ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

ANÁLISIS Y DOCUMENTACIÓN DE LA IMPLEMENTACIÓN DE LA CAPA FÍSICA IEEE 802.11a y 802.11p PARA REDES AD HOC EN EL SIMULADOR OMNET++

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

GISSELA KATHERINE ALOBUELA LOACHAMÍN

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

CHRISTIAN ANDRÉ PADILLA OLIVO

DIRECTOR: Ing. LUIS FELIPE URQUIZA AGUIAR, PhD.

CODIRECTOR: Ing. MARTHA CECILIA PAREDES PAREDES, PhD.

Quito, agosto 2019

AVAL

Certificamos que el presente trabajo fue desarrollado por Gissela Katherine Alobuela Loachamín y Christian André Padilla Olivo, bajo nuestra supervisión.

Ing. Luis Felipe Urquiza Aguiar, PhD.
DIRECTOR DEL TRABAJO DE TITULACIÓN

Ing. Martha Cecilia Paredes Paredes, PhD.
CODIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Nosotros, Gissela Katherine Alobuela Loachamín y Christian André Padilla Olivo, declaramos bajo juramento que el trabajo aquí descrito es de nuestra autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que hemos consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración dejamos constancia de que la Escuela Politécnica Nacional podrá hacer uso del presente trabajo según los términos estipulados en la Ley, Reglamentos y Normas vigentes.

Gissela Katherine Alobuela Loachamín	Christian André Padilla Olivo

DEDICATORIA

Dedico este Proyecto de Titulación a mis padres, Mariana y Enrique, que siempre estuvieron apoyándome. Ustedes me enseñaron el valor del trabajo arduo y con su ejemplo de lucha y constancia me motivaron a seguir superándome para obtener un mejor futuro. Muchas gracias por todo, esto es para ustedes.

Gissela Katherine Alobuela Loachamín

Dedico este trabajo a las personas más importantes de mi vida: Claudia, Julio y Kevin.

Christian André Padilla Olivo

AGRADECIMIENTO

Agradezco a Dios por haberme guiado en el transcurso de mi vida y por darme la sabiduría necesaria para superar los obstáculos que se presentaron durante el camino para culminar esta importante etapa.

A mis padres Enrique y Marina, les agradezco de todo corazón por todo el apoyo brindado y todo el esfuerzo y sacrificio realizado para darme un futuro mejor. Gracias por su buen ejemplo y por hacer de mí una mujer de bien. Por sus consejos y por alentarme a luchar por cumplir todas mis metas. Sin todo lo que han hecho por mí, no hubiera alcanzado esta meta.

A mi abuelita María Enriqueta por todos sus consejos y por ser un gran ejemplo en mi vida.

A mi abuelita María y mis abuelos Francisco y Manuel que en paz descansen, por ser un ejemplo de lucha y entrega.

A Pedro, por su apoyo incondicional y por alentarme a no rendirme. Por siempre estar a mi lado pese a cualquier circunstancia y por ser la mejor compañía.

A la Escuela Politécnica Nacional y a sus maestros, por brindarme una excelente formación académica.

Al Dr. Luis Urquiza y a la Dra. Cecilia Paredes, por haber dirigido este proyecto y por estar siempre dispuestos a ayudarnos y colaborar con el desarrollo y culminación de este. Gracias por su tiempo y paciencia.

A mi compañero de proyecto Christian, por su total compromiso en el desarrollo del proyecto. Por todo su esfuerzo y dedicación se logró finalizar este proyecto.

Gracias a todos mis amigos por acompañarme durante vida universitaria y por darme ánimos para lograr culminar con esta etapa de mi vida.

Gissela Katherine Alobuela Loachamín

AGRADECIMIENTO

Agradezco a mis padres, maestros y amigos. Mis padres porque siempre porporcionaron apoyo incondicional, esto fue de suma importancia para superar obstáculos en el camino.

Mis maestros debido a los valiosos conocimmientos impartidos durante la carrera. En especial quisiera agradecer al Dr. Luis Urquiza y Dra. Cecilia Paredes que guiaron este trabajo y cuyo esmero permitió que culminemos este proyecto.

Mis amigos de verdad porque este camino lo recorrimos como un equipo, gracias por la ayuda brindada.

Christian André Padilla Olivo

ÍNDICE DE CONTENIDO

RESUMEN

Debido a la gran cantidad de dispositivos involucrados en una red ad hoc, realizar experimentos en escenarios reales resulta costoso y requiere de mucho tiempo y esfuerzo, por lo que el uso de un simulador de red es necesario. Puesto que los protocolos de capa MAC y red han sido ampliamente estudiados y desarrollados en OMNeT++, el presente proyecto de titulación se centra en el estudio de la capa física de redes inalámbricas ad hoc IEEE 802.11 en este simulador.

De forma más precisa, nuestro estudio se enfoca en la simulación de redes ad hoc con tecnología IEEE 802.11a e IEEE 802.11p sobre OMNeT++, el cual permite simular este tipo de redes mediante los frameworks INETMANET y Veins, respectivamente. Específicamente, se realiza el análisis de la implementación de la capa física del estándar IEEE 802.11a para MANETs (Mobile Ad Hoc Networks) e IEEE 802.11p para VANETs (Vehicular Ad Hoc Networks) en dicho simulador. El análisis de la capa física se realiza a nivel de código con lo cual se describe el funcionamiento de los componentes principales de INETMANET y Veins, y se explica cómo se realiza el proceso de transmisión/recepción mediante diagramas.

Todo el estudio realizado permite establecer características implementadas y no implementadas en el simulador, con lo que podemos determinar las limitaciones de OMNeT++ y de sus Frameworks INETMANET y Veins para modelar la capa física de redes ad hoc IEEE 802.11a e IEEE 802.11p. Además, a partir de escenarios de simulación se comprobó el comportamiento del throughput y retardo usando diferentes modelos de propagación disponibles en el simulador.

PALABRAS CLAVE: MANET, VANET, OMNeT++, INETMANET, Veins.

ABSTRACT

Performing experiments in real scenarios is expensive and requires a lot of time and effort due to the large number of devices involved in an ad hoc network. Hence, the use of a simulator is necessary to test new protocols in early stages. MAC and network layer protocols have been extensively studied, for this reason, this project focuses on the study of ad hoc wireless networks physical layer in OMNeT++.

Our study focuses on IEEE 802.11a and IEEE 802.11p ad hoc networks in OMNeT ++. This application allows to simulate this type of networks using INETMANET and Veins Frameworks, respectively. Specifically, we analyze physical layer implementation of the IEEE 802.11a standard for MANETs (Mobile Ad Hoc Networks) and IEEE 802.11p for VANET (Vehicular Ad Hoc Networks) on this simulator. The physical layer analysis is performed at code level, we describe the operation of main INETMANET and Veins components and make use of diagrams to explain how the transmission/reception process is performed.

This study permits to establish which features are implemented or not in the simulator, this allows to define restraints for OMNET ++ and its frameworks INETMANET and Veins for modeling ad hoc networks at the physical layer level. In addition, we will use the included simulation scenarios to test behavior of throughput and delay using different propagation models available in the simulator.

KEYWORDS: MANET, VANET, OMNeT++, INETMANET, Veins.

1. INTRODUCCIÓN

Últimamente, las redes ad hoc han sido ampliamente estudiadas debido a que permiten la comunicación entre dispositivos en todo lugar y en cualquier momento sin utilizar infraestructura. En particular, las MANET (Mobile Ad Hoc Network) son estudiadas por la comunidad científica debido a la movilidad de sus dispositivos y el constante cambio de sus enlaces inalámbricos. De la misma manera las redes para comunicación entre vehículos e infraestructura de avenidas conocidas como redes VANET (Vehicular Ad Hoc Network) han sido de gran interés.

Los dispositivos de red de estas redes ad hoc deben estar preparados para encaminar el tráfico en una red que cambia constantemente. Además, por la gran cantidad de dispositivos involucrados en la comunicación, realizar experimentos en escenario reales resulta complicado y costoso, razón por la cual es necesario el uso de simuladores. Gracias a los simuladores desarrollados durante los últimos años, los investigadores pueden estudiar, evaluar e implementar redes sin el requerimiento de una fuerte cantidad de dinero para la implementación de mucho tiempo para la realización de pruebas.

En el área de estudio de redes ad hoc, OMNeT++ es uno de los simuladores más utilizados, el mismo que facilitará la ejecución de pruebas variando condiciones de la red que es objeto de estudio. El análisis que se realizará contempla redes ad hoc IEEE 802.11a e IEEE 802.11p; el simulador OMNeT++ cuenta con los frameworks INETMANET y Veins respectivamente para simulación de este tipo de redes. En este simulador los protocolos de capa MAC y red han sido ampliamente estudiados, mientras que la capa física muy pocas veces ha sido objeto de investigación. Por estos motivos, este proyecto se enfoca en el estudio de la implementación y modelado de las capas físicas de redes ad hoc IEEE 802.11a e IEEE 802.11p en el simulador OMNeT++.

Modelar adecuadamente la capa física permitirá garantizar la fiabilidad en los resultados y conclusiones obtenidas a través del simulador. En general, la tarea de modelar la capa física requiere de mucho tiempo y esfuerzo debido a que la simulación de fenómenos como propagación, desvanecimiento, interferencia y procesamiento de la señal en el transmisor y receptor debe ser tan realista como sea posible. Elegir OMNeT++ como herramienta de simulación para el estudio de las capas físicas en mención implica el análisis de modelos a nivel de código para establecer el alcance y las limitaciones de estudiar redes IEEE 802.11a e IEEE 802.11p mediante simulaciones con los frameworks INETMANET y Veins.

Es así como este proyecto documenta cómo se realizó la implementación de la capa física IEEE 802.11a y 802.11p para redes ad hoc en el simulador OMNeT++ a través del análisis

de su código. Gracias al análisis se podrá describir la funcionalidad de los componentes principales de INETMANET y Veins, establecer cómo interactúan los diferentes componentes y se podrá explicar el proceso de transmisión/recepción con el uso de diagramas. Al concluir el análisis mencionado será posible comparar la capa física implementada en los frameworks con el estándar correspondiente para poder establecer las limitaciones que tiene OMNeT++ en la simulación de redes ad hoc IEEE 802.11a y 802.11p.

Con este estudio se facilitará información a nuevos investigadores, quienes podrán decidir si OMNeT++ es la herramienta idónea para su investigación. Esto permitirá al investigador conocer los modelos que están incorporados y motivar el desarrollo de nuevos modelos y funcionalidades según sus requerimientos.

1.1 OBJETIVOS

El objetivo general de este Estudio Técnico es estudiar la implementación de la capa física IEEE 802.11a y 802.11p para redes ad hoc en el simulador OMNeT++ a través del análisis de su código.

Los objetivos específicos de este Estudio Técnico son:

- Analizar la estructura del código a nivel de capa física de los frameworks INETMANET y Veins para la identificación de los componentes principales.
- Describir el proceso de transmisión y recepción en las capas físicas implementadas en INETMANET y Veins mediante diagramas.
- Determinar el alcance de OMNeT++ y sus framework INETMANET y Veins para la simulación de redes ad hoc IEEE 802.11a y 802.11p a nivel de capa física.
- Analizar cómo se ven afectados el throughput y el retardo de los mensajes usando distintos modelos de propagación para INETMANET y Veins.

1.2 ALCANCE

El proyecto está constituido por un análisis de la implementación de la capa física para los estándares IEEE 802.11a e IEEE 802.11p en INETMANET y Veins, respectivamente. La capa física de ambos estándares es muy similar, sin embargo, su implementación en el

simulador se realizó en dos frameworks distintos esto implica que el estudio del código se haga por separado.

El análisis del código permitirá describir cómo se lleva a cabo el proceso de transmisión/recepción de una trama entre un par de nodos en los frameworks INETMANET y Veins. Esto incluye los componentes principales y la interacción que llevan a cabo para simular una red IEEE 802.11a con INETMANET o IEEE 802.11p con Veins en modo ad hoc. El proceso de transmisión/recepción al que se hace mención se describirá a través de diagramas que muestren la interacción entre clases y métodos lo mismo que se logrará mediante una caracterización de los frameworks mencionados a través del análisis de sus códigos con relación a los estándares.

Los diagramas mencionados no son estandarizados, puesto que se pretende reducir la dificultad del proceso de familiarización con INETMANET y Veins y para ello se buscarán las formas más convenientes para ilustrar la estructura de cada framework y la interacción de sus componentes.

Así la documentación incluye la descripción de los componentes principales de INETMANET y Veins y su interacción, y la explicación del proceso de transmisión/recepción mediante diagramas. Esto permitirá establecer las limitaciones de OMNeT++ para simular redes IEEE 802.11a e IEEE 802.11p; es decir, se definirán qué características del estándar se ven reflejadas en simulación y cuáles aún no han sido contempladas en INETMANET y Veins.

Adicionalmente, a partir de los escenarios de simulación incluidos en INETMANET y Veins se comprobará cómo cambian las mediciones de throughput y retardo promedio de los mensajes usando diferentes modelos de propagación disponibles en el simulador. Se obtendrán curvas de throughput vs número de nodos, throughput vs velocidad de vehículo, retardo vs número de nodos y retardo vs velocidad para ambientes de simulación con un número variable de vehículos (nodos).

El desarrollo de estas simulaciones contempla la creación de scripts para analizar los resultados de las simulaciones, la modificación de archivos de configuración y la implementación de código que será añadido a los ejemplos de INETMANET y Veins para obtención de resultados.

1.3 MARCO TEÓRICO

Las redes ad hoc no dependen de ninguna infraestructura predefinida para poder establecer la comunicación entre dispositivos, esta ventaja ha ocasionado que su uso se extienda ampliamente. En este proyecto, las redes ad hoc conocidas como MANETs (Mobile Ad Hoc Networks) y VANETs (Vehicular Ad Hoc Networks) son estudiadas a nivel de capa física según los estándares IEEE 802.11a e IEEE 802.11p respectivamente. Para esta sección comenzaremos con una descripción de las redes ad hoc que son objeto de estudio y luego se estudiará su capa física de acuerdo con el estándar IEEE 802.11. En este estudio de capa física se incluyen las especificaciones según los estándares IEEE 802.11a e IEEE 802.11p, estructura de trama y proceso de transmisión/recepción.

Además, se revisa brevemente los modelos de propagación que serán empleados en las simulaciones. Por otro lado, se estudian conceptos de modelamiento de OMNeT++ y su funcionamiento. Finalmente, se revisan conceptos básicos de programación orientada a objetos.

1.3.1 REDES AD HOC

Los rápidos avances de las tecnologías inalámbricas permiten comunicarnos sin necesidad de un cable y sin ningún tipo de restricciones geográficas y de tiempo [1]. Así, la necesidad de las personas de estar conectadas a través de una red inalámbrica ha ido aumentando y como consecuencia el uso de dispositivos con tecnología inalámbrica tiene mayor demanda.

Las redes ad hoc inalámbricas son un factor clave en la evolución de las redes inalámbricas [2]. Estas redes no dependen de un dispositivo central que administre la red, sino que todos los dispositivos se comunican entre sí de forma directa [3] por el canal inalámbrico sin ningún tipo de infraestructura. Los dispositivos de una red ad hoc están en igualdad de condiciones [4] y la responsabilidad de organización y control se distribuye entre todos los dispositivos de la red [5]. Estas redes son autónomas, se configuran y se organizan por sí mismas [4]; en consecuencia, presentan mayor flexibilidad debido a su topología dinámica. Gracias a las ventajas que presentan las redes ad hoc inalámbricas, su estudio continúa siendo de gran interés para la comunidad científica.

Las redes ad hoc se clasifican en: MANETs (Mobile Ad Hoc Networks), redes inalámbricas mesh (Mesh Networks) y redes de sensores (Wireless Sensor Networks) [6]. Nuestro estudio se centrará en las MANETs y un tipo específico de MANET conocido como VANET (Vehicular Ad Hoc Network).

1.3.2 MANET

Es una red autoconfigurable sin infraestructura cuyas estaciones móviles se conectan a través de enlaces inalámbricos que forman topología de red dinámica. Los nodos que se encuentran en rango de otros se pueden comunicar directamente y son responsables de descubrirse unos a otros [7]. Para que sea posible la comunicación con nodos fuera de rango, nodos intermedios reenvían la información hasta arribar a su destino.

Las MANET se caracterizan por su movilidad, autonomía y topología dinámica [8], presentando ventajas como: fácil y rápido despliegue, independencia de infraestructura y rentabilidad económica [9].

Debido a su gran flexibilidad, las MANET posibilitan un creciente número de aplicaciones como: operaciones de emergencia, búsqueda y rescate, operaciones militares y policiales, tráfico vehicular, recuperación ante desastres naturales, redes de comunicación en entornos civiles, redes de área personal, etc [8].

Entre los tipos de MANET tenemos**:** VANET, FANET, InVANET [10], Imobile Ad Hoc Network [11]. Nuestro estudio se centrará en un tipo específico de MANET, las VANETs (Vehicular Ad Hoc Network).

Las VANETs son un caso específico de MANET tradicionales usadas para la comunicación entre vehículos e infraestructura de avenidas. El principal objetivo de estas redes es configurar y mantener las comunicaciones entre los vehículos que la conforman sin usar un servidor centralizado [3]. Las VANETs tienen muchas aplicaciones, una de las más críticas consiste en transmitir información en situaciones de emergencia en las que no se dispone de infraestructura.

Los principales componentes de una VANET son las AU (Appliaction Unit), OBU (On Board Unit) y RSU (Road Side Unit) [12]. Las OBUs son dispositivos que van dentro de los vehículos y que se comunican con otras OBUs o con las RSUs. Las AUs también se encuentran dentro del vehículo, hacen uso de información provista por las RSUs u OBUs.

Las AUs se comunican con la red haciendo uso de las OBUs [12]. Las RSUs no forman parte de los vehículos, se encuentran en lugares como intersecciones o espacios de parqueo.

En una VANET, un vehículo se puede comunicar con otro vehículo directamente, a esto se denomina Vehicle to Vehicle (V2V), o un vehículo puede comunicarse con una RSU (Road Side Unit), a esto se conoce como Vehicle to Infraestructure (V2I) [3].

1.3.3 TECNOLOGÍAS DE REDES AD HOC MÓVILES

En su comienzo las redes ad hoc móviles eran utilizadas en aplicaciones militares y en la mitad de 1990 con la comercialización de las tecnologías inalámbricas [13] y con la formación del Grupo de Trabajo de Redes Ad Hoc Móviles de la IETF (Internet Engineering Task Force) para estandarización de protocolos, estas tecnologías tuvieron muchas más aplicaciones [14].

Las tecnologías que permiten la formación de redes ad hoc se resumen en la Tabla 1.1, las cuales han sido ampliamente estudiadas en los últimos años [13].

Tabla 1.1. Tecnologías de redes móviles Ad Hoc [13].

Tecnología	Tasa de datos teórica	Frecuencia	Alcance	Consumo de potencia
IEEE 802.11b	1, 2, 5.5 y 11 Mbit/s	2.4 GHz	25-100 m (indoor) 100-500 m (outdoor)	- 30 mW
IEEE 802.11g	Hasta 54 Mbit/s	2.4 GHz	25-50 m (indoor)	- 79 mW
IEEE 802.11a	6, 9, 12, 24, 36, 49 y 54 Mbit/s	5 GHz	10-40 m (indoor)	40 mW, 250mW o 1W
Bluetooth (IEE 802.15.1)	1 Mbit/s (v1.1)	2.4 GHz	10 m (hasta 100m)	1 mW (hasta 100 mW)
UWB (IEEE 802.15.3)	110-480 Mbit/s	3-10 GHz	~10 m	100 mW, 250 mW
IEEE 802.15.4 (por ejemplo, Zigbee)	20, 40 or 250 Kbit/s	868 MHz, 915 MHz o 2.4 GHz	10-100 m	1 mW
HiperLAN2	Hasta 54 Mbit/s	5 GHz	10-150 m	200 mW o 1 W
HomeRF	1 Mbit/s (v1.0) 10 Mbit/s (v2.0)	2.4 GHz	~50 m	100 mW
IEEE 802.16 IEEE 802.16a IEEE 802.16e (Banda Ancha)	32-134 Mbit/s Hasta 75 Mbit/s Hasta 15 Mbit/s	10-66 GHz < 11Ghz < 6GHz	2-5 Km 7-10 Km (max 50 Km) 2-5 Km	Control de potencia complejo

Como se puede observar las tecnologías más utilizadas son IEEE 802.11 e IEEE 802.16 que corresponden a WiFi y WiMAX. En nuestro caso de estudio, nos interesa el estándar IEEE 802.11 y sus especificaciones para capa física tanto para redes MANET IEEE 802.11a y redes VANET IEEE 802.11p.

El estándar IEEE 802.11 fue lanzado en 1997 y desde entonces ha tenido múltiples actualizaciones para ponerse al día con los avances de las tecnologías de comunicación mencionadas [15]. A continuación, se realizará el estudio de la capa física de los

estándares IEEE 802.11 a e IEEE 802.11p y los procesos ejecutados durante la transmisión y recepción de mensajes.

1.3.4 ESTÁNDAR IEEE 802.11a

El estándar define especificaciones de capa física. Define una modulación OFDM (Orthogonal Frequency Division Multiplexing) y una frecuencia de trabajo de 5GHz por lo que tiene mayores velocidades hasta 54Mbps [16]. A nivel de subcapa MAC utiliza CSMA/CA como método de acceso al igual que IEEE 802.11 legacy.

1.3.4.1 Especificaciones de capa física para IEEE 802.11a

En el estándar IEEE 802.11a se definen los parámetros de capa física definidos en la Tabla 1.2. Este estándar opera en la banda de 5GHz, con un canal de 20MHz y una señal de 16.6MHz. La tasa de transmisión especificada va de 6 a 54 Mbps. Las subportadoras se modulan con BPSK, QPSK, 16QAM o 64QAM. Esto depende de las condiciones del canal [17]. Para codificación se utilizan velocidades de 1/2, 2/3 o 3/4. Se tienen 52 subportadoras, de las cuales 48 son para datos y 4 actúan como pilotos con una separación de 0,3125 MHz [18]. La duración del símbolo OFDM es de 4us, con un período de guardia de 800ns [18].

Tabla 1.2. Parámetros de la capa física de IEEE 802.11a.

Parámetro	Valor especificado
Espaciamiento de canal	20 MHz
Ancho de banda de señal	16.6 MHz
Espaciamiento de subportadora	0,3125 MHz
Tasa de datos	6,9,12,18,24,36,48,54 Mbits/s
Modulación	BPSK, QPSK, 16QAM, 64QAM
Velocidad de Codificación	1/2, 2/3 o 3/4
Número total de subportadoras	52 (-26 a +26)
Número de subportadoras de datos	48
Número de subportadoras piloto	4 (-21, -7, +7, +21) BPSK
Subportadoras DC	0
Duración de símbolo OFDM	4 μ sec
Intervalo de guarda	800 η sec

1.3.4.2 Estructura de canales de IEEE 802.11a

El espectro de la banda de 5GHz utilizada en IEEE 802.11a no es continuo y se divide en varias subbandas [19], una parte va desde 5.15GHz a 5.35GHz y la otra de 5.725 GHz a 5.825GHz, es decir, ocupa el espectro de las bandas U-NII como se muestra en la Tabla 1.3.

Tabla 1.3. Características de las bandas U-NII *[19] [20]* .

Banda	Canales	Banda de Frecuencia (GHz)	Frecuencias de portadora (GHz)	Ambientes de aplicación
U-NII 1	4	5.15-5.25	5.18 5.2 5.22 5.24	Indoor
U-NII 2	4	5.25-5.35	5.26 5.28 5.3 5.32	Indoor/Outdoor
U-NII 3	4	5.725-5.825	5.745 5.765 5.785 5.805	Outdoor

Cada banda U-NII proporciona 4 canales para tener un total de 12 canales en el espectro [20].

IEEE 802.11a utiliza OFDM por medio del cual se divide el ancho de banda disponible en N subportadoras paralelas de igual ancho de banda para transmisión de información. El estándar especifica para cada canal un ancho de banda total de 20MHz.

Como se visualiza en la Figura 1.1, un canal contiene un símbolo OFDM con 4 subportadoras piloto y 48 subportadoras de datos, dando un total de 52 subportadoras numeradas de -26 a 26, con una separación entre subportadoras de 0.3125 MHz.

Las subportadoras de datos utilizan la misma modulación ya sea BPSK, QPSK, 16QAM y 64QAM, mientras que las subportadoras piloto (-21, -7, 7, 21) utilizan BPSK.

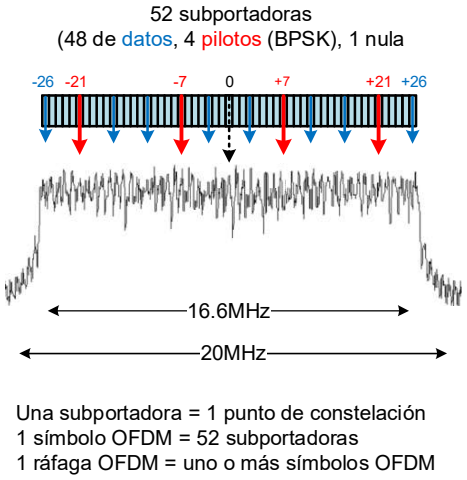

Figura 1.1 Espectro de frecuencia de un canal OFDM [21].

1.3.5 ESTÁNDAR IEEE 802.11p

Los estándares IEEE 802.11p e IEEE 1609 definen el acceso inalámbrico en ambientes vehiculares, se denominan WAVE (Wireless Access in Vehicular Environments). WAVE proporciona una arquitectura para las comunicaciones V2X, destinada al uso de aplicaciones de seguridad y eficiencia vial [22]. En la Figura 1.2, se ilustra la arquitectura de WAVE [23].

