage 0
Initializing module BLEV2_testnwk.slave[0].netwl, stage 0
Node: slave[0] Initializing  Nwk Layer: BLE_BasicNwk
Initializing module BLEV2_testnwk.slave[0].tranl, stage 0
Initializing module BLEV2_testnwk.slave[0].appl, stage 0
Initializing module BLEV2_testnwk.slave[0].batteryStats, stage 0
Initializing module BLEV2_testnwk.slave[0].battery, stage 0
SimpleBattery_CurrentChart::initialize
Initializing module BLEV2_testnwk.slave[0].nodeStat_Consumption, stage 0
Stat_ConsumptionChart: LogActive=1
Initializing module BLEV2_testnwk.slave[0].nodeStat_Frequency, stage 0
Stat_FrequencyHops: LogActive=1
Initializing module BLEV2_testnwk.slave[0].nodeStat_RadioState, stage 0
Stat_RadioState: LogActive=1
Initializing module BLEV2_testnwk.slave[0].nodeStat_QuerryLength, stage 0
Stat_QuerryData: LogActive=1
Initializing module BLEV2_testnwk.slave[0].nodeStat_BLEConnection, stage 0
Stat_BLEConnection: LogActive=1
Initializing module BLEV2_testnwk.NwkStats[0], stage 0
Initializing module BLEV2_testnwk.connectionManager, stage 1
Initializing module BLEV2_testnwk.master[0].arp, stage 1
Initializing module BLEV2_testnwk.master[0].mobility, stage 1
initializing MobilityBase stage 1
initial position. x = 315 y = 172 z = 0
visual position. x = 315 y = 172 z = 0
Initializing module BLEV2_testnwk.master[0].nic.mac, stage 1
BLEMacV2 init stage: 1
```

```
Initializing module BLEV2_testnwk.master[0].nic.phy, stage 1
Initializing module BLEV2_testnwk.master[0].netwl, stage 1
Node: master[0] Initializing  Nwk Layer: BLE_BasicNwk
Initializing module BLEV2_testnwk.master[0].tranl, stage 1
Initializing module BLEV2_testnwk.master[0].appl, stage 1
Initializing module BLEV2_testnwk.master[0].batteryStats, stage 1
Initializing module BLEV2_testnwk.master[0].battery, stage 1
SimpleBattery_CurrentChart::initialize
Initializing module BLEV2_testnwk.master[0].nodeStat_Consumption, stage 1
Initializing module BLEV2_testnwk.master[0].nodeStat_Frequency, stage 1
Initializing module BLEV2_testnwk.master[0].nodeStat_RadioState, stage 1
Initializing module BLEV2_testnwk.master[0].nodeStat_QuerryLength, stage 1
Initializing module BLEV2_testnwk.master[0].nodeStat_BLEConnection, stage 1
Initializing module BLEV2_testnwk.slave[0].arp, stage 1
Initializing module BLEV2_testnwk.slave[0].mobility, stage 1
initializing MobilityBase stage 1
initial position. x = 70 y = 172 z = 0
visual position. x = 70 y = 172 z = 0
Initializing module BLEV2_testnwk.slave[0].nic.mac, stage 1
BLEMacV2 init stage: 1
Initializing module BLEV2_testnwk.slave[0].nic.phy, stage 1
Initializing module BLEV2_testnwk.slave[0].netwl, stage 1
Node: slave[0] Initializing  Nwk Layer: BLE_BasicNwk
Initializing module BLEV2_testnwk.slave[0].tranl, stage 1
Initializing module BLEV2_testnwk.slave[0].appl, stage 1
Initializing module BLEV2_testnwk.slave[0].batteryStats, stage 1
Initializing module BLEV2_testnwk.slave[0].battery, stage 1
SimpleBattery_CurrentChart::initialize
Initializing module BLEV2_testnwk.slave[0].nodeStat_Consumption, stage 1
Initializing module BLEV2_testnwk.slave[0].nodeStat_Frequency, stage 1
Initializing module BLEV2_testnwk.slave[0].nodeStat_RadioState, stage 1
Initializing module BLEV2_testnwk.slave[0].nodeStat_QuerryLength, stage 1
Initializing module BLEV2_testnwk.slave[0].nodeStat_BLEConnection, stage 1
** Event #1  T=0  BLEV2_testnwk.master[0].netwl (BLE_BasicNwk, id=10), on selfmsg
`timer-StartNode' (cMessage, id=54)
** Event #2  T=0  BLEV2_testnwk.master[0].netwl (BLE_BasicNwk, id=10), on selfmsg
`timer-GenerateNewData' (cMessage, id=55)
** Event #3  T=0  BLEV2_testnwk.slave[0].netwl (BLE_BasicNwk, id=25), on selfmsg
`timer-StartNode' (cMessage, id=60)
currentCmdAdvertisePars->Adv_Type4
** Event #4  T=0  BLEV2_testnwk.slave[0].netwl (BLE_BasicNwk, id=25), on selfmsg
`timer-GenerateNewData' (cMessage, id=61)
 BLETestNetworkStatistics on change_NumBytes. New NumBytes=40 Change: 40
** Event #5  T=0  BLEV2_testnwk.master[0].nic.mac (BLEMacV2, id=20), on
`BLE_NWKtoMAC_CMD' (cMessage, id=62)
BLEMacV2::handleUpperCommand_Adv: LESetAdvertisingParametersCommand
BLEMacV2::handleUpperCommand_Adv: New Adv_Type=4 Advertising_Interval_Max=2048
 myAdvrtPars.Adv_Type4
BLEMacV2::updateAdvertismentPkt
Adv_Type=1
AdvA=00-00-00-00-00-00
InitA=00-00-00-00-00-00
TxAdd=0
```