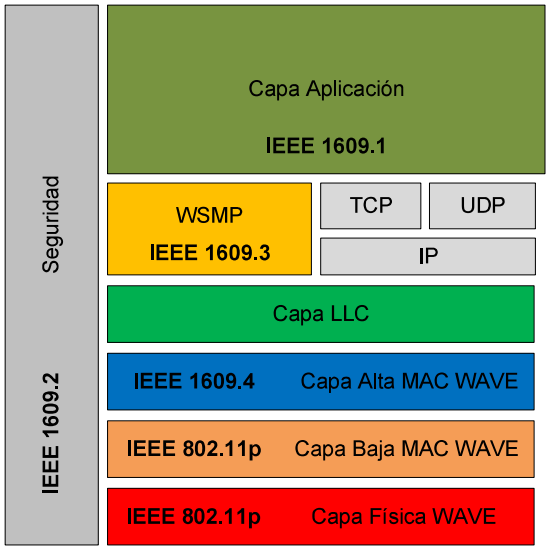

Figura 1.2. Arquitectura de WAVE [23].

1.3.5.1 Especificaciones de capa física del estándar IEEE 802.11p

Actualiza y expande 802.11a a nivel físico y MAC para mejorar su comportamiento en entornos vehiculares. Al igual que el protocolo IEEE 802.11a, el estándar IEEE 802.11p utiliza transmisión multiportadora OFDM; pero divide su espectro de 75MHz en 7 canales

de 10MHz, tiene una tasa de transmisión de hasta 27Mbps, opera en bandas de 5.8 y 5.9 GHz y tiene un rango de 300 a 1000m [24]. En la Tabla 1.4 se muestra los parámetros definidos para la capa física de IEEE 802.11p.

Tabla 1.4. Parámetros de capa física definidos en IEEE 802.11p [24].

Especificaciones	**IEEE 802.11p**
Tasa de datos (Mbps)	3, 4, 5, 6, 9, 12, 18, 24, 27
Modulación	BPSK, QPSK,16QAM, 64QAM
Tasa de codificación	1/2, 1/3, 3/4
Subportadoras	52
Duración de símbolo	8us
Tiempo de Guarda	1.6us
Duración de Preámbulo	32us
Espaciamiento de Subportadora	0.15625 MHz

1.3.5.2 Estructura de canales de IEEE 802.11p

El estándar IEEE 802.11p permite hacer uso de las frecuencias que van desde 5.850 a 5.925 GHz. El espectro de 75 MHz tiene 7 canales que empiezan en el 172 y terminan en el 184, véase la Figura 1.3. El canal 178 es conocido como CCH (Control Channel); es usado únicamente para controlar la transmisión de los broadcasts y el establecimiento de los enlaces [23]. Los canales 172 y 184 se usan exclusivamente para seguridad, el primero sirve para seguridad crítica y el segundo protege de congestión en otros canales [12]. Los canales 174, 176, 180 y 182 se conocen SCH (Service Channel).

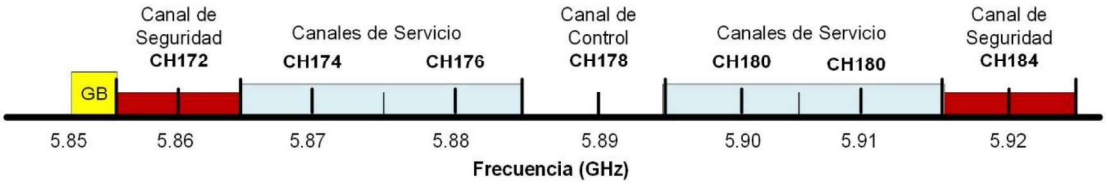

Figura 1.3. Canales en IEEE 802.11p [23].

1.3.6 COMPARACIÓN ENTRE IEEE 802.11a E IEEE 802.11p

Los estándares IEEE 802.11a e IEEE 802.11p se basan en la especificación para sistemas basados en OFDM (Orthogonal Frequency Division Multiplexing) de IEEE 802.11. Es decir que los dos estándares tienen el mismo formato de trama PLCP y realizan el mismo procesamiento en transmisión/recepción. Se diferencian por sus parámetros especificados en capa física como se observa en la Tabla 1.5.

Tabla 1.5. Comparación entre IEEE 802.11a e IEEE 802.11p [25].

Especificaciones	IEEE 802.11a	IEEE 802.11p
Tasa de datos (Mbps)	6, 9, 12, 18, 24, 36, 48, 54	3, 4, 5, 6, 9, 12, 18, 24, 27
Modulación	BPSK, QPSK,16QAM, 64QAM	BPSK, QPSK,16QAM, 64QAM
Tasa de codificación	½, 1/3, 3/4	½, 1/3, 3/4
Subportadoras	52	52
Duración de símbolo	4us	8us
Tiempo de Guarda	0.8us	1.6us
Período FFT	3.2us	6.4us
Duración de Preámbulo	16us	32us
Espaciamiento de Subportadora	0.3125 MHz	0.15625 MHz

El ancho de banda de IEEE 802.11p es 10MHz, siendo la mitad del ancho de banda de IEEE 802.11a [25] que es 20MHz. En cuanto al uso de frecuencias IEEE 802.11p trabaja en 5.9 GHz, mientras que IEEE 802.11a opera a 5GHz. Las tasas de datos de IEEE 802.11p son la mitad de IEEE 802.11a, pero se utilizan los mismos esquemas de modulación y velocidad de codificación. En los dos estándares se tienen 52 subportadoras, pero el espaciamiento de IEEE 802.11p es la mitad de IEEE 802.11a. Es por esto que la duración de símbolo, el tiempo de guarda, período FFT y duración del preámbulo de IEEE 802.11P es el doble de IEEE 802.11a.

1.3.7 CAPA FÍSICA DE LOS ESTÁNDARES IEEE 802.11a E IEEE 802.11p

La capa física del estándar IEEE 802.11a e IEEE 802.11p se basa en la capa física del estándar IEEE 802.11 en la especificación para sistemas basados en OFDM. Define dos subcapas a nivel de capa física: la subcapa PLCP (Physical Layer Convergence Procedure) y subcapa PMD (Physical Medium Dependent) [26], como se muestra en la Figura 1.4.

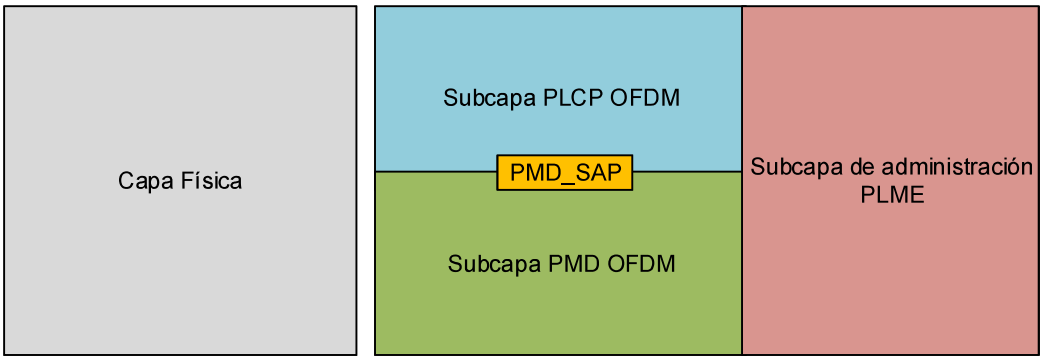

Figura 1.4. Subcapas a nivel de capa física [26].

La subcapa PLCP proporciona una función de convergencia para tener la mínima dependencia de la subcapa MAC y la subcapa PMD adapta la PDU MAC a un formato adecuado para su transmisión y recepción por medio de la encapsulación [26]. Todo esto simplifica los servicios de la interfaz física que se ofrecen a la capa superior.

La PLCP es responsable de la comunicación de la capa física con la capa MAC, convierte las PDU de la capa MAC en una trama OFDM [23].

La subcapa PMD define características y métodos de transmisión y recepción de información entre dos o más estaciones a través del medio inalámbrico [26]. Especifica técnicas de modulación, codificación y la operación de transmisión [16]. Además, define un conjunto de primitivas para descubrir la interfaz entre la PLCP y la PMD.

Adicionalmente se tiene una subcapa de administración PLME (Physic Layer Management Entity) que maneja las diferentes opciones que ofrece la capa física y coordina la interacción entre capa MAC y capa física [16].

1.3.8 ESTRUCTURA DE LA TRAMA PLCP

La trama PLCP para IEEE 802.11a e IEEE 802.11p es la misma que se especifica por el estándar IEEE 802.11 en el apartado de sistemas basados en OFDM. La trama PLCP contiene un preámbulo, un campo de señal y un campo de datos como se muestra en la Figura 1.5.

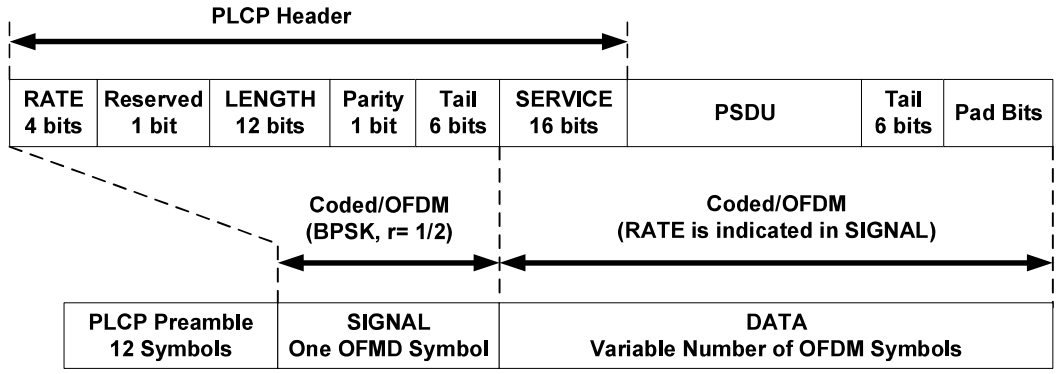

Figura 1.5. Formato de la trama PLCP [26].

Existen varios parámetros asociados al tiempo que permiten definir la duración de una trama PLCP [27]. La Figura 1.6 muestra como se calcula la duración de la trama PLCP.

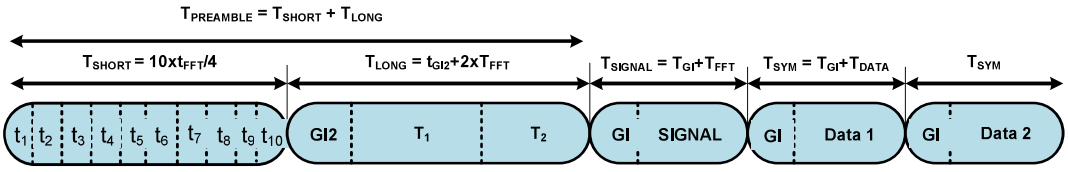

Figura 1.6. Duración de la trama PLCP [28].

1.3.8.1 Preámbulo PLCP

El preámbulo se usa parasincronización y para corregir el offset de tiempo y frecuencia [24]. Se usa para entrenar el VCO (Voltage Controlled Oscillator) del receptor con la señal recibida, para producir un reloj que esté sincronizado con la señal entrante, de modo que sea posible una óptima demodulación [23].

El preámbulo está compuesto de 12 símbolos de entrenamiento, 10 símbolos de entrenamiento corto STS (Short Term Symbol) denotados de t1 a t10 y 2 símbolos de entrenamiento largos LTS (Long Term Symbol) denotados por T1 y T2 [29] . Siete de los STS son responsables de la detección de señal, AGC (Automatic Gain Control) y selección de diversidad. Los símbolos restantes permiten sincronización y estimación de la frecuencia de las subportadoras [23]. Los dos LTS restantes, son utilizados para estimación del canal [30].

La Figura 1.7 muestra como se componen las secuencias de entrenamiento y sus funciones.

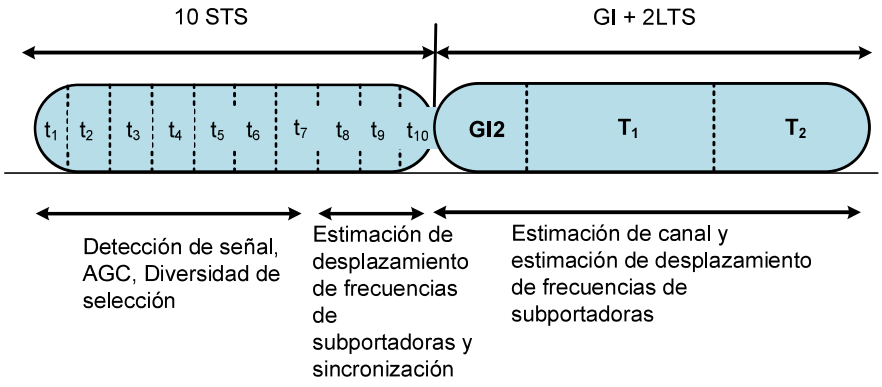

Figura 1.7. Preámbulo de la trama PLCP [26].

En cuanto a la duración de estas secuencias de entrenamiento sus valores se encuentran especificados en la Tabla 1.5.

1.3.8.2 Campo de señal (Cabecera PLCP)

El campo señal (Signal) se utiliza para identificar la tasa de datos y para proporcionar información sobre longitud [30]. Tiene una longitud de 24 bits que se subdividen en cinco campos: RATE, RESERVED, LENGTH, PARITY, y TAIL [23]. Ver Figura 1.8.

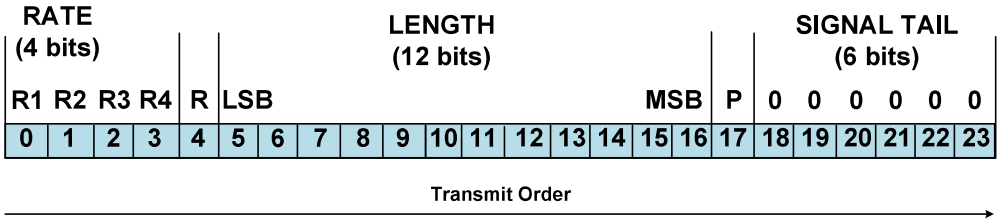

Figura 1.8 Cabecera de la trama PLCP [26].

El campo **velocidad (RATE)** contiene cuatro bits que representan la velocidad a la que se transmite [28], sus valores se encuentran definidos en el estándar IEEE 802.11 sección 17.3.4.2.

El campo **reservado R (RESERVED)** de un bit está ubicado en el cuarto bit y se reserva para usos futuros [28].

El campo **longitud (LENGTH)** contiene los siguientes doce bits e indica el número de octetos de la subcapa MAC que serán transmitidos [28].

El bit de **paridad P (PARITY)** obtiene paridad positiva (par) de los bits del 0 al 16 [28].

El campo **cola (TAIL)** está formado por seis bits que se configuran en 0. Estos ceros se usan para sincronizar el descrambler y para poner el codificador convolucional en cero [26].

Los campos velocidad, reservado, longitud, paridad, cola y servicio constituyen un símbolo OFDM con la información de velocidad de transmisión y tasa de codificación. La codificación del campo señal se realiza con BPSK y con código convolucional a R=1/2 [27]. Este proceso incluye codificación convolucional, entrelazado (interleaving), proceso de mapeo en modulación, inserción del piloto y modulación OFDM. El resultado es un símbolo OFDM en el cual no se realiza aleatorización (scrambling).

1.3.8.3 Campo de datos

Este campo de la trama PLCP contiene el campo servicio (service), PSDU, bits de cola (tail bits) y PAD bits. En la Figura 1.9 se muestra la estructura del campo de datos.

El campo **servicio (SERVICE)** contiene 16 bits (ver Figura 1.9), de los cuales los primeros siete bits (0-6) se configuran en 0 y son usados para sincronización y desaleatorizacion (descrambling) en el receptor. Los bits que siguen (7-15) se reservan para uso futuro y también son configurados en 0 [26] .

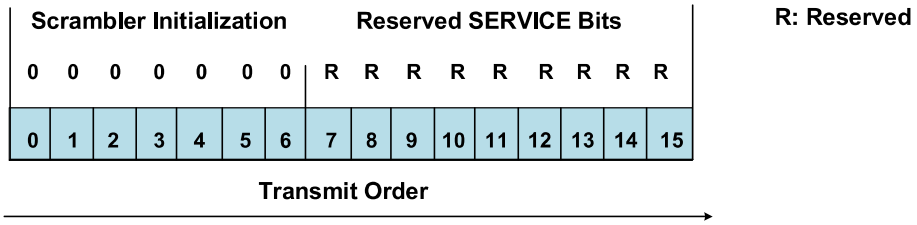

Figura 1.9. Campo servicio del campo de datos de la trama PLCP [26].

El campo **cola (TAIL PPDU)** son seis bits configurados en 0 que hacen que el codificador convolucional llegue al estado cero, lo cual permite la disminución de la probabilidad de error al momento de realizar la decodificación en el receptor [26].

Los **bits de relleno PAD** son un número múltiplo del número de bits de datos por símbolo OFDM N_{DBPS} [26]. Para completar el número de bits a veces se requiere realizar un relleno de bits, el cual se calcula de la siguiente manera:

$$N_{SYM} = Ceiling\left(\frac{16+8*LENGTH+6}{N_{DBPS}}\right) \quad (1.1)$$

$$N_{DATA} = N_{SYM} * N_{DBPS} \quad (1.2)$$

$$N_{PAD} = N_{DATA} - (16 + 8 * LENGTH + 6) \quad (1.3)$$

Donde:

$N_{SYM,}$ es el número de símbolos OFDM

$N_{DATA,}$ es el número de bits del campo de datos.

$N_{PAD,}$ es el número de bits de relleno, configurados en 0

$N_{DBPS,}$ es el número de bits de datos por símbolo OFDM

La función ceiling retorna el número entre mayor o igual a su argumento. Para este cálculo es necesario conocer la longitud **LENGTH** del PSDU de la capa MAC [26].

1.3.9 PROCESO DE TRANSMISIÓN/RECEPCIÓN DE LA TRAMA PLCP

La Figura 1.10 presenta el proceso de transmisión a nivel de capa física. Cuando los datos se pasan a la capa física, se someten a aleatorización (scrambling) para evitar una secuencia larga de bits que pueda causar errores [30]. Posteriormente, los datos son codificados con FEC y luego los datos se someten a entrelazado (interleaving) para evitar ráfagas de errores [23]. Después se realiza el mapeo de constelaciones con diferentes modulaciones y a continuación, se realiza la inserción de pilotos para el ensamblado de la trama, y después se realiza la transformada inversa de Fourier y se inserta el prefijo cíclico para formar el símbolo OFDM. Finalmente se inserta el preámbulo y se envía al medio físico. En la recepción, se realiza los procesos inversos.

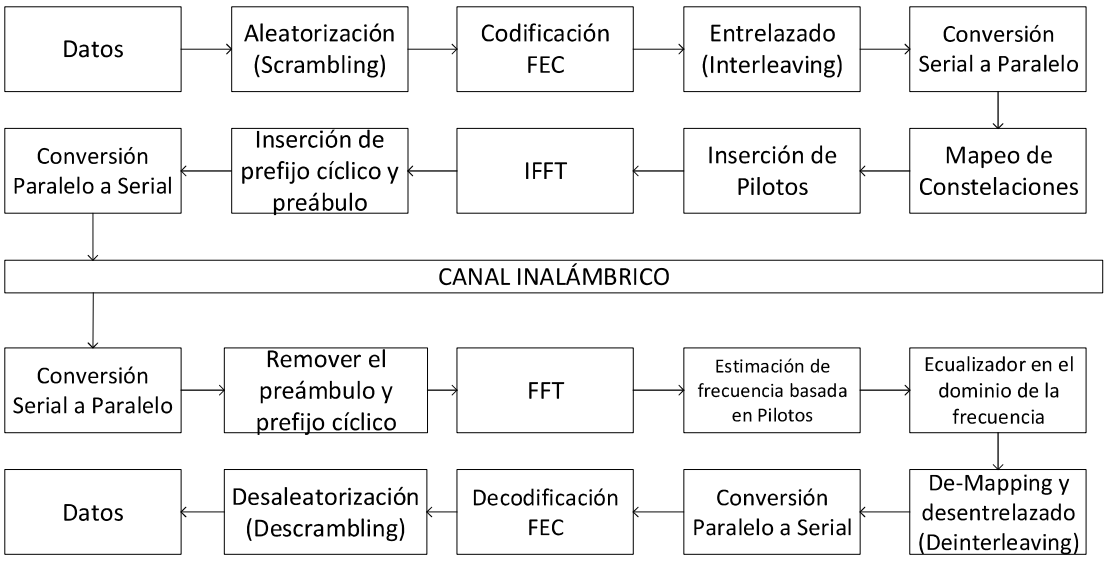

Figura 1.10. Proceso de transmisión a nivel de capa física de IEEE 802.11a e IEEE 802.11p. Adaptado de [23].

A continuación, describiremos estos procesos realizados en el transmisor y en el receptor, y los parámetros que define el estándar para cada uno. El proceso realizado para la formación del símbolo OFDM se describe en la siguiente sección.

1.3.9.1 Aleatorizador y desaleatorizador (scrambling/descrambling)

El campo de datos se aleatoriza con una trama de longitud 127 para sincronización [26]. Para esto se utiliza un generador polinomial S(x) que se describe con la siguiente ecuación:

$$S(x) = x^7 + x^4 + 1 \tag{1.4}$$

Tanto para la transmisión y recepción se utiliza el mismo aleatorizador.

1.3.9.2 Codificador y decodificador

Por medio de la codificación de canal se constrarrestan los efectos producidos por el canal. El campo de datos PLCP utiliza un codificador de 1/2, 3/4 y 2/3 para los estándares estudiados. La codificación depende de los bits de entrada y del registro de desplazamiento cuyo polinomio generador es G0: 133 y G1: 171 [28].

La decodificación se realiza en el receptor con el mismo polinomio generador. En este proceso se utiliza el algoritmo de Viterbi para hallar la secuencia más probable que produce una secuencia de fuentes de información de Markov.

1.3.9.3 Entrelazado y desentrelazado (Interleaving y deinterleaving)

El intercalado de los bits codificados se realiza en un bloque de la longitud del símbolo OFDM. El entrelazado define dos permutaciones [26]:

- Una permutación que asegura que los bits codificados adyacentes sean mapeados en subportadoras no adyacentes.
- Una segunda permutación que asegura que los bits codificados adyacentes sean mapeados de manera alternada del bit menos significativo al más significativo.

Para el desentrelazado se realiza la operación inversa.

1.3.9.4 Mapeo de modulación de subportadora

La modulación utilizada depende de la velocidad, el estándar define modulaciones BPSK, QPSK, 16QAM y 64QAM. Los bits que resultan de la aleatorización se dividen en grupos de 1, 2, 4 o 6 bits para dichas modulaciones y se convierten en números complejos los cuales representan puntos en la constelación [28]. Estos grupos de bits utilizan codificación de Grey por medio de la cual dos grupos de bits sucesivos varían solo en un bit.

Con la combinación de codificación y modulación se determina la velocidad de datos como se indica en la tabla. Estos valores están especificados en el estándar IEEE 802.11 y se toman los valores correspondientes a un espaciamiento de canal de 20MHz y 10MHz. En la Tabla 1.6, se detallan dichos valores.

La selección de la tasa del código y de la modulación, tiene un efecto directo en la tasa de datos del sistema [24].

Tabla 1.6. Parámetros que dependen de la modulación.

Modulación	Tasa de codificación (R)	Velocidad de datos (Mb/s) IEEE 802.11a	Velocidad de datos (Mb/s) IEEE 802.11p
BPSK	1/2	6	3
BPSK	3/4	9	4.5
QPSK	1/2	12	6
QPSK	3/4	18	9
16-QAM	1/2	24	12
16-QAM	3/4	36	18
64-QAM	2/3	48	24
64-QAM	3/4	54	27

1.3.10 OFDM

Para realizar la modulación OFDM en el transmisor se realizan los siguientes procesos, mostrados en la Figura 1.11. Proceso de transmisión en OFDM

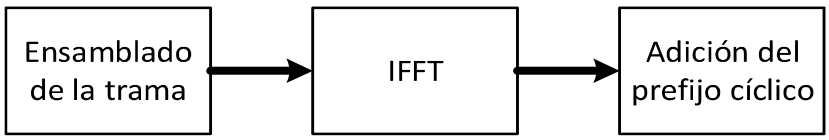

Figura 1.11. Proceso de transmisión en OFDM.

Para el receptor se realizan los procesos mostrado en la Figura 1.12.

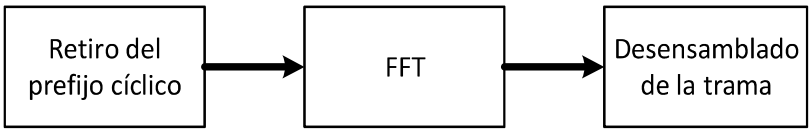

Figura 1.12. Proceso de recepción en OFDM.

El símbolo OFDM se forma por 64 subportadoras:

- 4 subportadoras piloto
- 48 subportadoras de datos (0-47)
- 11 subportadoras vacias
- 1 subportadora DC

Las subportadoras piloto corresponden a los lugares -21, -7, 7, 21 y hacen que el sistema se vuelva robusto frente a desplazamientos de frecuencia y ruidos de fase.

En el transmisor, por medio de la transformada inversa de Fourier de forma el símbolo OFDM con la suma de N señales moduladas en canales de un mismo ancho de banda. En el receptor se realiza el proceso inverso con la transformada de Fourier [31].

El prefijo cíclico se utiliza para reducir los efectos de la interferencia intersímbolo (ISI) y la interferencia entre portadoras (ICI). Se forma con las muestras de la última parte de la señal en el dominio del tiempo, el cual es colocado al comienzo del símbolo con una duración de 1/4 de la duración del símbolo OFDM [32]. En el transmisor se añade este prefijo cíclico, mientras que en el receptor es retirado.

Existen dos tipos de intervalos de Guarda: intervalos G11 e intervalos G12 que se insertan luego del prefijo cíclico. Los intervalos G11 se utilizan en los extremos del espectro OFDM para establecer separación con bandas adyacentes. Los intervalos G12, se colocan al principio de cada símbolo OFDM para evitar problemas de ISI e ICI que ocurren debido al multipath [33].