```
RxAdd=0
Length(bytes)=14
** Event #6  T=0  BLEV2_testnwk.master[0].nic.mac (BLEMacV2, id=20), on
`BLE_NWKtoMAC_CMD' (cMessage, id=63)
BLEMacV2::updateDataChannelMap
Printing ChannelMap:
TotalChNumber=2
Channel N=0 State=1
Channel N=1 State=1
Channel N=2 State=0
Channel N=3 State=0
Channel N=4 State=0
Channel N=5 State=0
Channel N=6 State=0
Channel N=7 State=0
Channel N=8 State=0
Channel N=9 State=0
Channel N=10 State=0
Channel N=11 State=0
Channel N=12 State=0
Channel N=13 State=0
Channel N=14 State=0
Channel N=15 State=0
Channel N=16 State=0
Channel N=17 State=0
Channel N=18 State=0
Channel N=19 State=0
Channel N=20 State=0
Channel N=21 State=0
Channel N=22 State=0
Channel N=23 State=0
Channel N=24 State=0
Channel N=25 State=0
Channel N=26 State=0
Channel N=27 State=0
Channel N=28 State=0
Channel N=29 State=0
Channel N=30 State=0
Channel N=31 State=0
Channel N=32 State=0
Channel N=33 State=0
Channel N=34 State=0
Channel N=35 State=0
Channel N=36 State=0
Channel N=37 State=0
Channel N=38 State=0
Channel N=39 State=0
** Event #7  T=0  BLEV2_testnwk.master[0].nic.mac (BLEMacV2, id=20), on
`BLE_NWKtoMAC_CMD' (cMessage, id=64)
BLEMacV2::handleUpperCommand_Initiate: printing old settings:
BLEMacV2::handleUpperCommand_Initiate: LE_Create_Connection_Command
** Event #8  T=0  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35), on
`BLE_NWKtoMAC_CMD' (cMessage, id=65)
```

```
BLEMacV2::handleUpperCommand_Adv: LESetAdvertisingParametersCommand
BLEMacV2::handleUpperCommand_Adv: New Adv_Type=4 Advertising_Interval_Max=160
 myAdvrtPars.Adv_Type4
BLEMacV2::updateAdvertismentPkt
Adv_Type=1
AdvA=00-00-00-00-00-00
InitA=00-00-00-00-00-00
TxAdd=0
RxAdd=0
Length(bytes)=14
** Event #9  T=0  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35), on
`BLE_NWKtoMAC_CMD' (cMessage, id=66)
BLEMacV2::handleUpperCommand_Adv: LESetAdvertisingDataCommand
 myAdvrtPars.Adv_Type4
BLEMacV2::updateAdvertismentPkt
Adv_Type=1
AdvA=00-00-00-00-00-00
InitA=00-00-00-00-00-00
TxAdd=0
RxAdd=0
Length(bytes)=14
** Event #10  T=0  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35), on
`BLE_NWKtoMAC_CMD' (cMessage, id=67)
BLEMacV2::handleUpperCommand_Adv: LESetAdvertiseEnableCommand
** Event #11  T=0  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35), on `{This is
a test data message}' (cPacket, id=68)
** Event #12  T=0  BLEV2_testnwk.master[0].nic.mac (BLEMacV2, id=20), on selfmsg
`event-SwitchState' (cMessage, id=18)
** Event #13  T=0  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35), on selfmsg
`event-SwitchState' (cMessage, id=43)
** Event #14  T=0  BLEV2_testnwk.master[0].nic.mac (BLEMacV2, id=20), on selfmsg
`timer-InitiatingScanIntervalTimer' (cMessage, id=3)
BLEMacV2:updateStatusInitiating on EV_INIT_INTERVAL. starting receive in channel
37
** Event #15  T=0.000004  BLEV2_testnwk.master[0].nic.phy (PhyLayerBattery,
id=21), on selfmsg `{radio switching over}' (cMessage, id=20)
** Event #16  T=0.000004  BLEV2_testnwk.master[0].nic.mac (BLEMacV2, id=20), on
`{Radio switching over}' (cMessage, id=72)
** Event #17  T=0.1  BLEV2_testnwk.master[0].nic.mac (BLEMacV2, id=20), on
selfmsg `timer-InitiatingScanIntervalTimer' (cMessage, id=3)
BLEMacV2:updateStatusInitiating on EV_INIT_INTERVAL. starting receive in channel
38
** Event #18  T=0.108472517372  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35),
on selfmsg `timer-AdvertisementEvent' (cMessage, id=25)
BLEMacV2:attachSignal_Adv. SENING ADV PDU. ADV PDU type=1 Length=112 bits, Access
Address=-1903575338 InitA=00-00-00-00-00-00
** Event #19  T=0.108476517372  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #20  T=0.108476517372  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on `ADV_DIRECT_IND' (BLE_Adv_MacPkt, id=73)
** Event #21  T=0.108476517372  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35),
on `{Radio switching over}' (cMessage, id=74)
```

```
** Event #22  T=0.108476517372  BLEV2_testnwk.master[0].nic.phy (PhyLayerBattery,
id=21), on `ADV_DIRECT_IND' (MiximAirFrame, id=75)
** Event #23  T=0.108652517372  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{transmission over}' (cMessage, id=46)
** Event #24  T=0.108652517372  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35),
on `{Transmission over}' (cMessage, id=76)
** Event #25  T=0.108652517372  BLEV2_testnwk.master[0].nic.phy (PhyLayerBattery,
id=21), on selfmsg `ADV_DIRECT_IND' (MiximAirFrame, id=75)
** Event #26  T=0.108802517372  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #27  T=0.108802517372  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35),
on `{Radio switching over}' (cMessage, id=77)
** Event #28  T=0.118326517372  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35),
on selfmsg `timer-AdvertisementNextPDU' (cMessage, id=26)
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. CurrentAdvChannel=37
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. NextAdvChannel=38
BLEMacV2:attachSignal_Adv. SENING ADV PDU. ADV PDU type=1 Length=112 bits, Access
Address=-1903575338 InitA=00-00-00-00-00-00
** Event #29  T=0.118476517372  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #30  T=0.118476517372  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on `ADV_DIRECT_IND' (BLE_Adv_MacPkt, id=78)
** Event #31  T=0.118476517372  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35),
on `{Radio switching over}' (cMessage, id=79)
** Event #32  T=0.118476517372  BLEV2_testnwk.master[0].nic.phy (PhyLayerBattery,
id=21), on `ADV_DIRECT_IND' (MiximAirFrame, id=80)
** Event #33  T=0.118652517372  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{transmission over}' (cMessage, id=46)
** Event #34  T=0.118652517372  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35),
on `{Transmission over}' (cMessage, id=81)
** Event #35  T=0.118652517372  BLEV2_testnwk.master[0].nic.phy (PhyLayerBattery,
id=21), on selfmsg `ADV_DIRECT_IND' (MiximAirFrame, id=80)
** Event #36  T=0.118802517372  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #37  T=0.118802517372  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35),
on `{Radio switching over}' (cMessage, id=82)
** Event #38  T=0.128326517372  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35),
on selfmsg `timer-AdvertisementNextPDU' (cMessage, id=26)
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. CurrentAdvChannel=38
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. NextAdvChannel=39
BLEMacV2:attachSignal_Adv. SENING ADV PDU. ADV PDU type=1 Length=112 bits, Access
Address=-1903575338 InitA=00-00-00-00-00-00
** Event #39  T=0.128476517372  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #40  T=0.128476517372  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on `ADV_DIRECT_IND' (BLE_Adv_MacPkt, id=83)
** Event #41  T=0.128476517372  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35),
on `{Radio switching over}' (cMessage, id=84)
** Event #42  T=0.128476517372  BLEV2_testnwk.master[0].nic.phy (PhyLayerBattery,
id=21), on `ADV_DIRECT_IND' (MiximAirFrame, id=85)
** Event #43  T=0.128652517372  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{transmission over}' (cMessage, id=46)
```

```
** Event #44  T=0.128652517372  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35),
on `{Transmission over}' (cMessage, id=86)
** Event #45  T=0.128652517372  BLEV2_testnwk.master[0].nic.phy (PhyLayerBattery,
id=21), on selfmsg `ADV_DIRECT_IND' (MiximAirFrame, id=85)
** Event #46  T=0.128802517372  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #47  T=0.128802517372  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35),
on `{Radio switching over}' (cMessage, id=87)
** Event #48  T=0.138326517372  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35),
on selfmsg `timer-AdvertisementNextPDU' (cMessage, id=26)
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. CurrentAdvChannel=39
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. NextAdvChannel=-1
** Event #49  T=0.138326517372  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35),
on selfmsg `timer-AdvertisementEndEvent' (cMessage, id=27)
** Event #50  T=0.2  BLEV2_testnwk.master[0].nic.mac (BLEMacV2, id=20), on
selfmsg `timer-InitiatingScanIntervalTimer' (cMessage, id=3)
BLEMacV2:updateStatusInitiating on EV_INIT_INTERVAL. starting receive in channel
39
** Event #51  T=0.214708154335  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35),
on selfmsg `timer-AdvertisementEvent' (cMessage, id=25)
BLEMacV2:attachSignal_Adv. SENING ADV PDU. ADV PDU type=1 Length=112 bits, Access
Address=-1903575338 InitA=00-00-00-00-00-00
** Event #52  T=0.214712154335  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #53  T=0.214712154335  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on `ADV_DIRECT_IND' (BLE_Adv_MacPkt, id=88)
** Event #54  T=0.214712154335  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35),
on `{Radio switching over}' (cMessage, id=89)
** Event #55  T=0.214712154335  BLEV2_testnwk.master[0].nic.phy (PhyLayerBattery,
id=21), on `ADV_DIRECT_IND' (MiximAirFrame, id=90)
** Event #56  T=0.214888154335  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{transmission over}' (cMessage, id=46)
** Event #57  T=0.214888154335  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35),
on `{Transmission over}' (cMessage, id=91)
** Event #58  T=0.214888154335  BLEV2_testnwk.master[0].nic.phy (PhyLayerBattery,
id=21), on selfmsg `ADV_DIRECT_IND' (MiximAirFrame, id=90)
** Event #59  T=0.215038154335  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #60  T=0.215038154335  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35),
on `{Radio switching over}' (cMessage, id=92)
** Event #61  T=0.224562154335  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35),
on selfmsg `timer-AdvertisementNextPDU' (cMessage, id=26)
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. CurrentAdvChannel=37
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. NextAdvChannel=38
BLEMacV2:attachSignal_Adv. SENING ADV PDU. ADV PDU type=1 Length=112 bits, Access
Address=-1903575338 InitA=00-00-00-00-00-00
** Event #62  T=0.224712154335  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #63  T=0.224712154335  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on `ADV_DIRECT_IND' (BLE_Adv_MacPkt, id=93)
** Event #64  T=0.224712154335  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35),
on `{Radio switching over}' (cMessage, id=94)
```