1.3.11 CANAL INALÁMBRICO

En una comunicación inalámbrica la transmisión de señales se realiza a través de la interfaz aire por medio de ondas electromagnéticas. El uso de este medio inalámbrico provee grandes ventajas como movilidad y portabilidad entregando información a gran cantidad de usuarios.

La señal que se propaga por el medio inalámbrico se ve afectada por varios fenómenos que hacen que se atenúe, entre estos fenómenos podemos mencionar la presencia de obstáculos que producen reflexión, difracción, absorción, dispersión y múltiples caminos que pueden llegar a cancelar la señal.

La propagación de la señal que viaja por un canal inalámbrico es muy compleja debido a los fenómenos mencionados y por tanto modelarlos también resulta complejo. Estos problemas producidos por la propagación dependen del entorno en el que se desarrolla la comunicación y para cada tipo de entorno se han desarrollado varios modelos que describen como se propaga la señal.

1.3.12 MODELOS DE PROPAGACIÓN

Los modelos de propagación estiman las condiciones que atraviesa una señal que viaja por el medio inalámbrico y lo modelan en base a propiedades del entorno. Existen varios modelos de propagación los cuales se pueden como se indica en la Tabla 1.7.

Tabla 1.7. Tipos de modelos de propagación.

Tipos	**Modelos**
Modelos de propagación analíticos	Dos rayos, Tierra Plana, Rec, etc.
Modelos de propagación empíricos y semi empíricos	Okumura Hata, Modelos para interiores, Cost-231, Ikegami, Walfisch,etc.
Modelos estocásticos de propagación	Log-normal, Rician, Rayleigh,etc.

A continuación, se describirán los modelos de propagación que serán utilizados en las simulaciones de INETMANET y Veins.

1.3.12.1 Modelo de Propagación en el Espacio Libre o Modelo de Friis

Es un modelo que estima como disminuye la potencia de recepción Pr en función de la distancia de separación entre el transmisor y el receptor [34]. Considera línea de vista entre el transmisor y el receptor y no toma en cuenta efectos como difracción, reflexión, etc. Se representa mediante la Ecuación 1.5 [34]:

$$P_r = \frac{P_t G_t G_r \lambda^2}{(4\pi)^2 d^2 L} \tag{1.5}$$

Donde, Pr es la potencia de recepción, Pt la potencia de transmisión, Gr ganancia de la antena del receptor, Gt ganancia de la antena del transmisor, λ la longitud de onda de la señal, d distancia y L factor de atenuación.

La Ecuación 1.5 se representa en dBm de la siguiente manera:

$$P_{r_{dBm}} = P_{t_{dBm}} + 10 \log_{10}\left(\frac{G_t G_r \lambda^2}{16 \pi^2 L}\right) - 20 \log_{10} d \tag{1.6}$$

1.3.12.2 Modelo de Reflexión de Dos Rayos

Es un modelo de propagación que toma en cuenta las reflexiones del suelo, considera que al receptor llega la señal directa y la señal reflejada [35].

Las pérdidas se representan con la Ecuación 1.7:

$$P_{r_{dBm}} = P_{t_{dBm}} + 10\ \log_{10}\left(\frac{{h_t}^2\ {h_r}^2 G_t G_r}{L}\right) - 40\ \log_{10} d \quad (1.7)$$

Esta expresión es válida si la distancia entre nodos es mayor que la suma de la altura de las antenas, si la reflexión es perfecta y si la diferencia de las fases de las dos señales es pequeña.

1.3.12.3 Log-Normal Shadowing

Las señales que se propagan en entornos irregulares atraviesan obstáculos antes de llegar al receptor, estos obstáculos introducen un factor aleatorio que sigue una distribución normal [35], definida por la Ecuación 1.8:

$$P_{r_j} = P_{i_j} 10^{-d_j/10} \quad (1.8)$$

En donde Prj es la potencia reflejada, transmitida o difractada por el j-ésimo obstáculo, Pij la potencia incidente sobre el obstáculo y dj las perdidas debidas al obstáculo en dBs.

El teorema de límite central permite concluir que las pérdidas en dBs siguen una distribución normal de media $\mu_{L_{dB}}$y desviación $\sigma_{L_{dB}}$.

El modelo indica que la potencia de recepción Pr está distribuida normalmente Xσ con una desviación estándar σ respecto a la potencia calculada en dB. Así, el modelo se caracteriza por la ecuación:

$$P_{r_{dBm}} = P_{r_{dBm}}(do) + 10\,\gamma\ \log_{10}(d/do) + X_\sigma \quad (1.9)$$

1.3.12.4 Modelo Breakpoint

Se basa en la determinación de la distancia denominada breakpoint. El modelo se describe mediante la Ecuación 1.10 [27]:

$$PL(d) = PL_o + 10\ n\ \log_{10}(d/do) \quad (1.10)$$

En donde, PLo es la pérdida por propagación a la distancia de referencia y n es el coeficiente de perdida.

1.3.12.5 Rayleigh Fading

Útil en ambientes outdoor con gran cantidad de elementos reflectores entre trasmisor y receptor. Si el ambiente de propagación no tiene línea de vista, múltiples copias de la señal

llegan al receptor como producto de la multitrayectoria. El modelo sigue una función de densidad de probabilidad descrita por la Ecuación 1.11 [36]:

$$P(r) = (r/\sigma^2)\, e^{-(r/2\sigma^2)} \tag{1.11}$$

En donde σ es el valor en rms de la señal recibida, $\frac{r^2}{2}$ es la potencia instantánea y σ^2 es el promedio de la potencia local de la señal recibida antes de la detección de la envolvente.

1.3.12.6 Rician Fading

Cuando se tiene un ambiente de propagación con línea de vista la variación aleatoria del nivel recibido sigue una distribución de Rician que se describe con la Ecuación 1.12 [36]:

$$P(r) = \frac{r}{\sigma^2}\, e^{-\left(\frac{r^2+A^2}{2\sigma^2}\right)} Io\left(\frac{Ar}{\sigma^2}\right) \tag{1.12}$$

En donde A, es la amplitud pico de la señal dominante con un valor mayor que cero, Io es la función modificada de Bessel de orden cero, $\frac{r^2}{2}$ es la potencia instantánea y σ es la desviación estándar de la potencia local [36].

Esta distribución se describe con la Ecuación 1.13 [36].

$$K = 10\, \log\frac{A^2}{2\sigma^2}\ (dB) \tag{1.13}$$

K representa la relación entre la potencia de la señal dominante y la potencia de las señales reflejadas. Si se tiene un canal dominante fuerte, este factor K está en el orden de unidades o decenas.

1.3.12.7 Nakagami Fading

Útil para modelar desvanecimiento en comunicaciones móviles, permite caracterizar señales terrestres móviles, ambientes indoor y señales reflejadas por la atmosfera [36]. Un factor m permite aproximar esta distribución a otras distribuciones como Rayleigh, Rician y media gaussiana. El modelo se describe mediante la función de distribución de probabilidad de la Ecuación 1.14 [27] .

$$P(r) = \frac{2}{\Gamma(m)} \left(\frac{m}{\overline{P_r}}\right)^m r^{2m-1}\, e^{-m\frac{r^2}{\overline{P_r}}} \quad (1.14)$$

En donde, $\overline{P_r}$ es el promedio de la potencia recibida, Γ es la función gamma, m es la figura de desvanecimiento (m>1/2), r^2 es la potencia instantánea de la señal recibida que satisface la función gamma.

1.3.12.8 Jakes Fading

Modelo empírico para ambientes vehiculares basado en la suma de sinusoides. Con el modelo se produce una señal que procesa el espectro Doppler. Asume que el receptor recibe rayos de todas las direcciones con un espaciamiento de 2π/N [37]. Este modelo se define por las ecuaciones:

$$r(t) = r_I(t) + jr_Q(t) \quad (1.15)$$

$$r_I = \frac{1}{\sqrt{N}} \sum_{m=1}^{N} \cos(2\pi F_d \cos \alpha_m t + a_m) \quad (1.16)$$

$$r_Q = \frac{1}{\sqrt{N}} \sum_{m=1}^{N} \sin(2\pi F_d \cos \alpha_m t + b_m) \quad (1.17)$$

En donde α_m es el ángulo de llegada, a_m y b_m son valores uniformemente distribuidos de 0 a 2 π.

1.3.13 OMNeT++

OMNeT ++ es un simulador modular de eventos discretos de redes orientado a objetos [38]. Su infraestructura se basa en una arquitectura de componentes para modelos de simulación [39]. Estos modelos se construyen a partir de componentes denominados módulos. A continuación, se describirá varios conceptos básicos relacionados con dichos módulos.

1.3.13.1 Conceptos básicos de modelado en OMNeT++

Un modelo en OMNeT++ se compone de módulos que se comunican por medio de intercambio de mensajes [39]. Los modulos activos se denominan módulos simples, los cuales se agrupan en módulos compuestos. Los módulos simples utilizan la librería de clases y como los módulos compuestos se forman a partir de estos módulos simples, se puede tener niveles de jerarquía ilimitados.

Los mensajes pueden representar tramas o paquetes en una red y contienen estructuras de datos complejas [39]. Estos mensajes se envían y reciben por medio de compuertas que constituyen tanto las interfaces de salida como las interfaces de entrada respectivamente. Estas compuertas están enlazadas por medio de una conexión y, en consecuencia, los mensajes pueden intercambiarse por una serie de conexiones entre módulos.

Las conexiones se utilizan para modelar enlaces físicos y permiten configurar parámetros como: velocidad de datos, retardo de porpagación, BER (Bit Error Rate), PER (Packet Error Rate), etc. Estos parámetros se configuran en archivos NED o en el archivo omnetpp.ini.

Todos los modelos de OMNeT++ están compuestos de los siguientes elementos [38]:

- Archivos NED: Se usan para describir la estructura de los modelos de simulación con parámetros y compuertas. Estos archivos se pueden escribir con cualquier editor de texto, pero el IDE ++ OMNeT permite la edición gráfica y de texto.
- Archivos .msg: Definen tipos de mensaje y permiten añadir campos de datos como encabezados. Debido a que los campos son atributos de una clase se crean subclases de un mensaje determinado para agregar nuevos campos.
- Código fuente de módulos simples: Son archivos de C ++, con extensión .h / .cc .

1.3.13.2 IDE de OMNeT++ y herramientas

El IDE (Integrated Development Environment) de OMNeT++ se basa en la plataforma de Eclipse, y lo extiende con nuevos editores, vistas, asistentes y funciones adicionales [38].

OMNeT++ contiene varias herramientas que permiten crear y configurar modelos, ejecutar simulaciones, registrar eventos durante la simulación y analizar resultados [38]; las cuales describiremos a continuación:

- Editor NED: permite crear y editar archivos NED de forma gráfica y en modo de texto. Contiene una lista de módulos simples, canales, interfaces y conectores para crear módulos compuestos.

- Editor de archivo ini: Permite configurar parámetros de un modelo de simulación.

- Launcher de simulación: permite ejecutar la simulación e ir visualizando su progreso.

– Tabla de secuencias (Sequence chart): permite registrar varios eventos durante la simulación: envío y recepción de mensajes, creación de módulos y conexiones, etc. Toda la información detallada de la simulación estará contenida en un archivo de registro de eventos (evento log file) permitiendo visualizar el intercambio de mensajes entre modulos y una tabla de secuencias.

– Scave: permite analizar los resultados de una simulación. Con esta herremianeta es posible procesar y visualizar los resultados de una simulación que se guardaron en archivos vectoriales (.vec) o escalares (.sca). Adicionalmente permite graficar los resultados tanto en un gráfico de líneas como un histograma.

1.3.13.3 Simulaciones en OMNeT++

En la Figura 1.13 se muestra el flujo para la creación de una simulación. Los primeros archivos en ser procesados son los mensajes (.msg), los que se convierten a código C++ utilizando un programa conocido como opp_msgc [39]. Luego, todo el código en C++, es compilado y enlazado con el kernel de simulación y las librerías de interfaz de usuario para formar un ejecutable de simulación o una librería compartida. Los archivos NED se cargan dinámicamente durante la ejecución de una simulación.

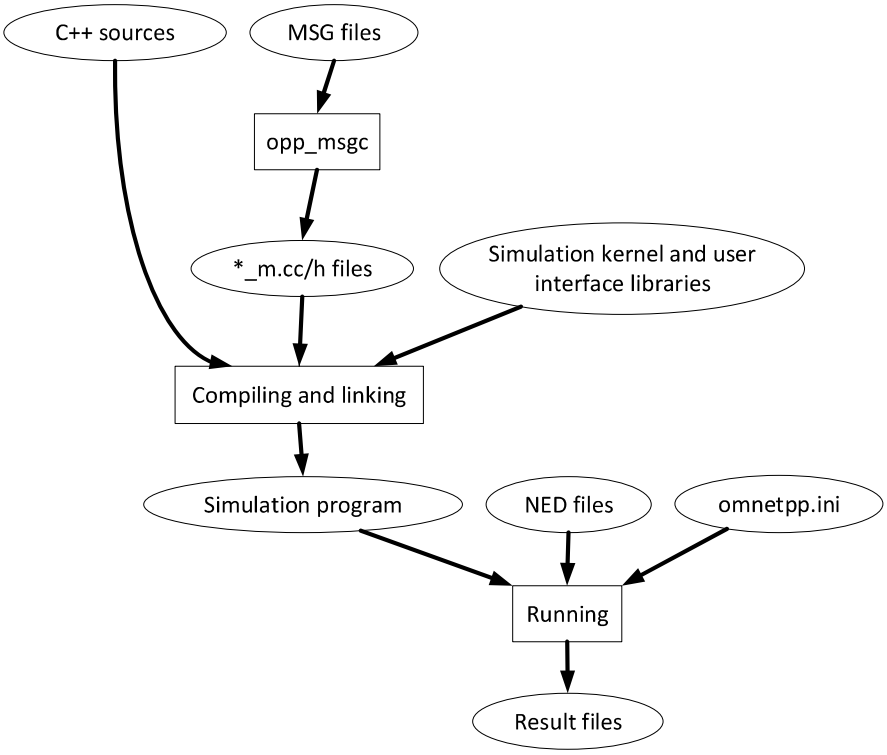

Figura 1.13. Flujo de Creacion de Simulaciones [39].

Cuando se inicia el programa, primero lee los archivos NED, a continuación, el archivo de configuración. Este archivo se llama normalmente omnetpp.ini. El archivo de configuración contiene parámetros que controlan cómo se ejecuta la simulación, los valores de los parámetros de un modelo, etc [39].

La salida de la simulación se escribe en archivos de resultados, estos son archivos de texto orientado a la línea que hace que sea posible procesarlos con una variedad de herramientas y lenguajes de programación.

1.3.14 CONCEPTOS BÁSICOS DE PROGRAMACIÓN ORIENTADA A OBJETOS

OMNeT++ utiliza C++ como lenguaje de programación, el cual permite manipulación de objetos. La programación orientada a objetos agrupa un conjunto de técnicas que permiten desarrollar y mantener fácilmente programas de una gran complejidad [40].

La programación orientada a objetos se fundamenta en los conceptos de objeto y clase. A continuación, describiremos los conceptos básicos de la programación orientada a objetos:

- Objeto: Instancia de una clase. Entidad autónoma que engloba en sí mismo datos y procedimientos necesarios para el tratamiento de esos datos [41]. Por ejemplo, a partir de la clase auto, se puede generar el objeto Mazda. Las clases podrían verse como un tipo de dato y los objetos son variables de ese tipo.

- Métodos: Son operaciones que se deben ejecutar varias veces a lo largo de un programa y por lo tanto se aíslan en un fragmento de código que se invoca con el nombre del método.

- Atributo: Son variables de una clase, pero constituye una cualidad que debe ser común a todos los objetos que se instancien a partir de la clase. Por ejemplo, todos los Autos (clase), pueden tener color, dimensiones, velocidad máxima, etc. Estos son los atributos.

- Clase: Es un tipo de dato, una estructura definida por el programador y define propiedades y comportamiento de un objeto [42], especifica características de un conjunto de objetos. Por ejemplo, a partir de la clase Ave se puede instanciar gallina, a partir de la clase dispositivo se puede generar el objeto computadora.

- Interfaz: conjunto de métodos y atributos que dispone un objeto para comunicarse con él [41].

- Miembros públicos, protegidos y privados: un miembro público es accesible en cualquier lugar en el que exista un objeto de una clase. Un miembro protegido es accesible desde las clases que se heredan de la clase que lo contiene. Un miembro privado es accesible solo por los mpetodos de la clase a la que pertenece. [43]
- Herencia: Constituye la capacidad de crear subclases a partir de clases existentes, por ejemplo, la subclase Ave puede heredar de la clase Animal.
- Encapsulación: Permite acceder a métodos de un objeto usando un punto seguido del nombre del método.
- Modularidad: permite subdividir una aplicación en pequeñas partes denominadas módulos.

2. METODOLOGÍA

En esta sección se realizará el estudio de la capa física de los Framework INETMANET y Veins correspondientes a los estándares IEE 802.11a e IEEE 802.11p respectivamente.

Para el estudio de IEEE 802.11a se revisa la arquitectura de INET Framework a partir del cual se construye INETMANET. Ya en el análisis de INETMANET se parte describiendo sus componentes principales y mensajes, así como los métodos más importantes que permiten modelar la capa física IEEE 802.11a. Así se detallará como se realiza el proceso de transmisión/recepción para redes MANETs.

A continuación, para el estudio de IEEE 802.11p se realizará un breve estudio de Mixim, un framework que constituye la base para la arquitectura que posee Veins. Luego de conocer como trabaja Mixim se estudia los componentes de Veins, mensajes y código para así poder establecer como se realiza el proceso de transmisión/recepción en VANETs.

Finalmente, se realiza las simulaciones a partir de ejemplos incluidos en Veins e INETMANET para evaluar el desempeño de la red en base a medidas de throughput y retardo utilizando diferentes modelos de propagación.

2.1 INET FRAMEWORK

2.1.1 DESCRIPCIÓN

INET Framework es una librería de modelos de código abierto para el entorno de simulación OMNeT++ [44]. INET permite a investigadores y estudiantes diseñar y validar de nuevos protocolos y escenarios para varios tipos de redes de comunicaciones ya que provee varios modelos para su estudio.

INET está implementado sobre OMNeT++ y por ende es mantenido por miembros del equipo de desarrolladores de OMNeT++. Así como OMNeT++ posee modelos que consisten en módulos que se comunican entre sí mediante el envío de mensajes. Estos modelos de INET pueden desarrollarse, ensamblarse, parametrizarse, ejecutarse y evaluarse sus resultados desde el IDE de Simulación OMNeT ++, o desde la línea de comando [44]. Sus componentes (agentes y protocolos) se combinan para formar dispositivos de red. Además, es posible la implementación de nuevos componentes o la modificación de los existentes.

INET presenta las siguientes características [44]:

- Modelos y protocolos para las capas del modelo OSI.
- Capa física con nivel de detalle escalable (modelos de propagación detallados, representación de nivel de bit / símbolo, etc.).
- Varios modelos de aplicaciones.
- Soporte de emulación de red.
- Soporte de movilidad.
- Modelado del entorno físico (obstáculos para la propagación de radio, etc.),
- Soporte de visualización,

2.1.2 AQUITECTURA

INET contiene modelos para [45]:

- Capas del modelo OSI: física, capa de enlace, red, transporte, aplicación.
- Protocolos de capa de red: IPv4 / IPv6.
- Protocolos de capa de transporte: TCP, UDP, SCTP.
- Protocolos de enrutamiento (ad-hoc y cableados).
- Protocolos de capa de enlace cableados e inalámbricos, interfaces cableadas e inalámbricas como Ethernet, PPP, IEEE 802.11, etc.
- Protocolos MANET, DiffServ, MPLS con señalización LDP y RSVP-TE.

Los protocolos están representados por módulos simples que se combinan para formar dispositivos de red. Muchos módulos son módulos compuestos y se forman por otros módulos simples. Los módulos existentes realizan varias funciones [44]:

- Implementar protocolos.
- Almacenar datos.
- Configurar automáticamente una red.
- Permitir movilidad de un nodo.
- Realizar tareas asociadas al canal de radio en comunicaciones inalámbricas.

Estos módulos son escritos en lenguaje C++ y se construyen por los siguientes componentes, como ya se indicó previamente [45]:

- Archivos .ned que describe la estructura del módulo.
- Archivos .msg que define el tipo de mensaje y contiene encabezados del protocolo y formatos del mensaje.
- Módulos simples programados en C++, los cuales contienen el sufijo .h y .cc.

Los módulos envían los mensajes mediante compuertas que constituyen las interfaces de entrada y salida y que reciben y envían dichos mensajes respectivamente [44]. Además, se definen parámetros en estos módulos que permiten su configuración, estos parámetros se definen en los archivos .ned o en el archivo de configuración omnet.ini.

2.1.3 EXTENSIONES

Varios framework se basan en INET y lo extienden para poder estudiar redes específicas [46] [47] [48]:

- OverSim para simulación de redes punto a punto.
- Veins para simulación de comunicación entre vehículos, compuesto por un simulador de red y simulador de tráfico por carretera SUMO.
- SimuLTE para simulación de LTE, con eNodeB y modelos de UE.
- INETMANET para simulación de redes móviles ad-hoc.
- MiXiM no es una extensión INET, sino un framework de modelado de OMNeT ++ independiente que se centra en la capa física inalámbrica, enlace de datos y red y puede utilizarse junto con INET.
- Se tienen otras extensiones como ReaSE, HIPSim ++, INET-HNRL, EPON, mCoA ++, EBitSim, INET / Quagga, entre otras.

Todas estas extensiones son mantenidas por varios grupos de investigación independientes.

2.1.4 CAPA FÍSICA

Para modelar la capa física es importante tomar en cuenta la propagación de la señal en el canal inalámbrico, fading, interferencia y decodificación. Así los modelos desarrollados en INET toman en cuenta las características nombradas y proveen varios parámetros para definir el comportamiento de los dispositivos [45].

La capa física representa las estructuras de datos de la señal en diferente nivel de detalle: dominio de paquete, dominio de bit, dominio de símbolo, dominio de muestreo o forma de onda y dominio analógico. En cada uno de los dominios nombrados la capa física realiza varias funciones específicas, como se describe en la Tabla 2.1.

Tabla 2.1. Estructuras de datos de la capa física [44].

Dominio	Funciones
Paquete	– Definición del mensaje que se envía y recibe de la MAC
Bit	– Inserción/remoción de CRC (Cyclic Redundancy Check) – Codificación/decodificación – FEC (Forward Error Correction) – Interleaving/deinterleaving – Scrambling/descrambling
Símbolo	– Modulación/demodulación – Inserción/remoción del preámbulo – Spreading/despreading – Conformación de haz de la antena – Codificación/decodificación en spacio/tiempo – Multiplexación y Demultiplexación
Muestreo o Forma de Onda	– Filtrado y formación del pulso – Censado de portadora y detección de energía – Sincronización y pll
Analógico	– Conversión DA/AD – PLL

En la capa física se tienen dos tipos de modelos que engloban las características principales para su modelamiento, estos son: modelos de radio y modelos del medio físico.

2.1.4.1 Modelos de Radio

Describen la capacidad que tiene el nodo de transmitir y recibir señales en el medio inalámbrico. Contienen modelos del transmisor, del receptor, de la antena, modelos de error y los modelos de consumo de energía [44].

- Modelos del transmisor: describen el proceso físico que convierte las tramas MAC en una señal eléctrica transmitida al medio. Incluye varios procesos como señalización de paquetes, codificación, scrambling, interleaving, modulación.
- Modelos del receptor: describen como se convierte la señal eléctrica en una trama MAC. Incluye varios procesos como demodulación, descrambling, deinterleaving, decodificación y deserialización.
- Modelos de antena: describen como el dispositivo físico convierte señales eléctricas en señales de radio y viceversa. Incluyen características de la antena como posición, orientación, ganancia, etc.

- Modelos de Error: determinan errores de recepción y describen como se ve afectada la relación señal a ruido en base a dichos errores.
- Modelos de consumo de energía: describen como el medio consume energía por los procesos realizados, principalmente por la transmisión y recepción de señales.

2.1.4.2 Modelos del medio físico

Permiten modelar el canal inalámbrico. Contiene varios modelos [44]:

- Modelos de propagación: describe como las señales se propagan a través del medio físico.
- Modelos de pérdida de camino: describen como se reduce la potencia de la señal al propagarse por el canal inalámbrico. Combina efectos de perdida en el espacio libre, difracción, reflexión, absorción.
- Modelos de perdida de obstáculos: describen como se reduce la potencia de la señal cuando esta pasa a través de obstáculos.
- Modelos Background Noise: describen como el ruido cambia sobre el tiempo y espacio.
- Modelos Neighbor Cache: permiten determinar los receptores vecinos para una transmisión dada.

2.1.5 PROCESO DE TRANSMISIÓN Y RECEPCIÓN EN CAPA FÍSICA

La Figura 2.1 describe el proceso de transmisión/recepción en una simulación con INET. El proceso de transmisión en capa física inicia con la recepción de la trama MAC en el transmisor, donde el modelo del transmisor crea una estructura de datos basada en la trama MAC y la petición de transmisión [45]. Se crea una estructura de datos por cada trama MAC y se envía al medio físico.

Cuando el transmisor envía la señal al medio, se calcula cuándo, dónde y cómo llega la información a los receptores [44]. A continuación, el modelo del medio determina el conjunto de receptores afectados dependiendo del rango de comunicación del transmisor y del modo de operación del transmisor. Con el resultado, el medio envía el mensaje a todos los receptores afectados.

Luego el modelo de atenuación utiliza la señal original transmitida y la señal de llegada para calcular la señal del receptor. Aplica el modelo de pérdida de trayectoria, el modelo de pérdida de obstáculos y el modelo multitrayecto a la transmisión [44].