```
** Event #65  T=0.224712154335  BLEV2_testnwk.master[0].nic.phy (PhyLayerBattery,
id=21), on `ADV_DIRECT_IND' (MiximAirFrame, id=95)
** Event #66  T=0.224888154335  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{transmission over}' (cMessage, id=46)
** Event #67  T=0.224888154335  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35),
on `{Transmission over}' (cMessage, id=96)
** Event #68  T=0.224888154335  BLEV2_testnwk.master[0].nic.phy (PhyLayerBattery,
id=21), on selfmsg `ADV_DIRECT_IND' (MiximAirFrame, id=95)
** Event #69  T=0.225038154335  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #70  T=0.225038154335  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35),
on `{Radio switching over}' (cMessage, id=97)
** Event #71  T=0.234562154335  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35),
on selfmsg `timer-AdvertisementNextPDU' (cMessage, id=26)
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. CurrentAdvChannel=38
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. NextAdvChannel=39
BLEMacV2:attachSignal_Adv. SENING ADV PDU. ADV PDU type=1 Length=112 bits, Access
Address=-1903575338 InitA=00-00-00-00-00-00
** Event #72  T=0.234712154335  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #73  T=0.234712154335  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on `ADV_DIRECT_IND' (BLE_Adv_MacPkt, id=98)
** Event #74  T=0.234712154335  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35),
on `{Radio switching over}' (cMessage, id=99)
** Event #75  T=0.234712154335  BLEV2_testnwk.master[0].nic.phy (PhyLayerBattery,
id=21), on `ADV_DIRECT_IND' (MiximAirFrame, id=100)
** Event #76  T=0.234888154335  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{transmission over}' (cMessage, id=46)
** Event #77  T=0.234888154335  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35),
on `{Transmission over}' (cMessage, id=101)
** Event #78  T=0.234888154335  BLEV2_testnwk.master[0].nic.phy (PhyLayerBattery,
id=21), on selfmsg `ADV_DIRECT_IND' (MiximAirFrame, id=100)
** Event #79  T=0.235038154335  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #80  T=0.235038154335  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35),
on `{Radio switching over}' (cMessage, id=102)
** Event #81  T=0.244562154335  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35),
on selfmsg `timer-AdvertisementNextPDU' (cMessage, id=26)
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. CurrentAdvChannel=39
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. NextAdvChannel=-1
** Event #82  T=0.244562154335  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35),
on selfmsg `timer-AdvertisementEndEvent' (cMessage, id=27)
** Event #83  T=0.3  BLEV2_testnwk.master[0].nic.mac (BLEMacV2, id=20), on
selfmsg `timer-InitiatingScanIntervalTimer' (cMessage, id=3)
BLEMacV2:updateStatusInitiating on EV_INIT_INTERVAL. starting receive in channel
37
** Event #84  T=0.318551971417  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35),
on selfmsg `timer-AdvertisementEvent' (cMessage, id=25)
BLEMacV2:attachSignal_Adv. SENING ADV PDU. ADV PDU type=1 Length=112 bits, Access
Address=-1903575338 InitA=00-00-00-00-00-00
** Event #85  T=0.318555971417  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
```

```
** Event #86  T=0.318555971417  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on `ADV_DIRECT_IND' (BLE_Adv_MacPkt, id=103)
** Event #87  T=0.318555971417  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35),
on `{Radio switching over}' (cMessage, id=104)
** Event #88  T=0.318555971417  BLEV2_testnwk.master[0].nic.phy (PhyLayerBattery,
id=21), on `ADV_DIRECT_IND' (MiximAirFrame, id=105)
** Event #89  T=0.318731971417  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{transmission over}' (cMessage, id=46)
** Event #90  T=0.318731971417  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35),
on `{Transmission over}' (cMessage, id=106)
** Event #91  T=0.318731971417  BLEV2_testnwk.master[0].nic.phy (PhyLayerBattery,
id=21), on selfmsg `ADV_DIRECT_IND' (MiximAirFrame, id=105)
** Event #92  T=0.318881971417  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #93  T=0.318881971417  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35),
on `{Radio switching over}' (cMessage, id=107)
** Event #94  T=0.328405971417  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35),
on selfmsg `timer-AdvertisementNextPDU' (cMessage, id=26)
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. CurrentAdvChannel=37
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. NextAdvChannel=38
BLEMacV2:attachSignal_Adv. SENING ADV PDU. ADV PDU type=1 Length=112 bits, Access
Address=-1903575338 InitA=00-00-00-00-00-00
** Event #95  T=0.328555971417  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #96  T=0.328555971417  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on `ADV_DIRECT_IND' (BLE_Adv_MacPkt, id=108)
** Event #97  T=0.328555971417  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2, id=35),
on `{Radio switching over}' (cMessage, id=109)
** Event #98  T=0.328555971417  BLEV2_testnwk.master[0].nic.phy (PhyLayerBattery,
id=21), on `ADV_DIRECT_IND' (MiximAirFrame, id=110)
** Event #99  T=0.328731971417  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{transmission over}' (cMessage, id=46)
** Event #100  T=0.328731971417  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Transmission over}' (cMessage, id=111)
** Event #101  T=0.328731971417  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on selfmsg `ADV_DIRECT_IND' (MiximAirFrame, id=110)
** Event #102  T=0.328881971417  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #103  T=0.328881971417  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=112)
** Event #104  T=0.338405971417  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on selfmsg `timer-AdvertisementNextPDU' (cMessage, id=26)
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. CurrentAdvChannel=38
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. NextAdvChannel=39
BLEMacV2:attachSignal_Adv. SENING ADV PDU. ADV PDU type=1 Length=112 bits, Access
Address=-1903575338 InitA=00-00-00-00-00-00
** Event #105  T=0.338555971417  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #106  T=0.338555971417  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on `ADV_DIRECT_IND' (BLE_Adv_MacPkt, id=113)
** Event #107  T=0.338555971417  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=114)
```

```
** Event #108  T=0.338555971417  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on `ADV_DIRECT_IND' (MiximAirFrame, id=115)
** Event #109  T=0.338731971417  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{transmission over}' (cMessage, id=46)
** Event #110  T=0.338731971417  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Transmission over}' (cMessage, id=116)
** Event #111  T=0.338731971417  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on selfmsg `ADV_DIRECT_IND' (MiximAirFrame, id=115)
** Event #112  T=0.338881971417  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #113  T=0.338881971417  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=117)
** Event #114  T=0.348405971417  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on selfmsg `timer-AdvertisementNextPDU' (cMessage, id=26)
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. CurrentAdvChannel=39
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. NextAdvChannel=-1
** Event #115  T=0.348405971417  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on selfmsg `timer-AdvertisementEndEvent' (cMessage, id=27)
** Event #116  T=0.4  BLEV2_testnwk.master[0].nic.mac (BLEMacV2, id=20), on
selfmsg `timer-InitiatingScanIntervalTimer' (cMessage, id=3)
BLEMacV2:updateStatusInitiating on EV_INIT_INTERVAL. starting receive in channel
38
** Event #117  T=0.421527317469  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on selfmsg `timer-AdvertisementEvent' (cMessage, id=25)
BLEMacV2:attachSignal_Adv. SENING ADV PDU. ADV PDU type=1 Length=112 bits, Access
Address=-1903575338 InitA=00-00-00-00-00-00
** Event #118  T=0.421531317469  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #119  T=0.421531317469  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on `ADV_DIRECT_IND' (BLE_Adv_MacPkt, id=118)
** Event #120  T=0.421531317469  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=119)
** Event #121  T=0.421531317469  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on `ADV_DIRECT_IND' (MiximAirFrame, id=120)
** Event #122  T=0.421707317469  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{transmission over}' (cMessage, id=46)
** Event #123  T=0.421707317469  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Transmission over}' (cMessage, id=121)
** Event #124  T=0.421707317469  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on selfmsg `ADV_DIRECT_IND' (MiximAirFrame, id=120)
** Event #125  T=0.421857317469  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #126  T=0.421857317469  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=122)
** Event #127  T=0.431381317469  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on selfmsg `timer-AdvertisementNextPDU' (cMessage, id=26)
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. CurrentAdvChannel=37
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. NextAdvChannel=38
BLEMacV2:attachSignal_Adv. SENING ADV PDU. ADV PDU type=1 Length=112 bits, Access
Address=-1903575338 InitA=00-00-00-00-00-00
** Event #128  T=0.431531317469  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
```

```
** Event #129  T=0.431531317469  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on `ADV_DIRECT_IND' (BLE_Adv_MacPkt, id=123)
** Event #130  T=0.431531317469  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=124)
** Event #131  T=0.431531317469  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on `ADV_DIRECT_IND' (MiximAirFrame, id=125)
** Event #132  T=0.431707317469  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{transmission over}' (cMessage, id=46)
** Event #133  T=0.431707317469  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Transmission over}' (cMessage, id=126)
** Event #134  T=0.431707317469  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on selfmsg `ADV_DIRECT_IND' (MiximAirFrame, id=125)
** Event #135  T=0.431857317469  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #136  T=0.431857317469  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=127)
** Event #137  T=0.441381317469  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on selfmsg `timer-AdvertisementNextPDU' (cMessage, id=26)
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. CurrentAdvChannel=38
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. NextAdvChannel=39
BLEMacV2:attachSignal_Adv. SENING ADV PDU. ADV PDU type=1 Length=112 bits, Access
Address=-1903575338 InitA=00-00-00-00-00-00
** Event #138  T=0.441531317469  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #139  T=0.441531317469  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on `ADV_DIRECT_IND' (BLE_Adv_MacPkt, id=128)
** Event #140  T=0.441531317469  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=129)
** Event #141  T=0.441531317469  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on `ADV_DIRECT_IND' (MiximAirFrame, id=130)
** Event #142  T=0.441707317469  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{transmission over}' (cMessage, id=46)
** Event #143  T=0.441707317469  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Transmission over}' (cMessage, id=131)
** Event #144  T=0.441707317469  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on selfmsg `ADV_DIRECT_IND' (MiximAirFrame, id=130)
** Event #145  T=0.441857317469  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #146  T=0.441857317469  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=132)
** Event #147  T=0.451381317469  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on selfmsg `timer-AdvertisementNextPDU' (cMessage, id=26)
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. CurrentAdvChannel=39
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. NextAdvChannel=-1
** Event #148  T=0.451381317469  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on selfmsg `timer-AdvertisementEndEvent' (cMessage, id=27)
** Event #149  T=0.5  BLEV2_testnwk.master[0].nic.mac (BLEMacV2, id=20), on
selfmsg `timer-InitiatingScanIntervalTimer' (cMessage, id=3)
BLEMacV2:updateStatusInitiating on EV_INIT_INTERVAL. starting receive in channel
39
** Event #150  T=0.522094447226  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on selfmsg `timer-AdvertisementEvent' (cMessage, id=25)
```

```
BLEMacV2:attachSignal_Adv. SENING ADV PDU. ADV PDU type=1 Length=112 bits, Access
Address=-1903575338 InitA=00-00-00-00-00-00
** Event #151  T=0.522098447226  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #152  T=0.522098447226  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on `ADV_DIRECT_IND' (BLE_Adv_MacPkt, id=133)
** Event #153  T=0.522098447226  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=134)
** Event #154  T=0.522098447226  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on `ADV_DIRECT_IND' (MiximAirFrame, id=135)
** Event #155  T=0.522274447226  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{transmission over}' (cMessage, id=46)
** Event #156  T=0.522274447226  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Transmission over}' (cMessage, id=136)
** Event #157  T=0.522274447226  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on selfmsg `ADV_DIRECT_IND' (MiximAirFrame, id=135)
** Event #158  T=0.522424447226  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #159  T=0.522424447226  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=137)
** Event #160  T=0.531948447226  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on selfmsg `timer-AdvertisementNextPDU' (cMessage, id=26)
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. CurrentAdvChannel=37
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. NextAdvChannel=38
BLEMacV2:attachSignal_Adv. SENING ADV PDU. ADV PDU type=1 Length=112 bits, Access
Address=-1903575338 InitA=00-00-00-00-00-00
** Event #161  T=0.532098447226  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #162  T=0.532098447226  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on `ADV_DIRECT_IND' (BLE_Adv_MacPkt, id=138)
** Event #163  T=0.532098447226  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=139)
** Event #164  T=0.532098447226  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on `ADV_DIRECT_IND' (MiximAirFrame, id=140)
** Event #165  T=0.532274447226  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{transmission over}' (cMessage, id=46)
** Event #166  T=0.532274447226  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Transmission over}' (cMessage, id=141)
** Event #167  T=0.532274447226  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on selfmsg `ADV_DIRECT_IND' (MiximAirFrame, id=140)
** Event #168  T=0.532424447226  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #169  T=0.532424447226  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=142)
** Event #170  T=0.541948447226  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on selfmsg `timer-AdvertisementNextPDU' (cMessage, id=26)
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. CurrentAdvChannel=38
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. NextAdvChannel=39
BLEMacV2:attachSignal_Adv. SENING ADV PDU. ADV PDU type=1 Length=112 bits, Access
Address=-1903575338 InitA=00-00-00-00-00-00
** Event #171  T=0.542098447226  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
```