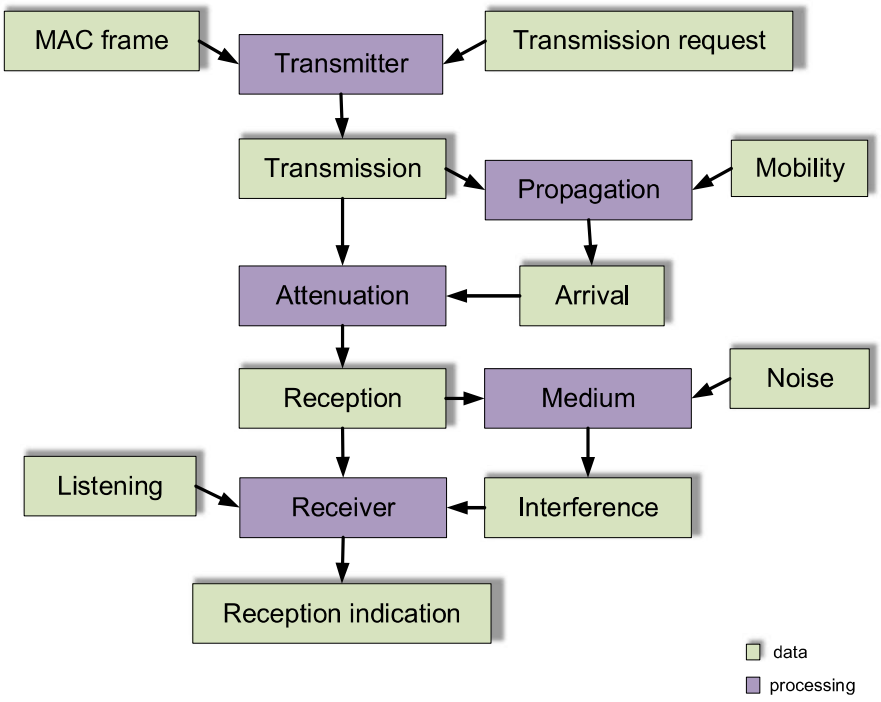

Figura 2.1. Diagrama de Bloques del Proceso de Transmisión y Recepción de INET [49].

Después el modelo de medio utiliza las señales recibidas y los ruidos interferentes para calcula el nivel de interferencia en la recepción. El modelo de ruido de fondo también calcula una señal de ruido la cual se agrega a la interferencia.

El proceso de recepción comienza cuando el radio receptor recibe un mensaje proveniente del radio transmisor. En este proceso el modelo del receptor determina si la recepción es o no es posible dependiendo de la potencia de recepción y otros parámetros. Si la recepción es posible determina si la recepción se intentó o no basándose en si otro proceso de recepción se está realizando y si la captura está habilitada o no.

Si la recepción se intentó se determina si la recepción fue exitosa o no, dependiendo del modelo de error [44]. El modelo de error calcula las tasas de error con la tasa de bits, el esquema demodulación y el resultado obtenido en el cálculo de la SNIR y la interferencia. Alternativamente, el modelo de error puede alternar los datos de la transmisión original para realizar estos cálculos.

El receptor determina el paquete MAC recibido utilizando el original, o decodificando. Finalmente, se agrega la indicación de recepción de capa física al paquete MAC como información de control y se lo envía a la capa superior.

2.2 INETMANET

2.2.1 DESCRIPCIÓN

INETMANET se basa en el framework INET, por lo que proporciona la misma funcionalidad. Sin embargo, contiene protocolos y componentes adicionales que son especialmente útiles al modelar la comunicación inalámbrica principalmente para redes ad hoc móviles.

Entre los principales protocolos y componentes de INETMANET se tienen [45]:

- Modelos de propagación como: Log Normal Shadowing, Nakagami, Rayleigh, Rice, Two Ray, etc.
- Protocolos de capa de enlaces como: 802.11 (a,b,g,n), 802.15.4, 802.16, etc.
- Protocolos de enrutamiento como: OLSR, DSR, AODV, DYMO, etc.
- Modelos de movilidad como: Ramdom Way Point, Gauss Markov, Chiang, TraCI, etc.
- Modelo de consumo de batería.
- Entre otros.

2.2.2 CAPA FÍSICA

Al igual que INET, INETMANET modela la capa física considerando la propagación de la señal en el canal inalámbrico, fading, interferencia y decodificación. Contiene tanto modelos de radio como modelos del medio físico representados a nivel de bit y nivel de paquete [44].

INETMANET está estructurado mediante directorios, su capa física contiene doce directorios como se muestra en la Figura 2.2. Cada directorio constituye un modelo diferente conformado por varios módulos.

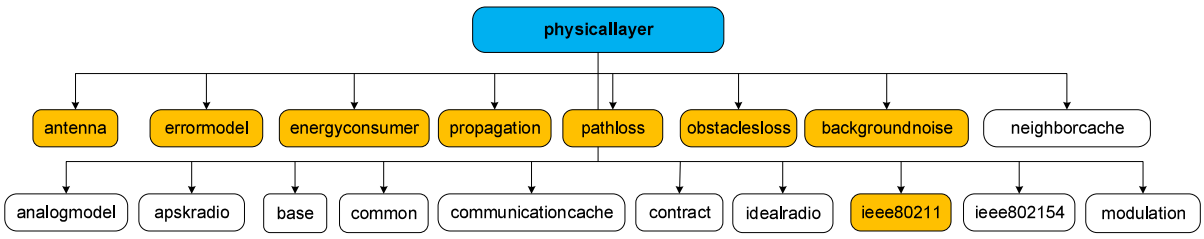

Figura 2.2. Directorios de la capa física de INETMANET [44].

Nos enfocaremos en el estudio de la clase ieee80211 y a partir de ésta, se estudiarán los modelos que este utiliza para representar la capa física del estándar IEEE 802.11a.

2.2.3 MODELO 802.11

El modelo 802.11 implementa una tarjeta interfaz de red NIC que puede actuar como AP, STA o en modo ad hoc y puede incluir QoS. Nos centraremos en el estudio de este modelo 80211 en capa física y sus componentes. Adicionalmente se revisarán modelos que permiten configurar redes en modo ad hoc y describiremos la estructura de la NIC IEEE 802.11.

Los módulos incluidos en el modelo ieee80211 se encuentran organizados en capas, lo que permite el análisis a nivel de dominio de paquete, bit, símbolo, muestra y dominio analógico. Así el modelo iee80211 contiene directorios estructurados como se muestra en la Figura 2.3.

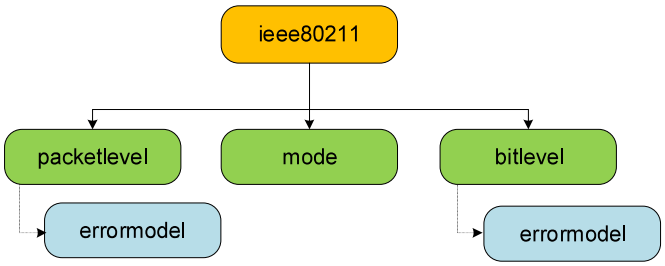

Figura 2.3. Directorio de ieee80211 en INETMANET.

Cada uno de estos directorios contiene varios archivos, los cuales se enumeran en la Figura 2.4.

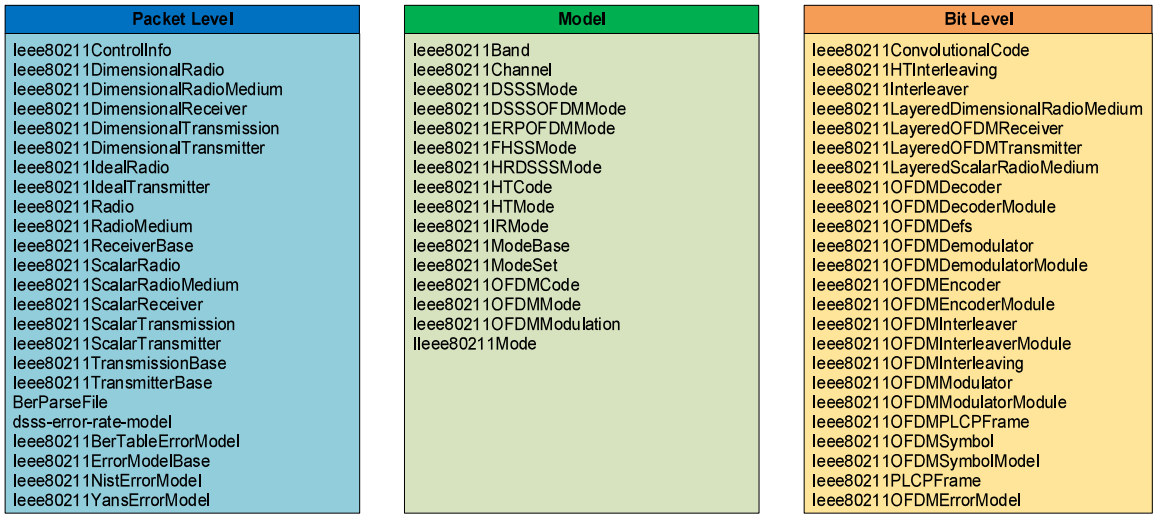

Figura 2.4. Archivos contenidos en los directorios de ieee80211 en INETMANET.

El directorio packet level contiene módulos que reciben y transmiten mensajes desde y hacia la capa MAC y muchos de estos extienden las funcionalidades de módulos simples [45].

El directorio mode contiene módulos que definen los modos de funcionamiento del estándar IEEE 802.11 es decir permite especificar si utiliza OFDM, DSSS, FHSS, bandas de frecuencia y canales [45].

El directorio bitlevel contiene módulos que describen los procesos que se realizan tanto en el trasmisor como en el receptor, lo que incluye codificación/decodificación, scrambling, interleaving, modulación/demodulacion, etc [45].

A continuación, se realizará la descripción de los módulos y mensajes mas importantes que se incluyen en los directorios nombrados.

2.2.4 NIVEL DE PAQUETE (packetlevel)

El directorio packetlevel contiene módulos compuestos que permiten el envío y recepción de mensajes desde y hacia capa MAC. Dichos módulos forman parte tanto de los modelos de radio como del medio físico. Adicionalmente contiene un directorio con modelos de error. Los módulos más importantes se describirán a continuación.

En los modelos del medio físico se tienen tres módulos descritos en la Tabla 2.2.

Tabla 2.2. Modelos de medio físico a nivel de paquete [44].

Clase	Descripción	Atributos
Ieee80211 Dimensional RadioMedium	Módulo compuesto que utiliza potencia de transmisión dimesional, la misma que cambia con el tiempo y/o frecuencia. Trabaja en conjunto con Ieee80211DimensionalRadio y extiende Ieee80211RadioMedium.	analogModelType backgroundNoiseType
Ieee80211 RadioMedium	Es un módulo compuesto que se utiliza junto con Ieee80211Radio. Este módulo hereda parámetros del medio físico de RadioMedium.	propagationType pathLossType backgroundNoise.power mediumLimitCache.carrierFrequency mediumLimitCache.minReceptionPower mediumLimitCache.minInterferencePower
Ieee80211 Scalar RadioMedium	Módulo compuesto que se usa en conjunto con el módulo Ieee80211ScalarRadio. Hereda atributos de Ieee80211RadioMedium y configura el modelo analógico y tipo de ruido de fondo	analogModelType backgroundNoiseType

En los modelos de radio se tienen varios módulos y dentro de los mismos de definen modelos para el transmisor y el receptor, los cuales se describen en la Tabla 2.3.

Tabla 2.3. Modelos de radio a nivel de paquete [44].

Clase	Descripción	Atributos
Ieee80211 Dimensional Radio	Constituye un módulo compuesto que utiliza potencia de transmisión dimensional en el dominio analógico, lo que significa que la potencia va a cambiar con el tiempo y/o frecuencia. Trabaja junto al modelo Ieee80211DimensionalRadioMedium	transmitterType recieverType
Ieee80211 IdealRadio	Es un módulo compuesto que utiliza el dominio analógico para su representación y trabaja en conjunto con el módulo IdealRadioMedium.	transmitterType receiverType
Ieee80211 Radio	Es un módulo compuesto que se usa junto con Ieee80211RadioMedium. Permite el uso de múltiples canales y diferentes modos de operación. Contiene submódulos de antena, transmisor, receptor y modelos de error, que permiten definir el tipo de submódulo a utilizarse.	carrierFrequency Bandwidth antennaType transmitter.preamble Duration transmitter.bitrate transmitter.headerBit Length transmitter.power receiver.sensitivity receiver.energyDetectio n receiver.snirThreshold
Ieee80211 ScalarRadio	Módulo compuesto que utiliza el dominio analógico y una potencia de transmisión escalar. Trabaja en conjunto con Ieee80211ScalarRadioMedium.	transmitterType recieverType
Ieee80211 Dimensional Receiver	Constituye un modelo de receptor que recibe información exitosamente cuando se tiene una SINR que sobrepasa el valor mínimo. El SINR y el modo de operación permiten calcular la tasa de error dependiendo del modelo de error utilizado. Extiende Ieee80211ReceiverBase	computeIsReceptionPos sible
Ieee80211 Dimensional Transmitter	Es un módulo compuesto que constituye un modelo de transmisión que envía información usando una potencia de transmisión dimensional	opMode bandName channelNumber modulation
Ieee80211 Ideal Transmitter	Módulo compuesto que es parte de los módulos de transmisor. Este módulo hereda atributos de Ieee80211TransmitterBase	communicationRange interferenceRange detectionRange
Ieee80211 Receiver Base	Constituye un modelo de receptores y un módulo compuesto. Éste sirve de base para los receptores y permite configurar diferentes modos de operación con diferentes preámbulos y también conmutación de canales. Adicionalmente define la modulación y el modelo de error	modulation errorModelType

Clase	Descripción	Atributos
Ieee80211 Scalar Receiver	Módulo compuesto que al igual que Ieee80211DimensionalReceiver recibe información exitosamente si se sobrepasa el límite SNIR y utiliza modelos de error para calcular la tase de error. Adicionalmente, extiende el módulo Ieee80211ReceiverBase.	_____
Ieee80211 Scalar Transmitter	Módulo compuesto que trabaja con una potencia de transmisión escalar. Hereda atributos de Ieee80211TransmitterBase	_____
Ieee80211 Transmitter Base	Constituye un módulo compuesto que sirve como base para modelar transmisores en diferentes modos de operación. Utiliza los mismos parámetros definidos en Ieee80211DimensionalTransmitter.	opMode bandName channelNumber modulation

En cuanto a los modelos de error, a nivel de paquete se tienen cuatro módulos contenidos en un directorio denominado errormodel, los cuales se indican en la Tabla 2.4. Estos modelos de error permiten calcular valores como BER y PER para distintas modulaciones.

Tabla 2.4. Modelos de error a nivel de paquete [44].

Clase	Descripción
Ieee80211BerTable ErrorModel	Junto con BerParseFile, permite crear una tabla en base a SNR, BER, PER. Hereda atributos de ErrorModelBase y permite configuración del modo de operación con opMode y un archivo que contiene la tabla de BER berTableFile
Ieee80211ErrorModel Base	Permite calcular Packet Error Rate, Bit Error Rate, Symbol Error Rate.
Ieee80211NistError Model	Módulo compuesto que constituye un modelo de error para modulaciones BPSK, QPSK, 16QAM. Permite el cálculo del PER y extiende ErrorModelBase.
Ieee80211YansError Model	Modulo compuesto que permite calcular el PER para BPSK y QAM, extendiendo ErrorModelBase.

2.2.5 MODO DE OPERACIÓN (mode)

Este directorio contiene varios módulos que permiten definir los modos de operación para el estándar IEEE 802.11, de los modos existentes nos interesa los que permiten definir OFDM, las bandas de frecuencia y canales a utilizarse. Estos módulos los describimos en la Tabla 2.5.

Tabla 2.5. Modos de operación [44].

Clase	Descripción
Ieee80211Band	Permite definir la banda de frecuencia para 2.4 GHz, 5GHz y 5.9GHz, así como el número de canales y la frecuencia central.
Ieee80211Channel	Obtiene el canal mediante la banda y número de canal.

Clase	Descripción
Ieee80211ModeBase	Obtiene el nombre del modo de operación.
Ieee80211ModeSet	Configura los modos de operación para el estándar IEEE 802.11.
Ieee80211OFDMCode	Define el uso de código convolucional, scrambling e interleaving.
Ieee80211OFDMMode	Define modos de preámbulo y datos de OFDM.
Ieee80211OFDMModulation	Define el tipo de modulacion utilizado en OFDM.
IIeee80211Mode	Obtiene todos los valores de los parámetros definidos en los modos de operación de IEEE 802.11.

2.2.6 NIVEL DE BIT (bitlevel)

En este directorio se encuentran modelos de radio tanto para el transmisor como el receptor específicamente para OFDM. Adicionalmente contiene modelos de medio físico y varios módulos para otras configuraciones en OFDM. A continuación, se describirán los módulos que contiene cada uno de los modelos indicados.

2.2.6.1 Transmisor

En el caso del transmisor la clase Ieee80211LayeredOFDMTransmitter define un módulo compuesto que contiene varios parámetros que cuyos valores serán definidos por los tipos de submódulos utilizados. Estos parámetros se describen en la Tabla 2.6:

Tabla 2.6. Atributos de Ieee80211LayeredOFDMTransmitter [50].

Nombre	Tipo	Descripción
levelOfDetail	string	Permite definir el nivel de detalle ya sea "packet","bit","symbol","sample". Por defecto symbol.
signalEncoderType	string	Define el tipo de codificador de señal utilizado. Para trabajar con el estándar se debe utilizar Ieee80211OFDMEncoder.
dataEncoderType	string	Define el tipo de codificador de datos utilizado. Para trabajar con el estándar se debe utilizar Ieee80211OFDMEncoder
signalModulatorType	string	Define el tipo de modulador de señal utilizado. Para trabajar con el estándar se debe utilizar Ieee80211OFDMModulator
dataModulatorType	string	Define el tipo de modulador de datos utilizado. Para trabajar con el estándar se debe utilizar Ieee80211OFDMModulator
pulseShaperType	string	Define el tipo de moldeador de pulso utilizado.
digitalAnalogConverterType	string	Define el tipo de conversor analógico/digital utilizado.
channelSpacing	double	Define el espaciado de canal
Power	double	Define la potencia de transmisión
carrierFrequency	double	Define la frecuencia de portadora
Bandwidth	double	Define el ancho de banda

Dependiendo del nivel de detalle especificado el transmisor realiza las siguientes funciones:

Tabla 2.7. Funciones de Ieee80211LayeredOFDMTransmitter [49].

Nivel de detalle	Funciones
Bit	Codificación de datos y codificación de señal.
Símbolo	Modulación de datos y señal, codificación de datos y codificación de señal.
Muestra	Moldeador de pulso, codificación de datos y codificación de señal, modulación de datos y señal y conversión A/D.

Todos estos procesos indicados son incluidos con los submodulos signalEncoder, dataEncoder, dataModulator, signalModulator, pulseShaper, digitalAnalogConverter como se muestra en la Figura 2.5.

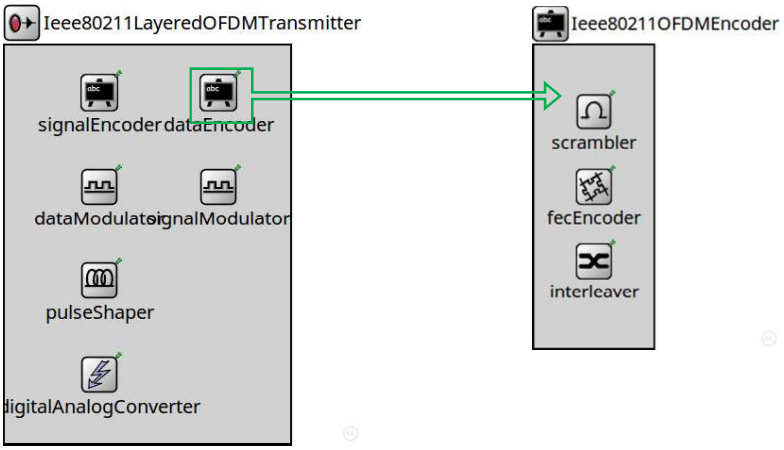

Figura 2.5. Submódulos de Ieee80211LayeredOFDMTransmitter.

También se tiene módulos que se encuentran incluidos dentro del proceso de transmisión como son codificación, interleaving y modulación. Todos estos módulos describiremos brevemente a continuación (véase Tabla 2.8).

Tabla 2.8. Módulos para transmisión [50].

Clase	Descripción
Ieee80211OFDM Encoder	Clase que define a este módulo compuesto que se encarga de la codificación a nivel de OFDM. Como se muestra en la Figura 2.5, contiene tres importantes submodulos: fecEncoder, scrambler, interleaver, los cuales se encargan de realizar la codificación convolucional, scrambling/descrambling e interleaving respectivamente.

Clase	Descripción
Ieee80211OFDM EncoderModule	Clase que permite inicializar el módulo Ieee80211OFDMEncoder el tipo de codificador convolucional, interleaving y scrambling.
Ieee80211 ConvolutionalCode	Define tasas de codificación para el código convolucional.
Ieee80211 Interleaver	Módulo compuesto que para realizar la permutación primero se cerciora que bits codificados de subportadoras adyacentes se mapeen en subportadoras no adyacentes y luego verifica que esos bits se mapeen en bits menos y más significativos de la constelación. Define dos variables tipo int: numberOfCodedBitsPerSymbol, numberOfCodedBitsPerSubcarrier.
Ieee80211HT Interleaving	Inicializa el objeto creado para interleaving de High Throughput OFDM.
Ieee80211OFDM Interleaver	Clase que define al módulo compuesto del mismo nombre que se encarga del interleaving. Para esto primero realiza la permutación que asegura que los bits codificados adyacentes se mapeen en subportadoras no adyacentes y luego realiza la permutación que se asegura que los bits codificados adyacentes se mapeen en bits menos y más significativos. Contiene dos parámetros: numberOfCodedBitsPerSymbol que define el número de bits codificados por símbolo y numberOfCodedBitsPerSubcarrier que define el numedo de bits codificados por subportadora.
Ieee80211OFDM InterleaverModule	Clase que permite inicializar Ieee80211OFDMInterleaving con numberOfCodedBitsPerSymbol y numberOfCodedBitsPerSubcarrier.
Ieee80211OFDM Interleaving	Clase que inicializa numberOfCodedBitsPerSymbol y numberOfCodedBitsPerSubcarrier.
Ieee80211OFDM Modulator	Clase que define al módulo compuesto del mismo nombre, realiza la modulación de subportadoras OFDM con las modulaciones BPSK, QPSK, 16-QAM, 64-QAM. Contiene el parámetro subcarrierModulation para definir el tipo de modulación y pilotSubcarrierPolarityVectorOffset para definir el desfase.
Ieee80211OFDM ModulatorModule	Clase que permite inicializar Ieee80211OFDMModulation con el tipo de modulación de subportadora subcarrierModulation y pilotSubcarrierPolarityVectorOffset.
Ieee80211OFDM Symbol	Clase que obtiene los símbolos OFDM formados.
Ieee80211OFDM SymbolModel	Clase que inicializa Ieee80211OFDMSymbolModel TransmissionSymbolModel en el transmisior y ReceptionSymbolModel en el receptor como se muestra a continuación TransmissionSymbolModel(headerSymbolLength, headerSymbolRate, payloadSymbolLength, payloadSymbolRate, symbols, headerModulation, payloadModulation) ReceptionSymbolModel(headerSymbolLength, headerSymbolRate, payloadSymbolLength, payloadSymbolRate, symbols)

2.2.6.2 Receptor

En el caso del transmisor la clase Ieee80211LayeredOFDMReceiver esta definida para OFDM, la cual constituye un módulo compuesto por varios submódulos a nivel de radio. Esta clase contiene los parámetros listados en laTabla 2.9.

Tabla 2.9. Parámetros de Ieee80211LayeredOFDMReceiver [50].

Nombre	Tipo	Descripción
levelOfDetail	string	Permite definir el nivel de detalle ya sea "packet","bit","symbol","sample". Por defecto symbol.
errorModelType	string	Define el tipo de modelo de error. Para el estándar es posible utilizar el modelo Ieee80211OFDMErrorModel
signalDecoderType	string	Define el tipo de decodificador de señal utilizado. Para trabajar con el estándar se debe utilizar Ieee80211OFDMDecoder.
dataDecoderType	string	Define el tipo de decodificador de datos utilizado. Para trabajar con el estándar se debe utilizar Ieee80211OFDMDecoder
signalDemodulatorType	string	Define el tipo de demodulador de señal utilizado. Para trabajar con el estándar se debe utilizar Ieee80211OFDMDemodulator
dataDemodulatorType	string	Define el tipo de demodulador de datos utilizado. Para trabajar con el estándar se debe utilizar Ieee80211OFDMDemodulator
pulseFilterType	string	Define el tipo de filtro de pulso.
analogDigitalConverterType	string	Define el conversor D/A
energyDetection	double	Define el nivel de energía que puede ser detectado
Sensitivity	double	Define la sensibilidad
carrierFrequency	double	Define la potencia de portadora
Bandwidth	double	Define el ancho de banda
snirThreshold	double	Define el umbral SNIR
channelSpacing	double	Define el espaciado de canal

Como se muestra en la Figura 2.6, se definen varios submodulos como errorModel, dataDecoder, signalDecoder, signalDemodulator, dataDemodulator, pulseFilter, analogDigitalConverter. El decodificador contiene los submódulos deinterleaver, fecDecoder y decrambler.

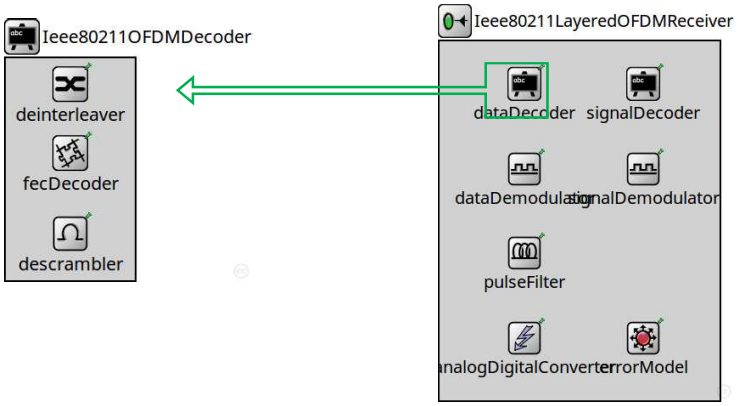

Figura 2.6. Submódulos de Ieee80211LayeredOFDMReceiver.

Adicionalmente, se tienen varios módulos que pueden definirse en el receptor, como son decodificador, demodulador y modelos de error. Todos estos constituyen módulos compuestos y se definen en la Tabla 2.16.

Tabla 2.10. Módulos para el receptor [50].

Clase	Descripción
Ieee80211OFDM Decoder	Clase que define a este módulo compuesto que se encarga de la codificación a nivel de OFDM. Como se muestra en la Figura 2.6, contiene tres importantes submodulos: fecEncoder, scrambler, interleaver, los cuales se encargan de realizar la codificación convolucional, scrambling/descrambling e interleaving respectivamente.
Ieee80211OFDM DecoderModule	Clase que permite inicializar el módulo Ieee80211OFDMEncoder el tipo de codificador convolucional, interleaving y scrambling.
Ieee80211OFDM Demodulator	Define tasas de codificación para el código convolucional.
Ieee80211OFDM DemodulatorModule	Módulo compuesto que para realizar la permutación primero se cerciora que bits codificados de subportadoras adyacentes se mapeen en subportadoras no adyacentes y luego verifica que esos bits se mapeen en bits menos y más significativos de la constelación. Define dos variables tipo int: numberOfCodedBitsPerSymbol, numberOfCodedBitsPerSubcarrier.
Ieee80211OFDM ErrorModel	Clase utilizada para definir el módulo compuesto del mismo nombre. Este módulo contiene el modelo de error utilizado en OFDM, el cual produce bits errados o determina si la información que recibe es errada. Contiene los parámetros dataSymbolErrorRate, dataBitErrorRate, signalSymbolErrorRate, signalBitErrorRate.

2.2.6.3 Medio físico

En el caso de los modelos de medio físico a nivel de bit, se tienen solamente dos módulos: uno dimensional y otro escalar. Estos dos modelos se encuentran organizados en capas y definen parámetros de propagación y pérdidas en el espacio. A continuación, describimos brevemente estos modelos.

Tabla 2.11. Modelos de medio físico a nivel de bit [44].

Clase	Descripción	Atributos
Ieee80211Layered DimensionalRadio Medium	Módulo compuesto que constituye un modelo del medio físico organizado en capas.	propagationType pathLossType analogModelType backgroundNoiseType backgroundNoise.power mediumLimitCache.carrierFrequency mediumLimitCache.minReceptionPower mediumLimitCache.minInterferencePower

Clase	Descripción	Atributos
Ieee80211Layered ScalarRadioMedium	Define los mismos valores de Ieee80211RadioMedium y contiene adicionalmente un modelo de capas para el modelo analógico.	propagationType pathLossType backgroundNoise.power mediumLimitCache.carrierFrequency mediumLimitCache.minReceptionPower mediumLimitCache.minInterferencePower analogModelType

2.2.6.4 Parámetros OFDM

A nivel de bit, se tiene un módulo que permite definir valores para el estándar IEEE 802.11 con OFDM denominado Ieee80211OFDMDefs. En el cual se definen números de subportadoras de datos, longitud del campo de señal codificado y decodificado, el inicio y fin de campo de señal, inicio y fin del campo velocidad de señal, el campo paridad, longitud del campo servicio y cola de la PPDU. Todos los valores relacionados con en inicio y fin de un campo en la trama PLCP indican su posición en dicha trama.

En la Tabla 2.12 se muestran los parámetros de configuración de Ieee80211OFDMDefs.

Tabla 2.12. Parámetros definidos en Ieee80211OFDMDefs.

Parámetro	Valor
NUMBER_OF_OFDM_DATA_SUBCARRIERS	48
DECODED_SIGNAL_FIELD_LENGTH	24
ENCODED_SIGNAL_FIELD_LENGTH	48
SIGNAL_RATE_FIELD_START	0
SIGNAL_RATE_FIELD_END	3
SIGNAL_LENGTH_FIELD_START	5
SIGNAL_LENGTH_FIELD_END	16
SIGNAL_PARITY_FIELD	17
PPDU_SERVICE_FIELD_BITS_LENGTH	16
PPDU_TAIL_BITS_LENGTH	6

2.2.7 OTROS MODELOS DE RADIO

2.2.7.1 Modelos de antena

Contiene módulos compuestos que describen las características de varias antenas, INETMANET ofrece siete tipos de antena:

- ConstantGainAntenna
- CosineAntenna
- DipoleAntenna
- InterpolatingAntenna

- IsotropicAntenna
- ParabolicAntenna
- PhaseArray

2.2.7.2 Modelos de error

Además de los módulos incluidos en el modelo ieee80211 se tiene el modelo StochasticErrorModel tanto para nivel de bit como para nivel de paquete.

2.2.7.3 Modelos de consumo de energía

Inetmanet contiene el modelo de consumo de energía StateBasedEnergyConsumer, en el cual la potencia se encuentra determinada por el modelo de radio, estado del transmisor y del receptor.

2.2.8 MODELOS DEL MEDIO FÍSICO

2.2.8.1 Modelos de pérdida de camino (pathloss)

Entre los modelos que contiene INETMANET para describir la pérdida de potencia por propagarse en el canal inalámbrico se tienen [49]:

- BreakpointPathLoss: implementa el modelo Break Point y realiza el cálculo de la perdida de potencia y la distancia máxima en la cual se tiene la pérdida mencionada.
- FreeSpacePathLoss: implementa el modelo de pérdidas en el espacio libre y realiza el cálculo de la perdida de potencia y la distancia.
- LogNormalShadowing: implementa el modelo Log-Normal y realiza el cálculo de la perdida de potencia a una distancia determinada incluyendo la distribución log-normal.
- NakagamiFading: implementa el modelo de Nakagami y realiza el cálculo de la perdida de potencia.
- RayleighFading: implementa el modelo de Rayleigh y realiza el cálculo de la perdida de potencia.
- RicianFading: implementa el modelo de Rician y realiza el cálculo de la perdida de potencia.
- SUIPathLoss: implementa el modelo de SUI y realiza el cálculo de la perdida de potencia.
- TwoRayGroundReflection: implementa el modelo de dos rayos y realiza el cálculo de la perdida de potencia.

– UWBIRStochasticPathLoss: implementa el modelo estocástico UWB y realiza el cálculo de la perdida de potencia y rango.

2.2.8.2 Modelos de propagación (propagation)

Existen dos modelos de propagación en este framework [50]:

– ConstantSpeedPropagation: modelo que calcula el tiempo de propagación proporcional a la distancia teniendo una velocidad de propagación constante.
– ConstantTimePropagation: calcula el tiempo de propagación sin dependencia de la distancia.

2.2.8.3 Modelos de pérdida por obstáculos (obstacleloss)

El framework estudiado contiene dos modelos de perdida por obstáculos, uno ideal y otro dieléctrico los cuales son IdealObstacleLoss y DielectricObstacleLoss respectivamente. Estos dos modelos realizan el cálculo de la pérdida de potencia debida a obtáculos.

2.2.8.4 Modelos analógicos (analoguemodel)

A nivel de paquete contiene un modelo analógico dimensional y uno escalar. El modelo DimensionalAnalogModel se forma por DimensionalNoise, DimensionalReception, DimensionalSNIR, DimensionalTransmission. El modelo ScalarAnalogModel contiene ScalarNoise, ScalarReception, ScalarSNIR, ScalarTransmission. Estos modelos calculan la potencia de la señal en recepción, nivel de potencia del ruido e interfencias, y la relación SNIR ya sea a nivel escalar o dimensional.

A nivel de bit tenemos DimensionalSignalAnalogModel, LayeredDimensionalAnalogModel, LayeredScalarAnalogModel, SignalAnalogModel, ScalarSignalAnalogModel. Estos modelos retornan la duración, frecuencia, ancho de banda y nivel de potencia de las señales.

2.2.8.5 Modelos de ruido de fondo (BackgroundNoise)

El framework estudiado contiene dos modelos de ruido de fondo los cuales son IsotropicScalarBackgroundNoise y IsotropicDimensionalBackgroundNoise. Los cuales generan ruido en el canal que se usan en cálculos escalares y dimensionales respectivamente.

2.2.9 MENSAJES

2.2.9.1 Ieee80211ControlInfo

Mensaje a nivel de paquete. Constituye información de control que es enviada a Radio o Ieee80211Radio. Esta información de control enviada al radio extiende RadioControlInfo.msg y depende del tipo de comando enviado: configuración, petición de transmisión o indicación de recepción [50].

Cuando se envía un comando de configuración la información de control enviada a ~Radio es la que se indica en la Tabla 2.13.

Tabla 2.13. Opciones de Ieee80211ControlInfo [50].

Clase	Nombre	Descripción
Ieee80211Configure RadioCommand	opMode	Variable tipo string que define el modo de operación
	modeSet	Especifica el modo a,b,g,p o n
	mode	Define el modo de transmisión
	band	Especifica la banda de tranajo
	channel	Especifica el canal
	channelNumber	Especifica el número de canales
ConfigureRadio Command	radioMode	Define el modo de radio
	power	Especifica la potencia de transmisión en W
	bitrate	Especifica la tasa de bit en bps
	modulation	Especifica el tipo de modulación
	carrierFrequency	Especifica la frecuencia de la portadora en Hz
	bandwidth	Especifica el ancho de banda en Hz

Cuando se envía una petición de transmisión la información de control enviada a ~Ieee80211Radio es la que se indica en la Tabla 2.14.

Tabla 2.14. Mensajes de control de Ieee80211ControlInfo [50].

Clase	Nombre	Descripción
Ieee80211TransmissionRequest	mode	Define el modo de transmisión
	channelNumber	Especifica el numero de canales
	channel	Especifica el canal
TransmissionRequest	power	Especifica la potencia de transmisión en W
	bitrate	Especifica la tasa de bit en bps
	carrierFrequency	Especifica la frecuencia de la portadora en Hz
	bandwidth	Especifica el ancho de banda en Hz

Cuando se envía una indicación de recepción la información de control enviada a ~Ieee80211Radio es la que se indica en la Tabla 2.15.

Tabla 2.15. Mensajes de confirmación de recepción de mensajes de control [50].

Clase	Nombre	Descripción
Ieee80211Reception Indication	mode	Especifica el modo de recepción
	channel	Especifica el canal
	snr	Define la minima relación señal a ruido
	lossRate	Define la velocidad de pérdida
	recPow	Define la potencia de recepción
	airtimeMetric	Define la métrica para airtime
	testFrameDuration	Define la duración de la trama de prueba
	testFrameError	Define la tasa de error de la trama de prueba
	testFrameSize	Define el tamaño de la trama de prueba
ReceptionIndication	bitErrorCount	Especifica el número de bits errados
	symbolErrorCount	Especifica el número de símbolos errados
	packetErrorRate	Especifica la tasa de error de paquetes en probabilidad de 0 a 1
	bitErrorRate	Especifica la tasa de error de bits en una probabilidad de 0 a 1
	symbolErrorRate	Especifica la tasa de error de símbolos en probabilidad de 0 a 1
	minRSSI	Especifica el mínimo RSSI (Recieve Signal Strength Indication)
	minSNIR	Especifica la mínima SNIR (Signal to Noise Interference Ratio)

2.2.9.2 Ieee80211PLCPFrame

Define el tipo de trama PLDCP que se formará, utiliza determinados valores para los diferentes modos de operación:

OFDM = 0;
DSSS = 1;
Infrared = 2;
FHSS = 3;
HRDSSS = 4;
ERP = 5;
HT = 6;

2.2.9.3 Ieee80211OFDMPLCPFrame

Define el formato de la trama PLCP para OFDM una vez que el tipo OFDM es definido en Ieee80211PLCPFrame.

Contiene los campos especificados en la Tabla 2.16:

Tabla 2.16. Campos de trama Ieee80211ODFMPLCPFrame .

Campo	Longitud	Descripción
type		Define tipo OFDM
rate	4 bits	Indica el número de octetos
length	12 bits	Contiene información sobre codificación y modulación

Los campos parity, reserved, service y pad bits no se representan explícitamente. La PSDU se construye como un paquete encapsulado y en el momento de formación de esta trama PLCP se toman en cuenta los campos faltantes.

Adicionalmente se define en los serializadores de Ieee802.11 que la cabecera de la trama PLCP tiene los campos: rate, reserved, length, parity, tail y service.

2.2.10 MÓDULO ADHOCHOST

Constituye un módulo compuesto que modela un host inalámbrico que realiza funciones como enrutamiento, movilidad y almacenamiento de energía y otras funcionalidades de las capas de aplicación, transporte, red, MAC y física [45].

Para configurar este host en modo ad hoc se define wlan[*].mgmtType como Ieee80211MgmtAdhoc. Como se muestra en la figura Figura 2.7 adhochost constituye un módulo compuesto formado por módulos de energía, movilidad, NICS, aplicación, transporte y red. Cada uno de estos módulos puede configurarse con ciertos protocolos definidos en cada capa de la arquitectura TCP/IP.

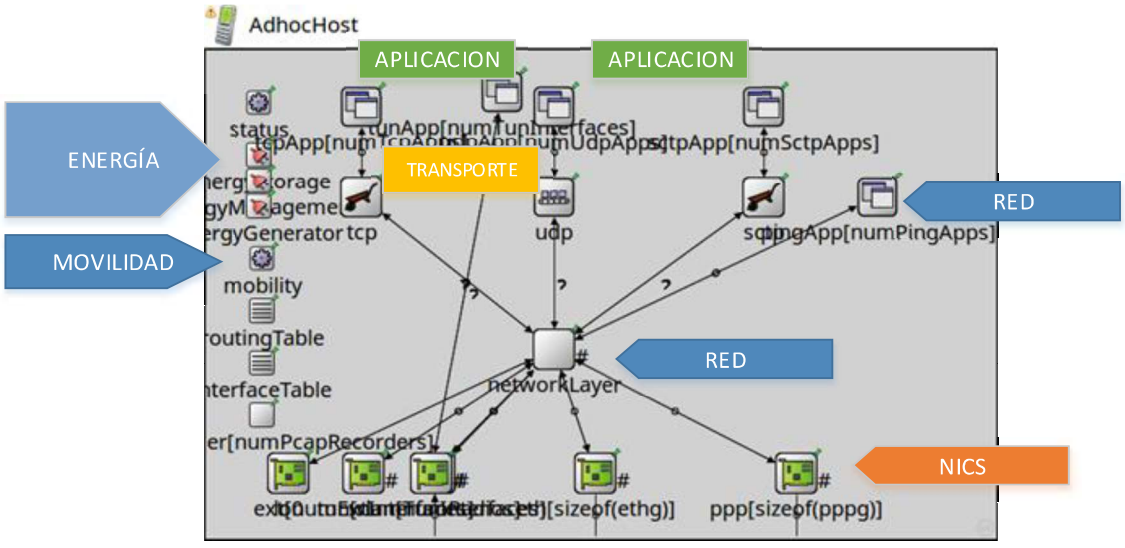

Figura 2.7. Módulo AdhocHost.

Permite configurar en capa de aplicación aplicaciones TCP/UDP, en capa de transporte los protocolos TCP y UDP, en capa de red soporta IPv4, ICMP y protocolos de enrutamiento ad hoc. Para capa MAC y física permite configurar la tarjeta de red NIC con IEEE 802.11, IEEE802.15.4, tipo ideal, etc. Esta descripción se representa en la siguiente figura.

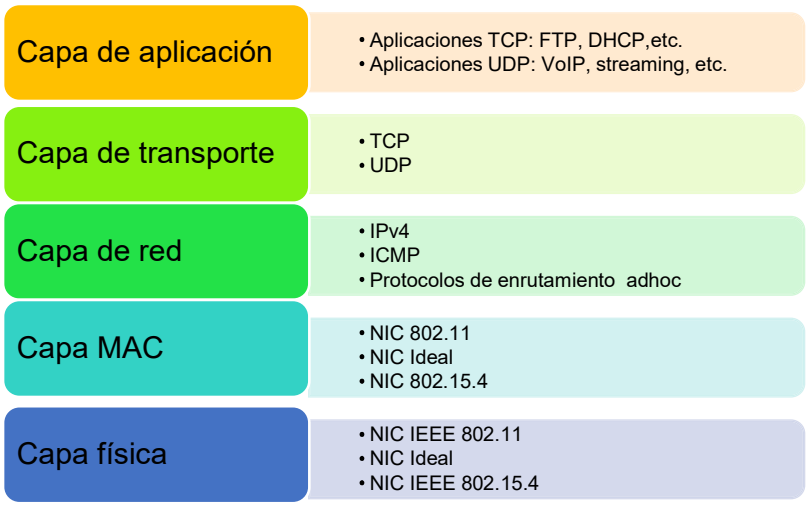

Figura 2.8. Arquitectura de Módulo AdhocHost [45].

El módulo adicionalmente hereda atributos de algunas clases como se indica en la Tabla 2.17.

Tabla 2.17. Atributos de Ieee80211MgmtAdhoc heredados de otras clases [50].

Clase	Atributos	Descripción
AdhocHost	wlan[*].mgmtType	Definido como Ieee80211MgmtAdhoc para trabajo en modo ad hoc
	forwarding	Definido como true
WirelessHost	numRadios	Define el número de radios, por defecto 1
StandardHost	numTcpApps	Define el tipo de aplicación para TCP.
	numSctpApps	Define el tipo de aplicación para UCP.
	numPingApps	Define el tipo de aplicación para TCP.
	hasTcp, hasUdp, hasSctp	Define cuantas transmisiones fueron realizadas
	tcpType	Define el tipo de paquete TCP
	udpType	Define el tipo de paquete UCP
	sctpType	Define el tipo de paquete SCTP
	networkLayer.proxyARP	Define si el proxy arp esta ejecutándose
NodeBase	numExtInterfaces	Define el número de interfaces externas
	numRadios	Define el número de radios
	numPcapRecorders	Define el número de Pcap Recorders
	numTunInterfaces	Define el número de interfaces Tun
	osgModel	Define el modelo para la visualización OSG
	osgModelColor	Define el color para visualización OSG
	mobilityType	Define el tipo de movilidad

Clase	Atributos	Descripción
	networkLayerType	Define el tipo de protocolo de capa de red
	routingTableType	Define la tabla de enrutamiento a formarse según el tipo de protocolo de red
	energyStorageType	Define el tipo de almacenamiento de energía
	energyManagementType	Define el tipo de gestion almacenamiento de energía
	energyGeneratorType	Define el tipo de generador para almacenamiento de energía

2.2.11 MÓDULO IEEE80211NIC

Este módulo permite configurar AdhocHost con el estándar IEEE 802.11 y sus diferentes modos de operación. En este módulo la NIC se forma de cuatro capas como se muestra en la Figura 2.9 .

Capa fisica (physical layer): modela la transmisión y recepción de tramas, características del canal de radio y determina si la trama se recibió correctamente [51].

Capa MAC: trasmite tramas de acuerdo con el protocolo CSMA/CA y recibe información de capas superiores para luego transmitirla [51].

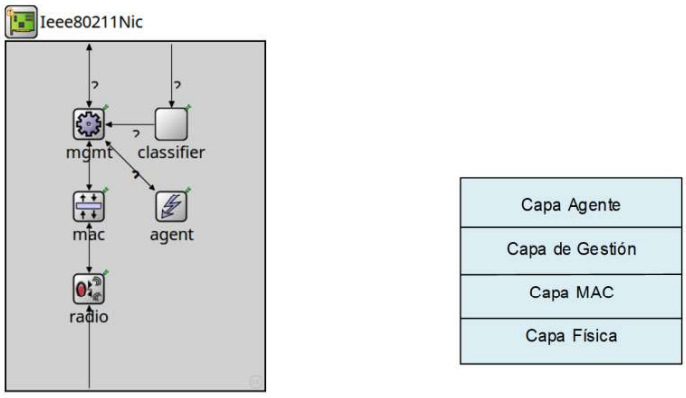

Figura 2.9. Arquitectura de módulo Ieee80211NIC.

Capa de Gestión (management): realiza la encapsulación y descapsulación de paquetes de datos para la MAC e intercambia tramas de gestión [51].

Capa Agente (agent): realiza escaneo, la autenticación y la asociación. Sólo está presente en el Ieee80211Nic con mgmtType = Ieee80211MgmtSTA

La NIC puede trabajar en varias formas dependiendo de cómo se configure mgmtType: [51]

- Ieee80211Nic: un NIC genérico (configurable)
- Ieee80211Nic con mgmtType = Ieee80211MgmtAdhoc: para modo ad hoc que es de nuestro interés.
- Ieee80211Nic con mgmtType = Ieee80211MgmtAP o Ieee80211MgmtAPSimplified: para usar en un punto de acceso
- Ieee80211Nic con mgmtType = Ieee80211MgmtSTA, Ieee80211MgmtSTASimplified: para usar en una estación en modo de infraestructura [2].

La NIC modela la capa física en el submódulo de radio y para el estándar definido se lo configura como Ieee80211ScalarRadio. Para la configuración del modo de operación es posible seleccionar entre a,b,g(erp),g(mixed), n y p.

Como previamente se indicó el modelo de radio contiene modelos de transmisor, receptor, antena y modelos de energía. Para configurar este módulo de acuerdo con el estándar IEEE 802.11a se requiere utilizar los módulos que utilizan OFDM indicando que el modo de operación será "a". Así se requiere configurar los valores tanto para el transmisor como para el receptor:

Tabla 2.18. Configuración transmisor y receptor

Elemento	Parámetro	Valor
Transmisor	**.wlan[*].radio.transmitterType	Ieee80211LayeredOFDMTransmitter
	**.wlan[*].radio.transmitter.channelSpacing	20MHz
	**.wlan[*].radio.transmitter.signalEncoderType	Ieee80211OFDMEncoder
	**.wlan[*].radio.transmitter.signalModulatorType	Ieee80211OFDMModulator
	**.wlan[*].radio.transmitter.dataEncoderType	Ieee80211OFDMEncoder
	**.wlan[*].radio.transmitter.dataModulatorType	Ieee80211OFDMModulator
	**.wlan[*].radio.transmitter.dataEncoder.fecType	ConvolutionalCoder
	**.wlan[*].radio.transmitter.dataEncoder.scramblerType	AdditiveScrambler
	**.wlan[*].radio.transmitter.dataEncoder.interleaverType	Ieee80211OFDMInterleaver
	**.wlan[*].radio.transmitter.signalEncoder.fecType	ConvolutionalCoder
	**.wlan[*].radio.transmitter.signalEncoder.scramblerType	
	**.wlan[*].radio.transmitter.signalEncoder.interleaverType	Ieee80211OFDMInterleaver
Receptor	**.wlan[*].radio.receiverType	Ieee80211LayeredOFDMReceiver
	**.wlan[*].radio.receiver.dataDecoderType	Ieee80211OFDMDecoder
	**.wlan[*].radio.receiver.errorModelType	Ieee80211OFDMDecoder
	**.wlan[*].radio.receiver.signalDecoderType	Ieee80211OFDMDecoder
	**.wlan[*].radio.receiver.signalDemodulatorType	Ieee80211OFDMDemodulator
	**.wlan[*].radio.receiver.dataDemodulatorType	Ieee80211OFDMDemodulator
	**.wlan[*].radio.receiver.dataDecoder.fecType	ConvolutionalCoder

Elemento	Parámetro	Valor
	**.wlan[*].radio.receiver.dataDecoder.descramblerType	AdditiveScrambler
	**.wlan[*].radio.receiver.dataDecoder.deinterleaverType	Ieee80211OFDMInterleaver
	**.wlan[*].radio.receiver.signalDecoder.fecType	ConvolutionalCoder
	**.wlan[*].radio.receiver.signalDecoder.descramblerType	
	**.wlan[*].radio.receiver.signalDecoder.deinterleaverType	Ieee80211OFDMInterleaver

Una vez establecido estos valores se ejecutará el estándar 802.11a requerido para nuestro estudio.

2.2.12 PROCESO DE TRANSMISIÓN/RECEPCIÓN

Los procesos de transmisión y recepción son posibles gracias a la clase Ieee80211Radio. La Figura 2.10 describe la relación de herencia entre las subclases Ieee80211DimensionalRadio, Ieee80211IdealRadio, Ieee80211ScalarRadio y Radio. El modelo Ieee80211Radio constituye una clase que hereda atributos de FlatRadioBase y consecuentemente de NarrowbandRadioBase. Esta última clase extiende funcionalidades de Radio que es la clase más importante la cual nos interesa estudiar ya que la misma contiene todos los métodos que realizan el proceso de transmisión y recepción.

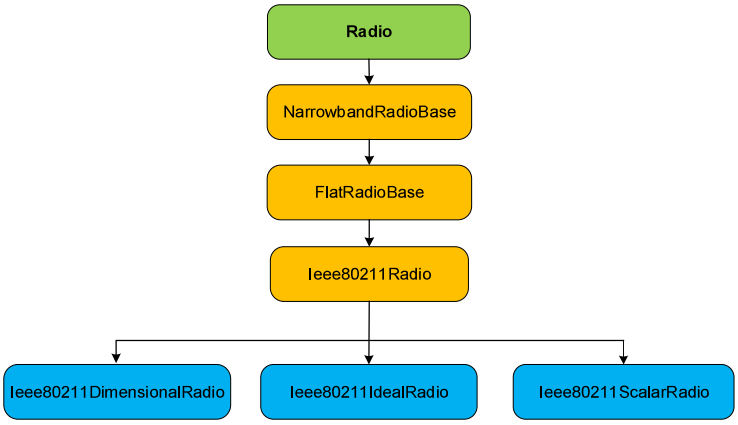

Figura 2.10. Diagrama de Herencia de la clase Radio

En la Figura 2.11 se muestran las subclases de RadioMedium. El modelo de radio trabaja siempre junto al modelo de medio físico que en este caso es Ieee80211RadioMedium, la misma que es una subclase de RadioMedium, la cual nos interesa estudiar ya que constituye un modelo base que representa todos los efectos del canal de transmisión y envía la RadioFrame a través del medio físico.