```
** Event #172  T=0.542098447226  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on `ADV_DIRECT_IND' (BLE_Adv_MacPkt, id=143)
** Event #173  T=0.542098447226  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=144)
** Event #174  T=0.542098447226  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on `ADV_DIRECT_IND' (MiximAirFrame, id=145)
** Event #175  T=0.542274447226  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{transmission over}' (cMessage, id=46)
** Event #176  T=0.542274447226  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Transmission over}' (cMessage, id=146)
** Event #177  T=0.542274447226  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on selfmsg `ADV_DIRECT_IND' (MiximAirFrame, id=145)
** Event #178  T=0.542424447226  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #179  T=0.542424447226  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=147)
** Event #180  T=0.551948447226  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on selfmsg `timer-AdvertisementNextPDU' (cMessage, id=26)
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. CurrentAdvChannel=39
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. NextAdvChannel=-1
** Event #181  T=0.551948447226  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on selfmsg `timer-AdvertisementEndEvent' (cMessage, id=27)
** Event #182  T=0.6  BLEV2_testnwk.master[0].nic.mac (BLEMacV2, id=20), on
selfmsg `timer-InitiatingScanIntervalTimer' (cMessage, id=3)
BLEMacV2:updateStatusInitiating on EV_INIT_INTERVAL. starting receive in channel
37
** Event #183  T=0.624821010171  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on selfmsg `timer-AdvertisementEvent' (cMessage, id=25)
BLEMacV2:attachSignal_Adv. SENING ADV PDU. ADV PDU type=1 Length=112 bits, Access
Address=-1903575338 InitA=00-00-00-00-00-00
** Event #184  T=0.624825010171  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #185  T=0.624825010171  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on `ADV_DIRECT_IND' (BLE_Adv_MacPkt, id=148)
** Event #186  T=0.624825010171  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=149)
** Event #187  T=0.624825010171  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on `ADV_DIRECT_IND' (MiximAirFrame, id=150)
** Event #188  T=0.625001010171  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{transmission over}' (cMessage, id=46)
** Event #189  T=0.625001010171  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Transmission over}' (cMessage, id=151)
** Event #190  T=0.625001010171  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on selfmsg `ADV_DIRECT_IND' (MiximAirFrame, id=150)
** Event #191  T=0.625151010171  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #192  T=0.625151010171  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=152)
** Event #193  T=0.634675010171  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on selfmsg `timer-AdvertisementNextPDU' (cMessage, id=26)
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. CurrentAdvChannel=37
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. NextAdvChannel=38
```

```
BLEMacV2:attachSignal_Adv. SENING ADV PDU. ADV PDU type=1 Length=112 bits, Access
Address=-1903575338 InitA=00-00-00-00-00-00
** Event #194  T=0.634825010171  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #195  T=0.634825010171  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on `ADV_DIRECT_IND' (BLE_Adv_MacPkt, id=153)
** Event #196  T=0.634825010171  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=154)
** Event #197  T=0.634825010171  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on `ADV_DIRECT_IND' (MiximAirFrame, id=155)
** Event #198  T=0.635001010171  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{transmission over}' (cMessage, id=46)
** Event #199  T=0.635001010171  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Transmission over}' (cMessage, id=156)
** Event #200  T=0.635001010171  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on selfmsg `ADV_DIRECT_IND' (MiximAirFrame, id=155)
** Event #201  T=0.635151010171  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #202  T=0.635151010171  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=157)
** Event #203  T=0.644675010171  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on selfmsg `timer-AdvertisementNextPDU' (cMessage, id=26)
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. CurrentAdvChannel=38
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. NextAdvChannel=39
BLEMacV2:attachSignal_Adv. SENING ADV PDU. ADV PDU type=1 Length=112 bits, Access
Address=-1903575338 InitA=00-00-00-00-00-00
** Event #204  T=0.644825010171  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #205  T=0.644825010171  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on `ADV_DIRECT_IND' (BLE_Adv_MacPkt, id=158)
** Event #206  T=0.644825010171  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=159)
** Event #207  T=0.644825010171  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on `ADV_DIRECT_IND' (MiximAirFrame, id=160)
** Event #208  T=0.645001010171  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{transmission over}' (cMessage, id=46)
** Event #209  T=0.645001010171  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Transmission over}' (cMessage, id=161)
** Event #210  T=0.645001010171  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on selfmsg `ADV_DIRECT_IND' (MiximAirFrame, id=160)
** Event #211  T=0.645151010171  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #212  T=0.645151010171  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=162)
** Event #213  T=0.654675010171  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on selfmsg `timer-AdvertisementNextPDU' (cMessage, id=26)
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. CurrentAdvChannel=39
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. NextAdvChannel=-1
** Event #214  T=0.654675010171  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on selfmsg `timer-AdvertisementEndEvent' (cMessage, id=27)
** Event #215  T=0.7  BLEV2_testnwk.master[0].nic.mac (BLEMacV2, id=20), on
selfmsg `timer-InitiatingScanIntervalTimer' (cMessage, id=3)
```