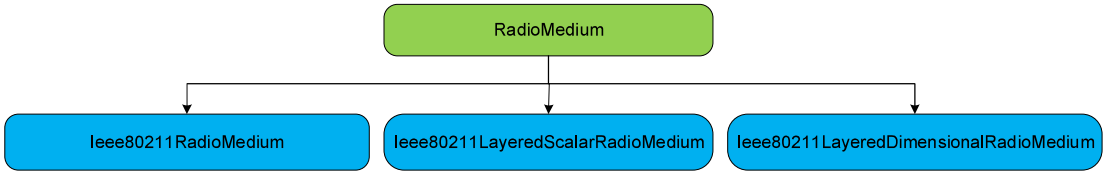

Figura 2.11. Diagrama de Herencia de la clase RadioMedium

2.2.13 MODELO DE RADIO

El modelo de radio definido con la clase Radio es el realiza el proceso de transmisión y recepción. Permite que el nodo ad hoc trabaje como receptor o transmisor, utiliza un conmutador para cambiar de un modo a otro. Al igual que todas sus subclases contiene el modelo de transmisor, receptor, antena y consumo de energía como se observa en la Figura 2.12.

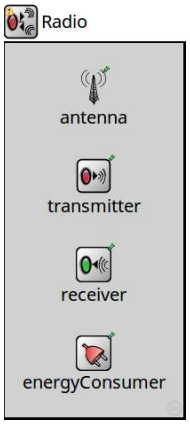

Figura 2.12. Estructura del módulo compuesto Radio.

2.2.13.1 Submódulo de transmisor

Describe todo el proceso en el que los paquetes se convierten en señales eléctricas. Contiene la interfaz ITransmitter la cual contiene los siguientes métodos:

- getMaxPower(): se otiene la máxima potencia de transmisión en watts.
- getMaxCommunicationRange(): obtiene el rango máximo de comunicación.
- getMaxInterferenceRange(): obtiene el máximo rango de interferencia
- createTransmission(): retorna transmisión en base a la trama MAC recibida. Esta transmisión se crea en base a ITransmission en base al cual se obtiene: un identificador de transmisión, el transmisor que realiza la transmisión, la trama PHY, la trama MAC, el tiempo de inicio de transmisión, el tiempo en el que

culmina la transmisión, la duración de la transmisión, la duración del preámbulo, la duración de la cabecera, la duración de datos, las coordenadas de inicio y fin, la orientación de la antena al inicio y fin de la transmisión.

2.2.13.2 Submódulo de receptor

Modela el proceso en el que la señal eléctrica se convierte en paquetes. Contiene la interfaz IReceiver que realiza los que se indica en cada uno de sus métodos:

- getMinInterferencePower(): obtiene el nivel de potencia mínimo de interferencia que el receptor ignora.
- getMinReceptionPower():obtiene el nivel de potencia mínimo que puede detectarse.
- createListening(): crea la señal listening que indica el estado del medio.
- computeListeningDecision(): retorna el resultado del proceso de listening donde ya se indica el estado del receptor.
- computeIsReceptionPossible(): indica si es posible o no recibir información.
- computeIsReceptionAttempted(): indica si la recepción fue intentada o no.
- computeIsReceptionSuccessful(): indica si la recepción fue exitosa o no.
- computeReceptionDecision(): especifica si la recepción es posible, fue intentada o si fue exitosa.
- computeReceptionResult(): retorna el resultado de la recepción.
- computeReceptionIndication(): retorna la indicación de recepción que constituye información de control enviada a la trama MAC.

2.2.13.3 Submódulo de antena

Describe como las señales eléctricas se convierten en señales de radio y viceversa. Como se indicó existen varios modelos de antenas que contienen la interfaz IAntenna con los métodos:

- getMobility(): obtiene la movilidad de la antena, posición y orientación.
- getNumAntennas(): obtiene el número de antenas.
- getMaxGain(): obtiene la máxima ganancia de la antena.
- computeGain(): calcula la ganancia en una dirección determinada.

2.2.13.4 Submódulo de consumo de energía

Contiene la interfaz IEnergyConsumer que contiene el método getPowerConsumption() que obtiene la enegía consumida.

2.2.13.5 Interfaz IRadio

Describe un dispositivo capaz de transmitir y recibir señales de radio simultáneamente. Define modos de radio (RadioMode), estados de recepción (ReceptionState) y estados de transmisión (TransmissionState); los cuales se describen en la Tabla 2.19.

Tabla 2.19. Parámetros de la interfaz IRadio *[50]*.

Modo	Estado	Descripción
RadioMode	RADIO_MODE_OFF	Indica que el radio está apagado, por lo que no es posible recibir ni transmitir tramas, el consumo de potencia es cero y la conmutación es lenta.
	RADIO_MODE_ SLEEP	Indica que el radio está en reposo, y tampoco es posible transmitir y recibir, el consumo de potencia es mínimo y la conmutación rápida.
	RADIO_MODE_ RECEIVER	Indica que el radio está listo para recibir información, no se puede transmitir, el consumo de potencia es medio en la recepción y bajo si el receptor está en estado libre.
	RADIO_MODE_ TRANSMITTER	El radio está listo para transmitir información, no es posible recibir. Así mismo cuando el transmisor está libre el consumo de energía es bajo. Mientras que cuando se transmite, el consumo de energía es alto.
	RADIO_MODE_ TRANSCEIVER	Indica que el radio puede recibir y transmitir simultáneamente. El consumo de potencia será medio cuando está libre, medio cuando recibe y alto cuando transmite.
	RADIO_MODE_ SWITCHING	Permita cambiar de un estado a otro y el consumo de energía es mínimo.
Reception State	RECEPTION_ STATE_UNDEFINED	Indica que no se conoce el estado del medio y no se puede detectar señales.
	RECEPTION_ STATE_IDLE	El medio se encuentra libre pero no detecta señales.
	RECEPTION_ STATE_BUSY	El medio físico está ocupado y se detecta una señal que no es suficientemente fuerte para recibir.
	RECEPTION_ STATE_RECEIVING	El medio está ocupado y la señal es suficientemente fuerte para ser detectada y recibida por el receptor.

Modo	Estado	Descripción
Transmission State	TRANSMISSION_ STATE_UNDEFINED	El estado de transmisión no se reconoce.
	TRANSMISSION_ STATE_IDLE	Indica que no se tiene ninguna señal en el medio.
	TRANSMISSION_ STATE_TRANSMITTING	Indica que el medio está ocupado y que se está realizando la transmisión de información.

Para realizar el proceso de transmisión o recepción en el módulo de radio se debe conocer en qué modo se encuentra y el estado del transmisor como el receptor. La transmisión o recepción continúa o finaliza dependiendo de la parte de la señal de radio que se transmite o recibe, definiéndose varias partes como indica la Tabla 2.20.

Tabla 2.20. Descripción de partes de una señal de Radio [50].

Parte de la señal	Descripción
SIGNAL_PART_NONE	No especifica ninguna parte de la señal
SIGNAL_PART_WHOLE	Especifica todas las partes de la señal
SIGNAL_PART_PREAMBLE	Preámbulo de la señal
SIGNAL_PART_HEADER	Cabecera de la señal
SIGNAL_PART_DATA	Datos

2.2.13.6 Métodos en el módulo de radio Radio

El módulo de Radio contiene varios métodos para realizar el proceso de transmisión/recepción, a continuación, se describe cada uno:

- getTransmissionInProgress(): verifica si se está transmitiendo.
- getReceptionInProgress(): verifica si se está recibiendo.
- getTransmittedSignalPart(): obtiene la señal transmitida.
- getReceivedSignalPart(): obtiene la señal en recibida.
- handleMessageWhenDown():identifica el mensaje enviado que se recibe de capas superiores.
- handleMessageWhenUp():identifica el mensaje que se envía a capas superiores.
- handleSelfMessage(): identifica si el mensaje es de temporización del receptor, transmisor o conmutación.
- startTransmission(): comienza la transmisión de la trama MAC recibida y crea la trama de radio. Inicializa la temporización en el transmisor.

- continueTransmission(): continua la transmisión si se está enviando la cabecera y preámbulo.
- endTransmission(): termina la transmisión al finalizar el envío de los datos y para la temporización.
- abortTransmission(): aborta la transmisión.
- createRadioFrame(): crea la trama de radio.
- startReception(): comienza la recepción e inicializa el temporizador.
- continueReception(): continua la recepción si se está recibiendo la cabecera y preámbulo.
- endReception(): finaliza la recepción al obtener todo el campo de datos.
- abortReception(): aborta la recepción.
- sendUp(): envía la trama MAC obtenida en el receptor hacia la capa MAC.
- isReceiverMode(): verifica si el radio está en modo recepción.
- isTransmitterMode():verifica si el radio está en modo transmisión.
- isListeningPossible(): verifica si se puede escuchar el canal.

2.2.13.7 Proceso de transmisión en el módulo de radio

En la Figura 2.13 se muestra el proceso de transmisión. Este proceso comienza cuando el módulo de radio recibe una trama de la capa MAC que tiene una solicitud de TransmisionRequest dentro de la información de control.

Cuando el módulo de radio se encuentra en estado transmisión emite una señal que indica el cambio de estado. Existe un temporizador que mide el proceso de transmisión, se inicia al comenzar la transmisión y cuando este expira el proceso termina y nuevamente el módulo de radio regresa al estado idle indicando este cambio con una señal. La capa MAC envía una trama Ieee80211Frame y en capa física se forma la trama PLCP con lo que se transmite el preámbulo, la cabecera y los datos formándose una RadioFrame.

Para que el transmisor trabaje con el estándar IEEE 802.11a se define el tipo Ieee80211LayeredOFDMTransmitter en transmitterType, módulo en el cual se realiza codificación, modulación. Para crear la transmisión se realizan todos los procesos mencionados y para cada de los cuales describiremos los métodos empleados.

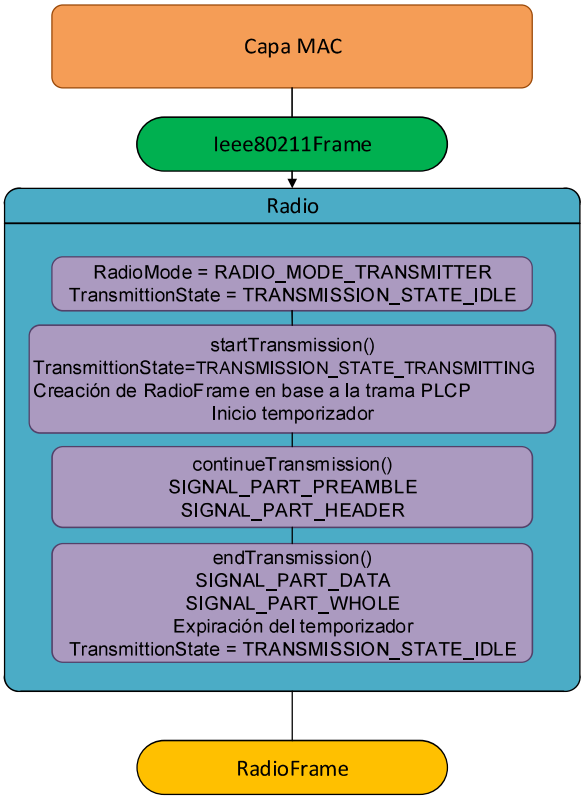

Figura 2.13. Proceso de Transmisión en INETMANET.

La estructura de Ieee80211LayeredOFDMTransmitter se muestra en la Figura 2.14.

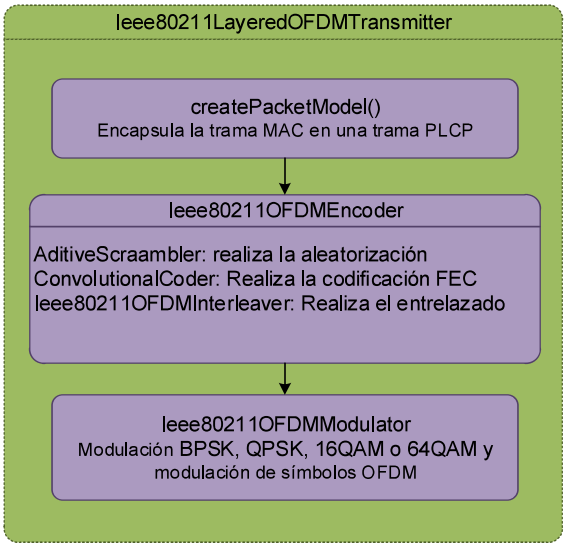

Figura 2.14. Proceso de Modulacion OFDM.

Ieee80211LayeredOFDMTransmitter puede trabajar en un modo denominado compliant que se define por una variable tipo booleana. En este modo no es posible configurar

parámetros como dataEncoder, signalEncoder, modulator, signalModulator, pulseShaper, digitalAnalogConverter, bandwidth, channelSpacing. Por esta razón en nuestro estudio se trabajará en modo compliant configurado en false para poder definir los parámetros de capa física según el estándar.

Los métodos que contiene Ieee80211LayeredOFDMTransmitter se describen a continuación:

- serialize(): convierte a binario y obtiene la longitud en bytes de la phyFrame.
- appendPadding(): obtiene el número de bits de relleno.
- createSignalFieldPacketMode():crea el campo de señal del modelo de paquete
- createDataFieldPacketModel(): crea el campo de datos del modelo de paquete.
- createPacketModel():define el tamaño de la cabecera PLCP en 48 bits y encapsula la trama MAC.
- createSymbolModel(): crea el símbolo OFDM.
- createBitModel(): obtiene los bits codificados.
- encodeAndModulate(): codifica y modula el campo de señal y de datos.
- createSampleModel(): no se encuentra implementado.
- createAnalogModel(): crea el modelo analógico, por defecto escalar createScalarAnalogModel().
- computeMode(): obtiene los modos para el tratamiento de la señal y de los datos, esto es modulación y codificación.

En el transmisor se crea la trama PLCP y una vez que la trama PLCP se forma se realizan varios procesos en el campo de señal y el campo de datos. En el campo señal se realiza codificación convolucional, aleatorización, mapeo en modulación inserción de piloto y modulación OFDM utilizando BPSK y R=1/2. En cambio, en el campo de datos se realiza primero entrelazado y posteriormente los demás procesos con la modulación y velocidad de codificación que sean especificadas.

El primer paso que se realiza en el transmisor constituye la codificación, Ieee80211OFDMEncoder realiza este proceso. Para esto contiene los métodos:

- encode(): método donde se realiza la aleatorización, posteriormente se utiliza FEC para la codificación convolucional y luego se realiza el entrelazado.
- getCode(): obtiene los bits codificados.

Para la aleatorización se puede utilizar AdditiveScrambler en el cual se especificará el generador polinomial x^7 + x^4 + 1. Para la codificación convolucional ConvolutionalCoder

la clase Ieee80211ConvolutionalCode define un generador polinomial 133, 171 con las velocidades 1/2, 2/3, 3/4 y 5/6. Para el entrelazado se utiliza Ieee80211OFDMInterleaver que define las dos permutaciones especificadas en el estándar.

El siguiente proceso en el transmisor constituye la modulación que se realiza con Ieee80211OFDMModulator, la misma que permite BPSK, QPSK, 16QAM, 64QAM. Cuyo método modulate() realiza modulación dividiendo la señal en grupos de bits que tienen una representación en números complejos de acuerdo a las tablas de modulación.

Luego se asocian en grupos de 48 y cada uno de estos se asocia a un símbolo OFDM en donde se insertan las subportadoras piloto.

Concluidos estos procesos la trama PLCP se constituye en una phyFrame que en el módulo de radio se nombra como RadioFrame, la misma que es enviada al medio inalámbrico.

2.2.13.8 Proceso de recepción en el módulo de radio

En la Figura 2.15 se presenta el diagrama de estados del proceso de recepción realizado en INETMANET. En este proceso constituye una parte importante la evaluación del estado de radio y la evaluación del estado de recepción.

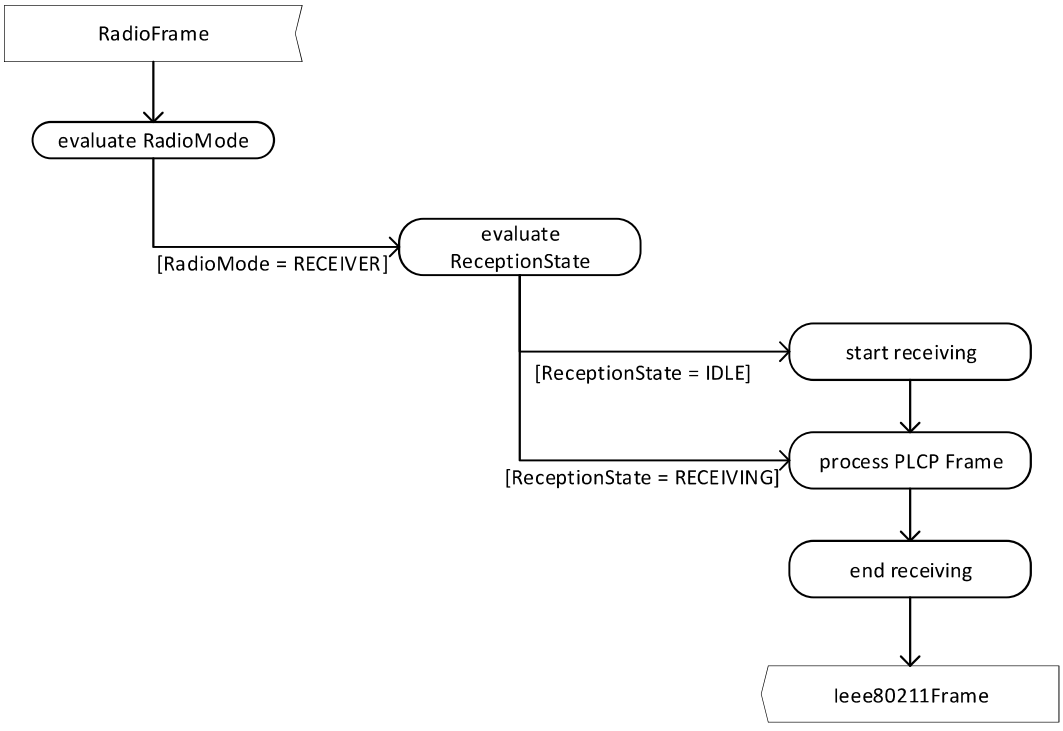

Figura 2.15. Diagrama de estados del proceso de recepción en INETMANET.

El proceso de recepción se muestra en la Figura 2.16; este inicia cuando el módulo de radio recibe una RadioFrame. Cuando la trama mencionada arriba al receptor cambia de estado a recepción. Así mismo existe un temporizador que empieza a correr el momento que comienza la recepción y expira al terminarla. Cuando la recepción es exitosa el módulo de radio añade ReceptionIndication como información de control. El radio cambia su estado a idle finalizado el proceso.

La trama de radio en el receptor en capa física es procesada para obtener la trama PLCP la cual es desencapsulada para obtención de la trama Ieee80211Frame que se envía a la capa MAC.

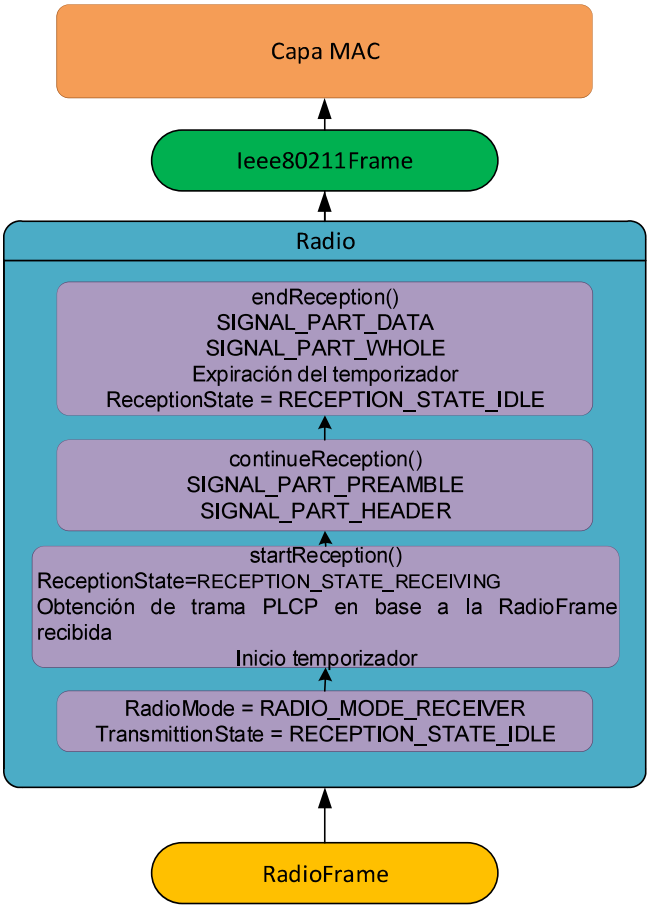

Figura 2.16. Proceso de Recepción en INETMANET.

Para que el receptor trabaje con IEEE 802.11a se define receptorType como Ieee80211LayeredOFDMReceiver (véase Figura 2.17) y para configurar el modulador, demodulador y aleatorizador se define isCompliant como false. Los métodos ejecutados en el receptor se describen a continuación:

- Los métodos computeListeningDecision(), computeIsReceptionPossible(), computeIsReceptionAttempted(), computeIsReceptionSuccessful(), computeReceptionDecision(), computeReceptionResult(), computeReceptionIndication() realizan los procesos indicados en la interfaz IReceiver.
- computeMode(): obtiene el modo de operación identificando el decodificador y demodulador definidos.
- getRate(): obtiene la velocidad de serializacion.
- getSignalFieldLength(): obtiene la longitud del campo de señal
- calculatePadding(): calcula los bits de relleno del campo de datos.
- getCodeRateFromDecoderModule(): obtiene la velocidad del decodificador.
- createSignalFieldSymbolModel(): obtiene el símbolo OFDM del campo de señal para demodularlo.
- createDataFieldSymbolModel():obtiene el símbolo OFDM del campo de datos para demodularlo.
- createCompleteSymbolModel():obtiene los símbolos OFDM.
- createSignalFieldBitModel(): demodula el campo de señal.
- createDataFieldBitModel(): demodula el campo de datos.
- createBitModel(): obtiene los bits de todos los campos.
- createSignalFieldPacketModel(): decodifica el campo de señal.
- createDataFieldPacketModel(): decodifica el campo de datos.

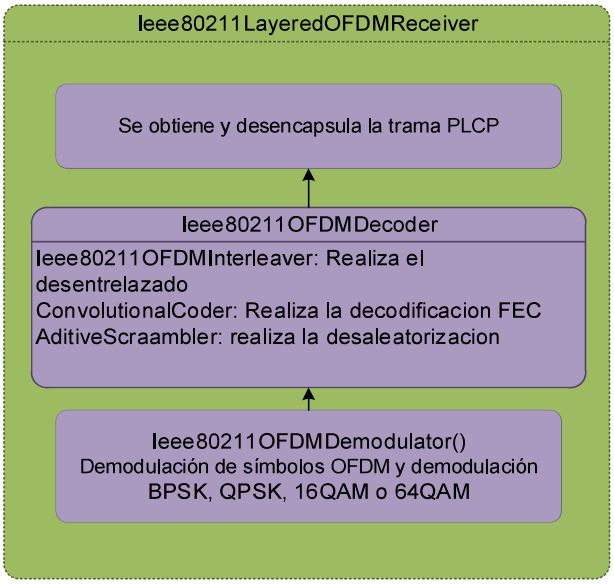

Figura 2.17. Estructura de IEEE80211LayeredOFDMReceiver.

El primer proceso realizado en el receptor es la demodulación del campo de señal y de datos, para lo cual se utiliza Ieee80211OFDMDemodulator. Aquí se realiza primero la demodulación OFDM de los símbolos y luego se ejecuta la demodulación BPSK, QPSK, 16QAM o 64QAM.

Ieee80211OFDMDecoder constituye el decodificador que realiza el desentrelazado, decodificación y desaleatorización de los campos de señal y de datos. El desentrelazado se realiza con Ieee80211OFDMInterleaver el cual utiliza las dos permutaciones definidas por el estándar. Luego con ConvolutionalCoder se realiza la decodificación FEC del campo de dato y del campo de señal. Finalmente, con AditiveScraambler se realiza la desaleatorizacion del campo de datos. En el campo de señal no se realiza este proceso. Finalmente, la señal de datos obtenida constituye la trama MAC la misma que es enviada a la capa superior.

2.2.14 INTERCAMBIO DE INFORMACIÓN EN LA CAPA FÍSICA IEEE 802.11a

El diagrama de bloques para el intercambio de mensajes se muestra en la Figura 2.18.

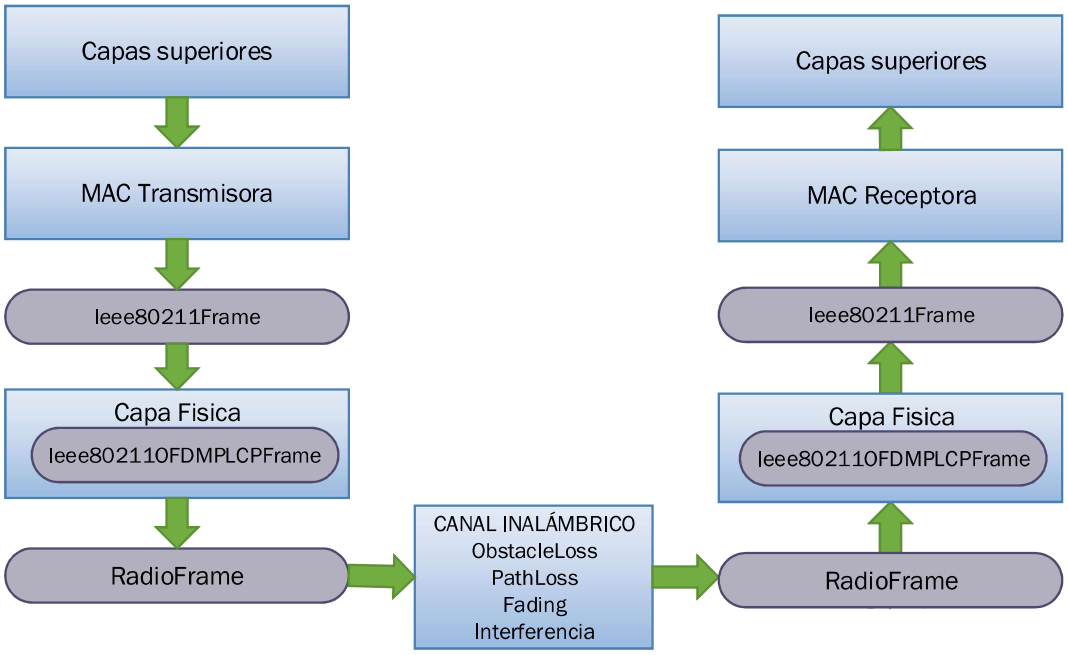

Figura 2.18. Intercambio de mensajes en INETMANET.

La capa MAC envía la trama MAC denominada Ieee80211Frame a la capa física y en capa física se forma la trama PLCP denominada Ieee80211OFDMPLCPFrame que para ser enviada al canal inalámbrico se transforma en una trama de radio denominada

RadioFrame. La trama de radio viaja por el canal inalámbrico y en el receptor la trama de radio se recibe y se obtiene la trama PLCP que luego es desencapsulada para obtener la trama MAC que será enviada a la capa MAC receptora.

2.2.15 PROCESOS EJECUTADOS EN EL MODELO DE MEDIO FÍSICO

La Figura 2.19 muestra la estructura del Modelo RadioMedium. El modelo de medio físico definido con la clase RadioMedium envía y recibe tramas de radio en el canal inalámbrico. Al igual que todas sus subclases contiene submodelos de propagación, pérdida de camino, modelo analógico, pérdida por obstáculos y ruido de fondo.

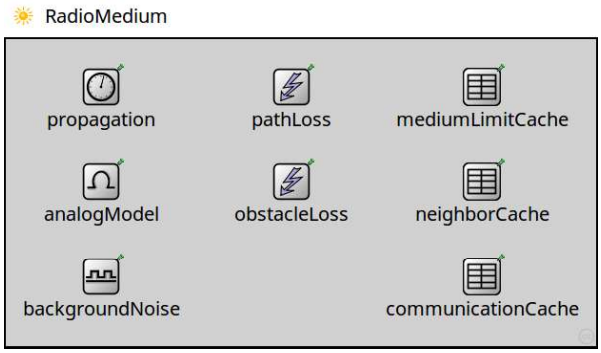

Figura 2.19. Modelo RadioMedium.

Primero indicaremos que realiza cada uno de los métodos de los submodulos y se describirán todos los procesos ejecutados por este modelo.

2.2.15.1 Submódulo de propagación

Como se indicó anteriormente se pueden definir dos tipos de módulos de propagación, los mismos que extienden PropagationBase en donde se define la propagationSpeed en mps. Contiene la interfaz IPropagation que modela la propagación de la señal en el medio físico a través del tiempo. Los métodos que contiene realizan lo siguiente:

- getPropagationSpeed(): retorna la velocidad de propagación de las señales de radio de 0 a infinito.
- computeArrival(): calcula las coordenadas en espacio y el tiempo de las transmisiones de un dispositivo en movimiento.

2.2.15.2 Submódulo de pérdida del camino

Como previamente se indicó hay varios modelos que es posible definir, cada uno de estos realiza el cálculo de la pérdida de potencia según sus especificaciones.

Este submodulo contiene la interfaz IPathLoss que describe la pérdida de potencia de la señal cuando la señal viaja a través del medio inalámbrico. Los métodos que contiene realizan lo siguiente:

- computePathLoss(): se tienen dos métodos para el cálculo de la pérdida del camino. Los dos retornan el factor de pérdida con una valor de 0 si hay pérdida y 1 si no hay pérdida. El primer método retorna el factor de perdida en función de las transmisiones y el segundo retorna este factor de pérdida en función de la velocidad de propagación, frecuencia y distancia.
- computeRange(): retorna el rango del factor de pérdida.

2.2.15.3 Submódulo de modelos analógicos

Según el tipo de modelo analógico definido se realiza el cálculo del nivel de potencia de recepción, ruido y SNIR. Este submódulo contiene la interfaz IAnalogModel que describe la atenuación de la señal mientras se propaga por el medio, incluyendo pérdida del camino, desvanecimiento, reflexión, refracción, absorción, etc. Contiene los métodos que se describen a continuación:

- computeReception(): calcula el nivel de potencia en recepción incluyendo toda la atenuación.
- computeNoise(): calcula el ruido total sumado a las señales interferentes.
- computeSNIR(): calcula el SNIR.

2.2.15.4 Submódulo de pérdida por obstáculos

Según el tipo de modelo de pérdida por obstáculos definido realiza el cálculo de la pérdida por obstáculos. El submódulo contiene la interfaz IObstacleLoss que describe la reducción del nivel de potencia que se debe a obstáculos. Un único método se encarga de realizar el cálculo de las pérdidas debidas a obstáculos computeObstacleLoss() en función de la frecuencia y posición de transmisión y recepción.

2.2.15.5 Submódulo de ruido de fondo

Este submódulo tiene la interfaz IRadioBackgroundNoise que genera ruido en el medio. Contiene el método computeNoise() que obtiene el nivel de potencia de este ruido.

2.2.15.6 Submódulo de caché límite del medio

Contiene la interfaz IMediumLimitCache que guarda temporalmente varios valores límites del medio.

2.2.15.7 Submódulo de caché de comunicacion

Contiene la interfaz ICommunicationCache que guarda de manera temporal varios resultados de cálculos relacionados a la comunicación a través del medio.

Submódulo de caché de vecinos

Formada por la interfaz INeighborCache que guarda las relaciones entre radios.

2.2.15.8 Métodos de RadioMedium

El modelo del medio físico contiene diferentes métodos que se describen a continuación:

- addTransmission(): añade nuevas transmisiones al medio físico.
- transmitPacket(): obtiene el paquete que se va a transmitir.
- createTransmitterRadioFrame(): obtiene la trama de radio en el transmisor.
- sendToAffectedRadios(): envía la trama de radio a todos los radios afectados.
- sendToAllRadios(): envía la trama de radio a todos los radios.
- sendToRadio(): envía la trama de radio al medio físico.
- isRadioMacAddress():Obtiene la dirección MAC del receptor según la tabla de direcciones MAC.
- isPotentialReceiver(): retorna el valor true si es posible realizar la recepción de la trama de radio enviada.
- isInCommunicationRange(): verifica si la señal está dentro del rango de comunicación.
- isInInterferenceRange(): verifica el rango de señales interferentes.
- isInterferingTransmission(): verifica si existen señales interferentes.
- removeNonInterferingTransmissions(): remueve los datos de transmisiones pasadas que no afectan a las nuevas.
- computeReception(): calcula el nivel de potencia de la señal recibida de acuerdo al modelo analógico definido.

- computeInterference(): calcula el nivel de potencia del ruido y señales interferentes de acuerdo al tipo de ruido de fondo definido.
- computeReceptionDecision(): con un contador indica cuantas veces se ha realizado este proceso y retorna valores de listening, reception, part, interference, snir.
- computeReceptionResult():con un contador indica cuantas veces se ha realizado este proceso y retorna valores de listening, reception, interference, snir.
- computeListeningDecision():con un contador indica cuantas veces se ha realizado este proceso y retorna listening, interference
- listenOnMedium(): escucha el medio, para verificar si es posible la comunicación.
- createReceiverRadioFrame(): crea la trama de radio en recepción.
- getArrival(): obtiene la señal que llega al medio.
- getListening(): obtiene la señal de listening.
- getReception(): obtiene el valor del nivel de la señal recibida.
- getInterference(): obtiene el nivel de potencia de señales interferentes.
- getNoise(): obtiene el nivel de ruido.
- getSNIR(): obtiene el SNIR
- receivePacket(): obtiene la trama recibida.
- isReceptionPossible(): verifica si es posible recibir tramas.
- isReceptionAttempted(): verifica si se intenta la recepción.
- isReceptionSuccessful(): verifica si la recepción fue exitosa.
- getReceptionDecision(): obtiene el valor de la decisión en recepción.
- getReceptionResult(): obtiene el valor recepción.
- receiveSignal(): recibe la señal en el receptor.

2.2.15.9 Descripción de procesos en el modelo de medio físico.

Una vez que el transmisor envía la señal al canal inalámbrico, el modelo del medio físico determina todos los receptores afectados en base al rango de comunicación del transmisor y el modo de operación, enviando la trama de radio a todos los receptores afectados.

Luego el modelo analógico utiliza la señal original transmitida y la señal de llegada para calcular el nivel de potencia de la señal del receptor, SNIR y ruido, para esto se aplica el modelo de pérdida de trayectoria y el modelo de pérdida de obstáculos. El modelo de ruido de fondo también calcula una señal de ruido la cual se agrega a la interferencia.

El modelo del receptor determina si la recepción es o no es posible dependiendo de la potencia de recepción y la sensibilidad.

2.3 MIXIM

2.3.1 DESCRIPCIÓN

MixiM es una extensión de OMNeT++ creada para simular todo tipo de comunicaciones inalámbricas con un conjunto de poderosas herramientas. Provee algunos modelos base que refinan la funcionalidad de OMNeT++ usando herencia; estos modelos contemplan las 3 dimensiones: tiempo, espacio y frecuencia. Adicionalmente, MiXiM es modular y extensible, permitiendo usar modelos existentes o facilitando la creación de nuevos modelos. Su estructura oculta la complejidad de las simulaciones ofreciendo al investigador una interfaz limpia, de modo que agregar nuevas funcionalidades no requiere conocer la arquitectura que está debajo de ella.

MiXiM divide al sistema de comunicaciones en capa MAC, capa física y canal inalámbrico. El comportamiento de las capas Física y MAC varían dependiendo de si se trata de un transmisor o un receptor [47].

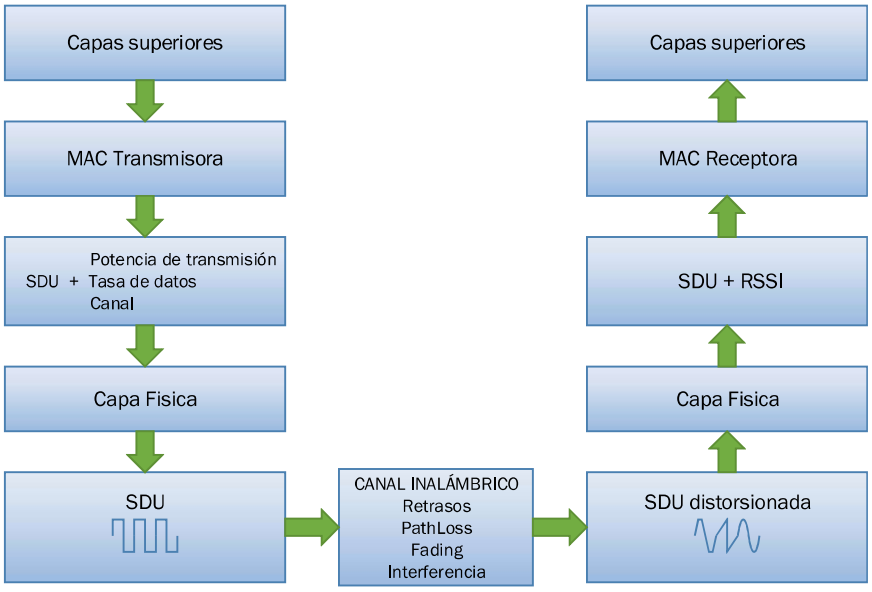

Figura 2.20. Sistema de comunicaciones en MiXim [47].

La Figura 2.20 muestra el sistema de comunicaciones tal como se modela en MiXim. La capa MAC del transmisor empaqueta los paquetes recibidos en una SDU (Service Data Unit). Luego agrega información de cómo transmitir el paquete y éste se transfiere a la

capa física, misma que genera una representación adecuada de la señal con esta información [52]. La señal se transmite a través del medio inalámbrico, donde se ve afectada por fenómenos como fading, shadowing, retrasos, pathloss e interferencia. La capa física del receptor recibe una señal distorsionada por el medio inalámbrico a la que intenta reconstruir para obtener la SDU original. Ésta SDU se pasa a la capa MAC con información adicional de la RSSI (Received Signal Strength Indicator).

2.3.2 UNA BREVE INTRODUCCIÓN A LA CAPA FÍSICA DE MiXiM

La capa física de MiXiM es responsable de enviar y recibir tramas, aplicar efecto al canal, detección de colisiones, cálculo del BER y de aplicar los AnalogueModels usados en la simulación [47].Cada una de estas funciones, son manejadas por distintas clases [46], como se puede ver en la Figura 2.21. Las 5 clases/funciones que componen a MiXiM están rodeadas por un un cuadro discontinuo: BasePhyLayer, Decider, AnalogueModel, Radio y ChannelInfo.

Las clases que se encuentran fuera del cuadro discontinuo (Signal, AirFrame, ConstMapping y Mapping), no forman parte de MiXiM, pero son necesarias para el funcionamiento de MiXim [47].

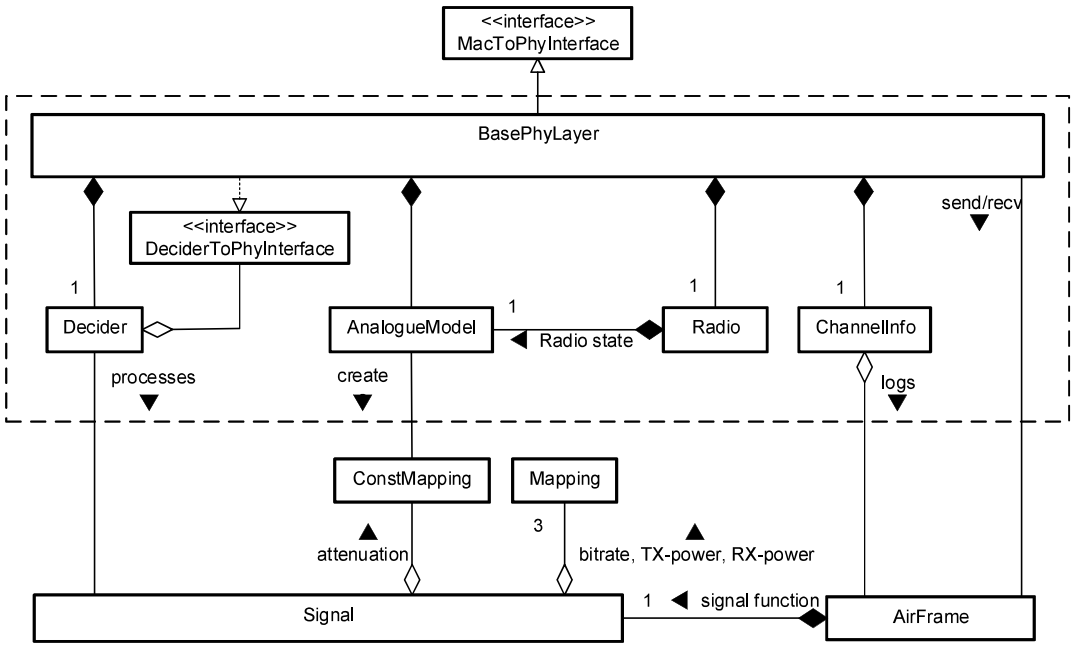

Figura 2.21. Gráfico de clase de la capa física de MiXiM (doxygen) [52].

Los AnalogueModels sirven para modelar las afectaciones del canal inalámbrico sobre una señal (Signal). Simulan fading, shadowing y pathloss de la señal (Signal) recibida [47].

Los objetos de la clase Signal, almacenan las cualidades físicas de la que se transmite por el canal inalámbrico. Las Signal se crean en la capa MAC, pero son afectadas en la capa física por los AnalogueModels, que modelan varias propiedades del canal inalámbrico.

La clase AirFrame, consituye el mensaje que se intercambia entre las entradas y salidas de las compuertas en el simulador [46] y que se envía entre capas físicas de dos nodos distintos. AirFrame es la clase contenedora de Signal. Las AirFrames contienen información como el instante de tiempo en el que la transmisión de la señal (Signal) inició.

BasePhyLayer constituye la clase mas importante, se encarga de conectar otras clases/funciones y de manejar mensajes de la capa MAC [46].Adicionalmente provee interfaces a las capas físicas y MAC de otros nodos [47].

Radio modela el tiempo que un transceiver pasa en un estado determinado (off, sleep, receiving, transmitting) y también modela el tiempo que toma la transición a cualquiera de los estados. A través de la clase Radio se puede especificar si el transceiver es Half o Full Duplex.

ChannelInfo administra el estado del canal, desde punto de vista de su nodo [46].Para hacerlo, hace un seguimiento de las AirFrame. ChanelInfo conoce todas las AirFrame que están siendo transmitidas y todas aquellas que están actualmente en el canal.

Decider es un elemento clave, su tarea es procesar cada AirFrame recibida [46]. Clasifica las AirFrame como señal o ruido (usando información de ChannelInfo) y calcula el BER de los mensajes.

2.3.3 EL PROCESO DE TRANSMISIÓN Y RECEPCIÓN EN MiXiM

El proceso de transmisión de MiXim se muestra en la Figura 2.22. Cuando llega una PDU desde capas superiores, la capa MAC 802.11 crea un objeto de Tipo Signal con información de control que está llena por defecto y la agrega a un paquete Mac80211Pkt, que contiene la información que se desea enviar. Adicionalmente, la capa MAC se encarga de encapsular y pasar el mensaje a la capa Física. La misma crea una AirFrame con la información que se desea transmitir y un objeto Signal [46].

Para transmitir la AirFrame, el nodo necesita conocer el destino, la clase ChannelAccess obtiene esta información de ConnectionManager. Esta clase porporciona una lista de todos los nodos conectados al nodo que pretende enviar la AirFrame.

Una vez que ChannelAccess ha identificado los nodos conectados, lleva la AirFrame a las compuertas de entrada de los otros nodos. Este comportamiento refleja la naturaleza de difusión del medio inalámbrico. Es responsabilidad de los nodos determinar si las señales tienen la potencia suficiente y si los datos si le corresponden [47].

Si se simula el retardo de propagación, ChannelAccess calcula el retardo de propagación basado en la distancia entre los nodos e invoca al método sendDelayed() de OMNeT++, que ocasiona el inicio de un temporizador para llevar la AirFrame cuando éste finalice.

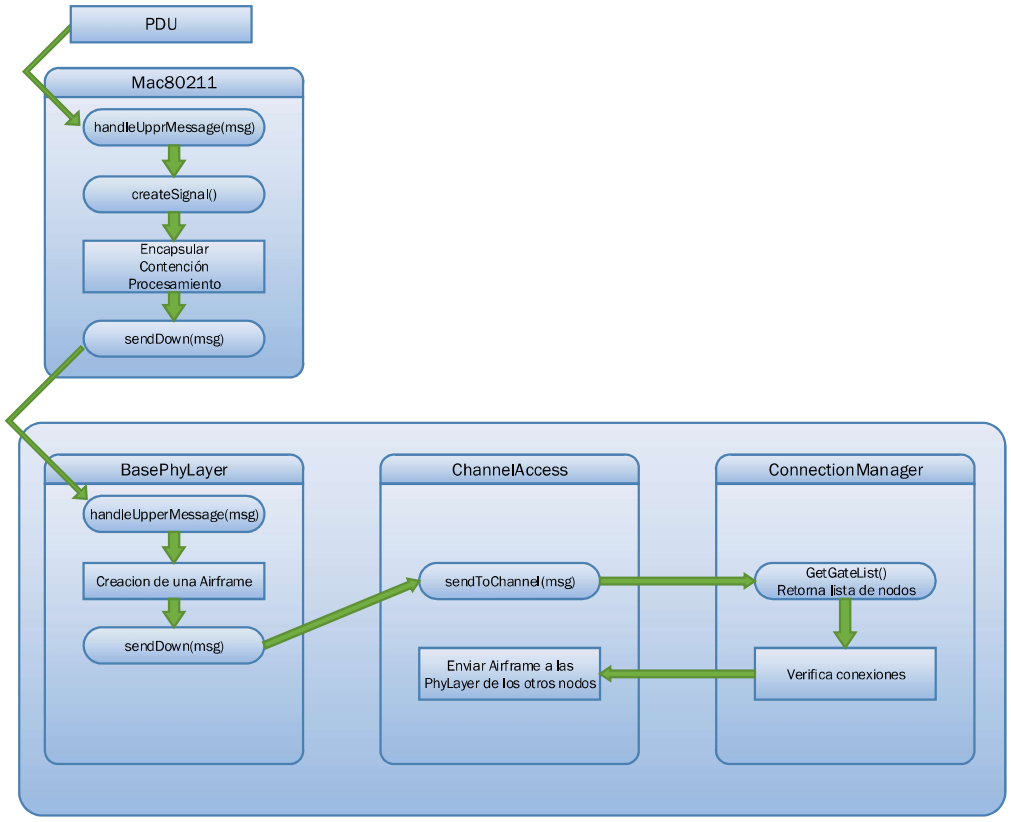

Figura 2.22. Proceso de Transmisión en MiXim.

El proceso de recepción de mensajes se muestra a través de un diagrama de estado que se muestra en la Figura 2.23. Cuando una AirFrame (mensaje) llega a la entrada de un nodo, es procesada por el método handleAirFrame() de la clase BasePhyLayer, donde se divide el proceso en 3 etapas que dependen en el estado de las AirFrame, la misma puede pasarse entonces a uno de varios métodos posibles.

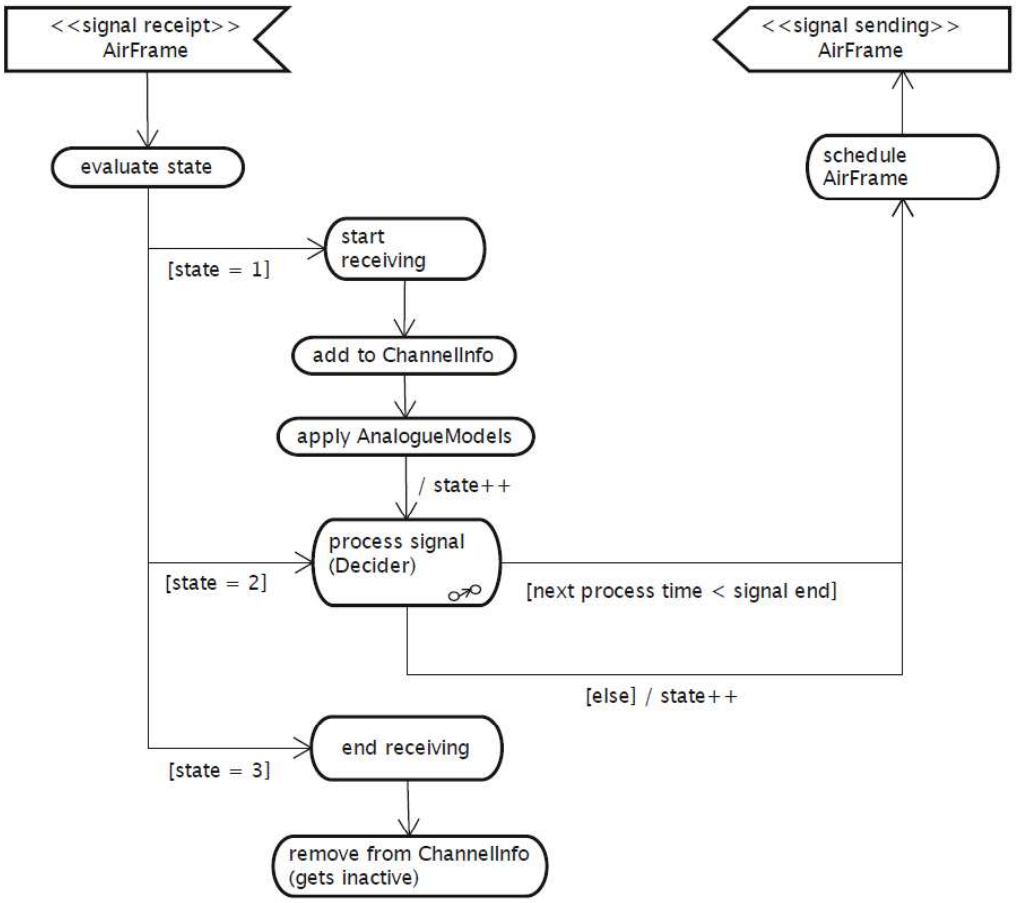

Figura 2.23. Diagrama de estados del proceso de recepción [52]

Para elegir un método, handleAirFrame() revisa el estado de la AirFrame [47]. Los estados posibles para estas tramas son: START_RECEIVE, RECEIVING Y END_RECEIVE identificados por números enteros (int). Si el mensaje se procesa por primera vez, el estado de esta será START_RECEIVE. Las etapas se muestran en la Figura 2.24:

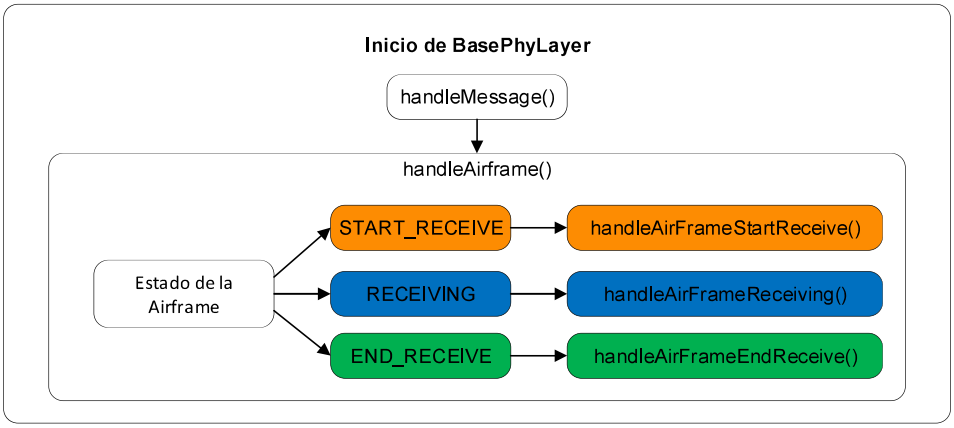

Figura 2.24. Método handleAirframe().