```
BLEMacV2:updateStatusInitiating on EV_INIT_INTERVAL. starting receive in channel
38
** Event #216  T=0.729597661287  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on selfmsg `timer-AdvertisementEvent' (cMessage, id=25)
BLEMacV2:attachSignal_Adv. SENING ADV PDU. ADV PDU type=1 Length=112 bits, Access
Address=-1903575338 InitA=00-00-00-00-00-00
** Event #217  T=0.729601661287  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #218  T=0.729601661287  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on `ADV_DIRECT_IND' (BLE_Adv_MacPkt, id=163)
** Event #219  T=0.729601661287  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=164)
** Event #220  T=0.729601661287  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on `ADV_DIRECT_IND' (MiximAirFrame, id=165)
** Event #221  T=0.729777661287  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{transmission over}' (cMessage, id=46)
** Event #222  T=0.729777661287  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Transmission over}' (cMessage, id=166)
** Event #223  T=0.729777661287  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on selfmsg `ADV_DIRECT_IND' (MiximAirFrame, id=165)
** Event #224  T=0.729927661287  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #225  T=0.729927661287  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=167)
** Event #226  T=0.739451661287  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on selfmsg `timer-AdvertisementNextPDU' (cMessage, id=26)
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. CurrentAdvChannel=37
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. NextAdvChannel=38
BLEMacV2:attachSignal_Adv. SENING ADV PDU. ADV PDU type=1 Length=112 bits, Access
Address=-1903575338 InitA=00-00-00-00-00-00
** Event #227  T=0.739601661287  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #228  T=0.739601661287  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on `ADV_DIRECT_IND' (BLE_Adv_MacPkt, id=168)
** Event #229  T=0.739601661287  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=169)
** Event #230  T=0.739601661287  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on `ADV_DIRECT_IND' (MiximAirFrame, id=170)
** Event #231  T=0.739777661287  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{transmission over}' (cMessage, id=46)
** Event #232  T=0.739777661287  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Transmission over}' (cMessage, id=171)
** Event #233  T=0.739777661287  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on selfmsg `ADV_DIRECT_IND' (MiximAirFrame, id=170)
** Event #234  T=0.739927661287  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #235  T=0.739927661287  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=172)
** Event #236  T=0.749451661287  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on selfmsg `timer-AdvertisementNextPDU' (cMessage, id=26)
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. CurrentAdvChannel=38
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. NextAdvChannel=39
```

```
BLEMacV2:attachSignal_Adv. SENING ADV PDU. ADV PDU type=1 Length=112 bits, Access
Address=-1903575338 InitA=00-00-00-00-00-00
** Event #237  T=0.749601661287  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #238  T=0.749601661287  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on `ADV_DIRECT_IND' (BLE_Adv_MacPkt, id=173)
** Event #239  T=0.749601661287  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=174)
** Event #240  T=0.749601661287  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on `ADV_DIRECT_IND' (MiximAirFrame, id=175)
** Event #241  T=0.749777661287  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{transmission over}' (cMessage, id=46)
** Event #242  T=0.749777661287  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Transmission over}' (cMessage, id=176)
** Event #243  T=0.749777661287  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on selfmsg `ADV_DIRECT_IND' (MiximAirFrame, id=175)
** Event #244  T=0.749927661287  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #245  T=0.749927661287  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=177)
** Event #246  T=0.759451661287  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on selfmsg `timer-AdvertisementNextPDU' (cMessage, id=26)
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. CurrentAdvChannel=39
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. NextAdvChannel=-1
** Event #247  T=0.759451661287  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on selfmsg `timer-AdvertisementEndEvent' (cMessage, id=27)
** Event #248  T=0.8  BLEV2_testnwk.master[0].nic.mac (BLEMacV2, id=20), on
selfmsg `timer-InitiatingScanIntervalTimer' (cMessage, id=3)
BLEMacV2:updateStatusInitiating on EV_INIT_INTERVAL. starting receive in channel
39
** Event #249  T=0.837719348551  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on selfmsg `timer-AdvertisementEvent' (cMessage, id=25)
BLEMacV2:attachSignal_Adv. SENING ADV PDU. ADV PDU type=1 Length=112 bits, Access
Address=-1903575338 InitA=00-00-00-00-00-00
** Event #250  T=0.837723348551  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #251  T=0.837723348551  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on `ADV_DIRECT_IND' (BLE_Adv_MacPkt, id=178)
** Event #252  T=0.837723348551  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=179)
** Event #253  T=0.837723348551  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on `ADV_DIRECT_IND' (MiximAirFrame, id=180)
** Event #254  T=0.837899348551  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{transmission over}' (cMessage, id=46)
** Event #255  T=0.837899348551  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Transmission over}' (cMessage, id=181)
** Event #256  T=0.837899348551  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on selfmsg `ADV_DIRECT_IND' (MiximAirFrame, id=180)
** Event #257  T=0.838049348551  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #258  T=0.838049348551  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=182)
```

```
** Event #259  T=0.847573348551  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on selfmsg `timer-AdvertisementNextPDU' (cMessage, id=26)
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. CurrentAdvChannel=37
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. NextAdvChannel=38
BLEMacV2:attachSignal_Adv. SENING ADV PDU. ADV PDU type=1 Length=112 bits, Access
Address=-1903575338 InitA=00-00-00-00-00-00
** Event #260  T=0.847723348551  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #261  T=0.847723348551  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on `ADV_DIRECT_IND' (BLE_Adv_MacPkt, id=183)
** Event #262  T=0.847723348551  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=184)
** Event #263  T=0.847723348551  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on `ADV_DIRECT_IND' (MiximAirFrame, id=185)
** Event #264  T=0.847899348551  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{transmission over}' (cMessage, id=46)
** Event #265  T=0.847899348551  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Transmission over}' (cMessage, id=186)
** Event #266  T=0.847899348551  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on selfmsg `ADV_DIRECT_IND' (MiximAirFrame, id=185)
** Event #267  T=0.848049348551  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #268  T=0.848049348551  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=187)
** Event #269  T=0.857573348551  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on selfmsg `timer-AdvertisementNextPDU' (cMessage, id=26)
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. CurrentAdvChannel=38
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. NextAdvChannel=39
BLEMacV2:attachSignal_Adv. SENING ADV PDU. ADV PDU type=1 Length=112 bits, Access
Address=-1903575338 InitA=00-00-00-00-00-00
** Event #270  T=0.857723348551  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #271  T=0.857723348551  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on `ADV_DIRECT_IND' (BLE_Adv_MacPkt, id=188)
** Event #272  T=0.857723348551  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=189)
** Event #273  T=0.857723348551  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on `ADV_DIRECT_IND' (MiximAirFrame, id=190)
** Event #274  T=0.857899348551  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{transmission over}' (cMessage, id=46)
** Event #275  T=0.857899348551  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Transmission over}' (cMessage, id=191)
** Event #276  T=0.857899348551  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on selfmsg `ADV_DIRECT_IND' (MiximAirFrame, id=190)
** Event #277  T=0.858049348551  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #278  T=0.858049348551  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=192)
** Event #279  T=0.867573348551  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on selfmsg `timer-AdvertisementNextPDU' (cMessage, id=26)
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. CurrentAdvChannel=39
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. NextAdvChannel=-1
```

```
** Event #280  T=0.867573348551  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on selfmsg `timer-AdvertisementEndEvent' (cMessage, id=27)
** Event #281  T=0.9  BLEV2_testnwk.master[0].nic.mac (BLEMacV2, id=20), on
selfmsg `timer-InitiatingScanIntervalTimer' (cMessage, id=3)
BLEMacV2:updateStatusInitiating on EV_INIT_INTERVAL. starting receive in channel
37
** Event #282  T=0.942519120265  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on selfmsg `timer-AdvertisementEvent' (cMessage, id=25)
BLEMacV2:attachSignal_Adv. SENING ADV PDU. ADV PDU type=1 Length=112 bits, Access
Address=-1903575338 InitA=00-00-00-00-00-00
** Event #283  T=0.942523120265  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #284  T=0.942523120265  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on `ADV_DIRECT_IND' (BLE_Adv_MacPkt, id=193)
** Event #285  T=0.942523120265  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=194)
** Event #286  T=0.942523120265  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on `ADV_DIRECT_IND' (MiximAirFrame, id=195)
** Event #287  T=0.942699120265  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{transmission over}' (cMessage, id=46)
** Event #288  T=0.942699120265  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Transmission over}' (cMessage, id=196)
** Event #289  T=0.942699120265  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on selfmsg `ADV_DIRECT_IND' (MiximAirFrame, id=195)
** Event #290  T=0.942849120265  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #291  T=0.942849120265  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=197)
** Event #292  T=0.952373120265  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on selfmsg `timer-AdvertisementNextPDU' (cMessage, id=26)
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. CurrentAdvChannel=37
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. NextAdvChannel=38
BLEMacV2:attachSignal_Adv. SENING ADV PDU. ADV PDU type=1 Length=112 bits, Access
Address=-1903575338 InitA=00-00-00-00-00-00
** Event #293  T=0.952523120265  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #294  T=0.952523120265  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on `ADV_DIRECT_IND' (BLE_Adv_MacPkt, id=198)
** Event #295  T=0.952523120265  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=199)
** Event #296  T=0.952523120265  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on `ADV_DIRECT_IND' (MiximAirFrame, id=200)
** Event #297  T=0.952699120265  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{transmission over}' (cMessage, id=46)
** Event #298  T=0.952699120265  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Transmission over}' (cMessage, id=201)
** Event #299  T=0.952699120265  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on selfmsg `ADV_DIRECT_IND' (MiximAirFrame, id=200)
** Event #300  T=0.952849120265  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #301  T=0.952849120265  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=202)
```

```
** Event #302  T=0.962373120265  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on selfmsg `timer-AdvertisementNextPDU' (cMessage, id=26)
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. CurrentAdvChannel=38
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. NextAdvChannel=39
BLEMacV2:attachSignal_Adv. SENING ADV PDU. ADV PDU type=1 Length=112 bits, Access
Address=-1903575338 InitA=00-00-00-00-00-00
** Event #303  T=0.962523120265  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #304  T=0.962523120265  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on `ADV_DIRECT_IND' (BLE_Adv_MacPkt, id=203)
** Event #305  T=0.962523120265  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=204)
** Event #306  T=0.962523120265  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on `ADV_DIRECT_IND' (MiximAirFrame, id=205)
** Event #307  T=0.962699120265  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{transmission over}' (cMessage, id=46)
** Event #308  T=0.962699120265  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Transmission over}' (cMessage, id=206)
** Event #309  T=0.962699120265  BLEV2_testnwk.master[0].nic.phy
(PhyLayerBattery, id=21), on selfmsg `ADV_DIRECT_IND' (MiximAirFrame, id=205)
** Event #310  T=0.962849120265  BLEV2_testnwk.slave[0].nic.phy (PhyLayerBattery,
id=36), on selfmsg `{radio switching over}' (cMessage, id=45)
** Event #311  T=0.962849120265  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on `{Radio switching over}' (cMessage, id=207)
** Event #312  T=0.972373120265  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on selfmsg `timer-AdvertisementNextPDU' (cMessage, id=26)
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. CurrentAdvChannel=39
BLEMacV2:updateStatusAdvertising on EV_ADV_NEXTPDU. NextAdvChannel=-1
** Event #313  T=0.972373120265  BLEV2_testnwk.slave[0].nic.mac (BLEMacV2,
id=35), on selfmsg `timer-AdvertisementEndEvent' (cMessage, id=27)
```

CONFIG.XML

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
	<AnalogueModels>
		<AnalogueModel type="SimplePathlossModel">
		<parameter name="alpha" type="double" value="4.0"/>
		<parameter name="carrierFrequency" type="double" value="2.450e+9"/>
	    </AnalogueModel>
	</AnalogueModels>
	<Decider type="SNRThresholdDecider">
		<!-- SNR threshold (as fraction) above which the decider consideres
a
			a signal as received correctly. -->
		<parameter name="snrThreshold" type="double"
value="1.12589254117942"/>
		<!-- RSSI (noise and signal) threshold (in mW) above which the
			channel is considered idle. If this parameter is
			ommited the sensitivity of the physical layer is
			used as threshold.-->
		<parameter name="busyThreshold" type="double" value="3.98107170553E-
9"/>
	</Decider>
</root>
```