En la Figura 2.25 se muestra el método handleAirFrameStartReceive(). Cuando una AirFrame tiene en sus campos un estado igual a START_RECEIVE, ésta se transfiere al método handleAirFrameStartReceive() donde es agregada a ChannelInfo [46]. Luego se le aplican los filtros de atenuación al campo Signal con AnalogueModels, se aplica un retardo de propagación y se cambia el estado de la AirFrame a RECEIVING, luego la AirFrame se pasa de nuevo a BasePhyLayer que usa el método handleMessage() y encuentra que ahora el estado es RECEIVING, por lo que pasa la AirFrame al método handleAirFrameReceiving().

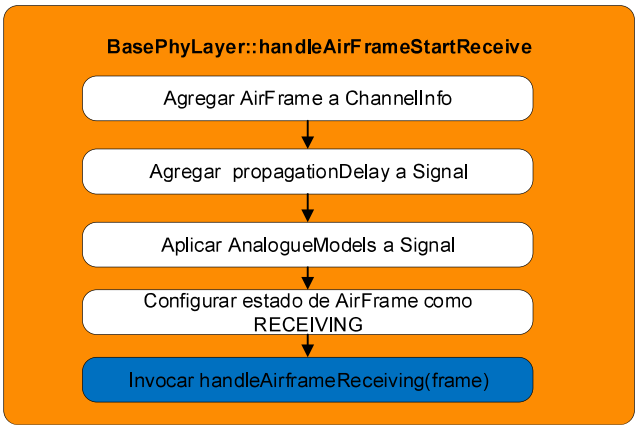

Figura 2.25. Método handleAirframeStartReceive().

El método handleAirframeReceiving y la ejecución del Decider se muestra en la Figura 2.26. EL primer procedimiento ejecutado es la invocación de el Decider. Este tiene que verificar si ha procesado esta trama antes o no. Si es la primera vez, emplea el método processNewSignal() y si no es así utiliza el método processSignalEnd(), existe también un método que no se usa: processSignalHeader() [47]. En primer lugar (cuando se invoca processNewSignal()), el Decider necesita comprobar que la señal tiene la potencia necesaria para ser recibida y que no hubo colisión. Para comprobar la potencia revisa el campo Signal de la AirFrame y para comprobar que no existen colisiones usa la información que provee ChannelInfo sobre la ocupación de canal.

Mas tarde, se invoca el método processSignalEnd(). Esta función determina si la AirFrame tiene errores o no. Para esto, calcula SNR en las AirFrames presentes en ChannelInfo (la información de ChannelInfo se solicita a través de BasePhyLayer.). Si la AirFrame ha sido recibida correctamemte, la AirFrame se pasa a la capa MAC, si la trama contiene errores, envía un mensaje de error a la capa MAC. Para finalizar la ejecución se cambia el estado de la AirFrame a END_RECEIVE y regresa a BasePhyLayer que invoca handleAirframeEndReceive().

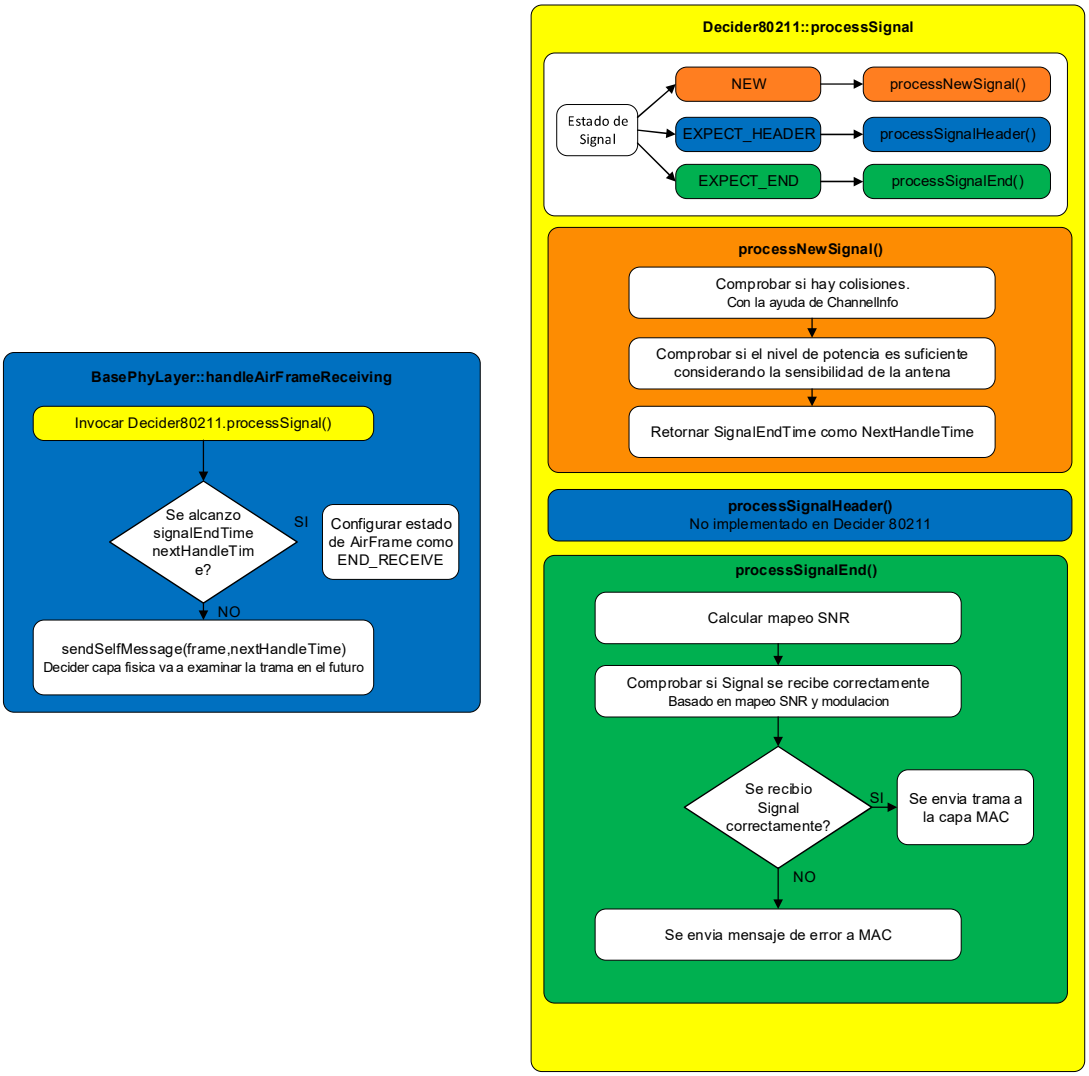

Figura 2.26. Método handleAirframeReceiving() y Decider80211.

En la Figura 2.27 se muestra el flujo de ejecución de handleAirFrameEndReceive. Este método Remueve las AirFrame de ChannelInfo y limpia Radio de los AnalogueModels.

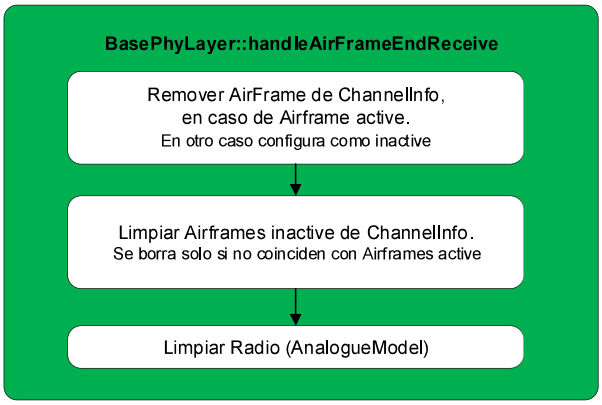

Figura 2.27. Método handleAirframeEndRecieve.

2.3.4 ANÁLISIS DE LOS COMPONENTES DE MiXiM

2.3.4.1 ChannelInfo

ChannelInfo da seguimiento a las AirFrame que se encuentran en el canal [47]. Desde el punto de vista de este bloque/clase, las AirFrame pueden tener 2 estados posibles:

Activo (active) e Inactivo (inactive). Cuando llega una AirFrame a la capa física, se agrega a ChannelInfo y se la configura como activa hasta que finalice la recepción de la misma. Cuando la recepción de una AirFrame ha finalizado (y por lo tanto haya sido configurada como inactiva), se almacena hasta que no existan otras AirFrame interfiriendo [46]. Otra de las razones por la que se deben almacenar las AirFrame es la posibilidad de calcular la SINR (signal to interference plus noise ratio) de una trama específica. ChannelInfo es empleada por el Decider para calcular el valor SINR.

Considere el ejemplo de la Figura 2.28:

- En t0, AirFrame 1 inicia y se configura como activa.
- En t1, AirFrame 2 inicia y se configura como activa.
- En t3, AirFrame 1 se configura como inactiva, pero no se elimina porque coincide con AirFrame 2
- En t4, AirFrame 3 se inicia y configura como activa.
- En t5, AirFrame 2 se configura como inactiva, pero no se elimina porque coincide con AirFrame 3. AirFrame 1 se elimina porque ya no está activa la AirFrame con la que coincidía.
- En t6, AirFrame 3 termina y es borrada porque no hay una AirFrame activa con la que coincida. AirFrame 2 es borrada porque la AirFrame 3 ya no está activa

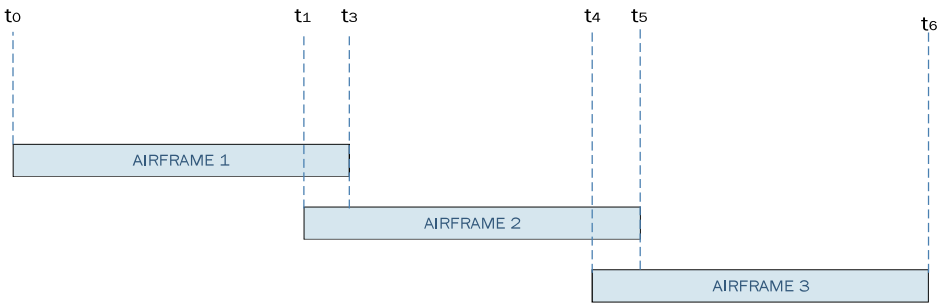

Figura 2.28. Ejemplo de Airframes que interfieren entre sí [47].

.

2.3.4.2 Signal

Todas las AirFrame tienen un campo Signal, que almacena la representación física de la AirFrame que viaja por el canal inalámbrico. El campo Signal a su vez contiene los siguientes parámetros:

- Inicio
- Duración
- Retardo de propagación
- Mapeos (mappings)

Los mappings representan la potencia de transmisión y recepción, tasa de datos y atenuaciones causadas por efectos del canal en la señal durante su transmisión.

El objeto Signal se crea en la capa MAC del remitente, el cual debe especificar:

- Potencia de transmisión
- Mapeo de tasa de datos

En la capa física se aplican a la AirFrame:

- Tiempo de inicio y duración
- Mappings de atenuación: son agregados por AnalogueModels
- Mapping de potencia de recepción es calculado bajo demanda multiplicando el Mapping de transmisión con cada mapping de atenuación de la señal.

La potencia de recepción es usada por el Decider para calcular SINR y realizar el análisis de errores en bits [46].

2.3.4.3 Radio

Cuando una Signal se transmite por el canal inalámbrico es atenuada por muchos efectos que se modelan con AnalogueModels [47]. Estos modelos no son los únicos factores que pueden afectar la transmisión (o recepción). El estado de la Radio también es relevante. Por ejemplo, una Radio (Half Dúplex), en estado transmitting no podría recibir datos de modo que todas las tramas que arriben al nodo serán descartadas durante la simulación. Radio contempla los siguientes estados:

- RX: Radio en estado de recepción.
- TX: Radio en estado de transmisión.
- SLEEP: Radio en estado de bajo consumo de energía.
- SWITCHING y NUM_RADIO_STATES: En realidad no es un estado (pero está listado como uno en el código de MiXiM), sirve para contar la cantidad de estados.

Es necesario porque el desarrollador podría hacer una subclase de Radio en la que se incluirían nuevos estados.

La Radio de un NIC es parte de la capa física y se implementa como una máquina de estados [46]. Radio tiene dos funcionalidades principales [47]:

- Simular los tiempos de conmutación entre estados
- Simular los efectos que una radio tiene en la recepción.

Tiempo de conmutación entre estados: Constituye el tiempo requerido para cambiar de un estado a otro. MiXiM contempla dos posibles escenarios: cuando el hardware es sofisticado, se puede considerar un tiempo de conmutación entre estados despreciable en otros casos podría ser deseable configurar un tiempo de conmutación entre estados.

En el caso de un tiempo de conmutación que no sea despreciable despreciable, la capa MAC es notificada con un mensaje de control RADIO_SWITCHING_OVER al terminar el proceso de conmutación. Para un tiempo de conmutación despreciable no se usan mensajes de control, y por lo tanto, no se degrada el rendimiento de la simulación.

Para cambiar el estado de una Radio se emplea el método setRadioState() en la interfaz MacToPhyInterface [47].

Efectos de una Radio en la recepción: Si el estado de una Radio es TX o SLEEP, no se puede recibir señales. Este comportamiento está representado por MiXiM haciendo uso de RadioStateAnalogueModel, que retorna mappings de atenuación salto a salto [46]. Si la Radio está en estado RX, la atenuación introducida por la misma es 0% en cualquier otro modo, la Radio introduce atenuación del 100%.

2.3.4.4 Analogue Models

La clase AnalogueModel constituye una interfaz para los modelos analógicos de la capa física. Un modelo analógico es un filtro que cambia el valor de atenuación de una señal para simular fenómenos como shadowing, pathloss u obstáculos.

En MiXiM es posible usar los modelos existentes o implementar unos modelos propios.

Los modelos disponibles (hasta la fecha de este documento) son:

- BreakPointPathlossModel
- IntensityModel
- JakesFading

- LogNormalShadowing
- PERModel
- RadioStateAnalogueModel
- RandomFreqTimeModel
- RandomFrequencyOnlyModel
- SimplePathlossModel
- UWBIRIEEE802154APathlossModel
- UWBIRStochasticPathlossModel

2.3.4.5 Decider

La tarea principal del Decider es evaluar las señales (Signal) recibidas. Esto significa categorizar señales (Signal) entrantes como interferencia o mensajes y calcular los errores del mensaje [52].

Decider también es responsable de evaluar el canal: es decir, realizar escucha de portadora (Carrier sensing), y reportar esa información a la capa MAC.

2.3.5 EVALUACIÓN DE LAS AIRFRAME

Todas las AirFrame se procesan varias veces, por el método processSignal() del Decider [52]. El número de veces que puede ser procesado depende de la implementación del Decider. Sin embargo, el proceso básico de evaluación de tramas se puede dividir en 3 etapas [47]:

1. Cuando se recibe una AirFrame por primera vez, decide en que instante de tiempo en el futuro, puede clasificar una AirFrame como ruido o como mensaje.
2. La segunda ocasión que el Decider recibe la AirFrame, tiene que decidir si la AirFrame es interferencia o un mensaje. Esto es posible analizando el mapping de SINR generado por ChannelInfo. Si se determina que la AirFrame es interferencia, el Decider emite una señal de control a la capa física para no recibir nuevamente esta AirFrame. Si la AirFrame es un mensaje, el Decider retorna el final de la trama como el siguiente tiempo en el que quiere recibir la AirFrame para su análisis.
3. En el paso final, el Decider tiene que analizar el atributo Signal de la AirFrame para buscar errores en los bits, para esto, crea un mapping SINR de toda la señal (Signal). Si la señal está bien, se envía la trama a la capa MAC. Si contiene errores envía un mensaje de error a la capa MAC.

2.3.5.1 Escucha del canal

El procedimiento de escucha de canal a nivel de capa física es importante para la capa MAC debido al uso de protocolos CSMA [52].

En la capa física de MiXiM es posible simular dos escenarios: el primero, cuando la escucha del canal toma tiempo, y el segundo, cuando el tiempo de procesamiento por la escucha es despreciable.

Para obtener el estado del canal sin tomar en cuenta el tiempo de escucha se emplea el método getChannelState() que se encuentra en MacToPhyInterface. GetChannelState() retorna un objeto de tipo ChannelState con la información del estado del canal [52].

Para considerar un tiempo para la escucha, la capa MAC envía un mensaje del tipo CHANNEL_SENSE_REQUEST a la capa física. Cuando la capa física recibe un mensaje de este tipo, el Decider escucha el canal por la cantidad de tiempo que haya sido especificada en el mensaje y contesta con un mensaje de control que contiene un objeto de tipo ChannelState con la información del estado del canal.

Los objetos Signal, están contenidos en las AirFrame; no contienen datos, almacenan las propiedades físicas (tasa de datos, potencia de transmisión, tiempo de inicio de transmisión de la trama transmitida) de las señales que se transmiten.

2.4 VEINS

2.4.1 INTRODUCCIÓN

Problemas críticos de tráfico como accidentes o congestión requieren del desarrollo de nuevos sistemas de transporte [47]. Si es posible modelar el tráfico en tiempo real dentro de una simulación, entonces se pueden proponer soluciones a los problemas como congestión de tráfico y búsqueda de rutas alternativas, reducción de luces rojas por ruta, control del flujo de vehículos para reducir emisiones de CO2, entre otros. Post-procesamiento es necesario para que todos los vehículos de una simulación de tráfico en tiempo real sean “inteligentes” de modo que establezcan comunicación entre ellos y puedan tomar una decisión. Las VANET tienen un gran desafío porque estos vehículos cambian la distancia entre ellos con mucha frecuencia, siendo necesario modificar los enlaces inalámbricos constantemente. Aquí aparecen los simuladores de red, que consideran a estos vehículos como nodos y hacen posible que se comuniquen entre ellos para tomar decisiones [52]. Uno de los simuladores de red mas célebres es OMNeT++, que puede simular VANETs gracias a un framework conocido como Veins.

2.4.2 DESCRIPCIÓN

Veins es un Framework diseñado para simular redes vehiculares (VANETs). Los modelos que se incluyen en Veins se ejecutan en OMNET++. Veins puede interactuar con generadores de movilidad para simular el tráfico en una región que sea de preferencia para el investigador.

MiXiM es un framework que modela principalmente fenómenos de capa física como atenuación, pathloss e interferencia. Veins hereda muchas clases de la API MiXiM, es decir, extiende algunas de sus funcionalidades con el propósito de simular lar redes vehiculares. Para ejecutar una simulación con Veins, se requiere de 4 elementos: OMNeT++, MiXim, Veins y SUMO.

La relación entre OMNeT++, MiXiM, Veins y SUMO se ilustra en la Figura 2.29.

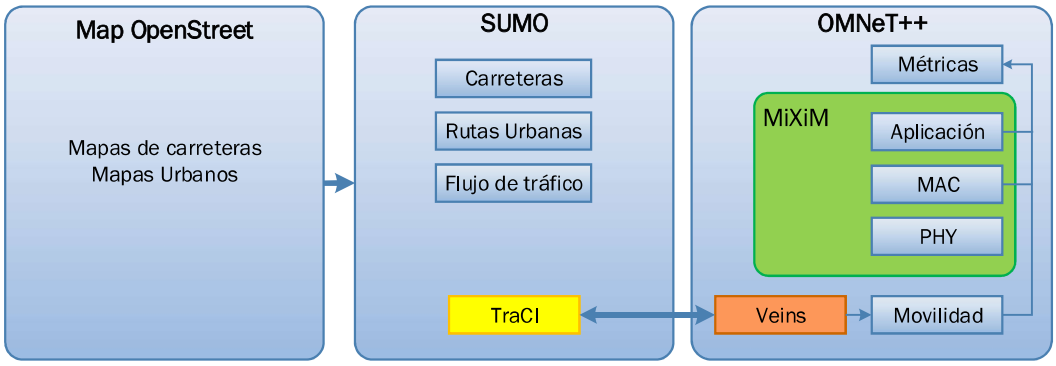

Figura 2.29. Relación entre OMNeT++, MiXiM, Veins y SUMO.

Durante la simulación, la información es procesada en cada nodo; como resultado de esta operación, se toma una decisión. Para empezar a modelar es necesario tener información de la ubicación geográfica como la ciudad o un área específica para que transiten los vehículos de la simulación; los mapas usados en las simulaciones se obtienen normalmente de OpenStreetMap [48], estos mapas se pasan a SUMO para modelar el tráfico y finalmente se pasan a Veins usando TraCI (*Traffic Control Interface*), un servidor que conecta las simulaciones en OMNeT++ con el modelamiento de flujo de tráfico de SUMO. OMNeT++ modela el sistema con nodos, pero estos se ven reflejados en SUMO como vehículos.

MiXiM y Veins forman parte de OMNeT++ y tienen una relación muy estrecha. Además de extender la capa física de MiXiM, Veins interactúa con su capa aplicación [48]. OMNeT++ maneja información conocida como métricas provenientes de Veins, la capas MAC y la

capa Aplicación de MiXiM. Las métricas son variables como el tiempo que está ocupado el canal o incluso la aceleración de los vehículos o las emisiones de CO2.

2.4.3 MENSAJES EN VEINS

2.4.3.1 Mac80211Pkt y AirFrame11p

Los mensajes mas importantes que se intercambian durante la ejecución de una simulación de Veins son: Mac80211Pkt y AirFrame11p. Las capas superiores envían sus datos a la capa MAC 802.11p que genera un paquete Mac802.11Pkt y se lo pasa a la capa física, ésta a su vez genera una AirFrame11p. Nótese las similitudes entre el paso de mensajes en Veins y el paso de mensajes de MiXiM (Figura 2.30).

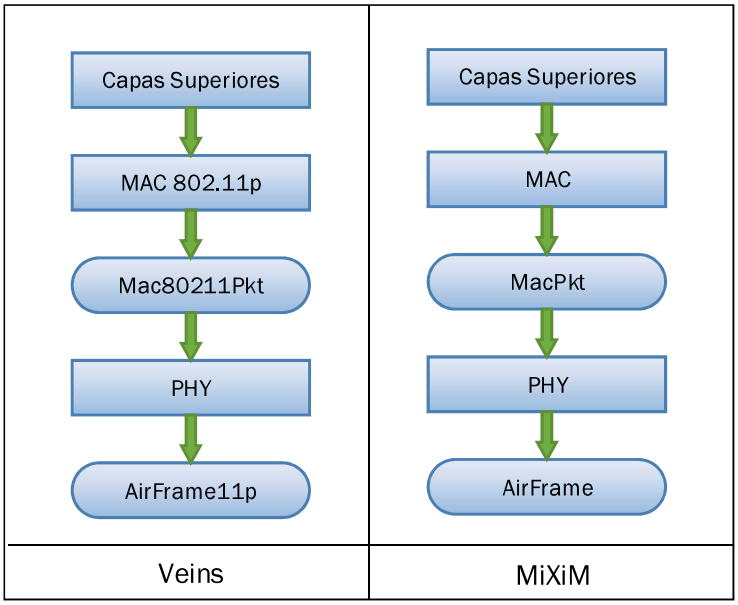

Figura 2.30. Comparacion del Paso de mensajes en Veins y MiXim.

Los paquetes Mac80211Pkt y AirFrame11p no solo son análogos a MacPkt y AirFrame; son extensiones de éstos. En Veins, Mac8011Pkt y AirFrame11p son clases que heredan los atributos de MacPkt y AirFrame.

En la Figura 2.31, se expone el formato de una trama 802.11.

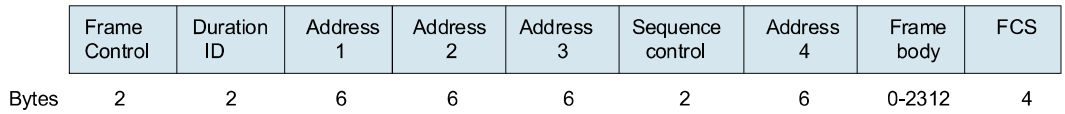

Figura 2.31. Trama IEEE 802.11.

En donde:

- Frame Control: Define el tipo y otra información.
- Duration: Tiempo esperado de ocupación del canal.
- Address 1: Dirección Origen.
- Address 2: Dirección Destino.
- Address 3: Dirección del transmisor.
- Address 4: Dirección del receptor.
- Sequence Control: Número de secuencia.

Los paquetes Mac80211Pkt conservan la misma estructura. Para cada campo, la clase Mac80211Pkt define un atributo distinto. Los atributos (campos) de la clase Mac80211Pkt se muestran en la Tabla 2.21. Algunos de los atributos han sido heredados de MacPkt, éstos están marcados en azul.

Tabla 2.21. Atributos de Mac80211Pkt.

Nombre	**Tipo**	**Descripción**
sequenceControl	int	Número de secuencia
Duration	simtime_t	La duración restante de la transmisión entre dos MACs. (RTS->CTS->DATA->ACK)
address4	int	Dirección del receptor
fragmentation	int	Parte del campo de control de la trama
destAddr	long	Direccion MAC de destino
informationDS	int	Parte del campo de control de la trama
srcAddr	long	Dirección MAC de origen
sequenceId	long	Número de sequencia para detectar mensajes duplicados
address3	int	Dirección del transmisor
Retry	bool	

En la Figura 2.32 se muestran los atributos (campos) de las AirFrame11p y AirFrame Tradicional. Las AirFrame11p extienden a las AirFrame de MiXiM con 2 atributos adicionales: underSensitivity y wasTransmitting.

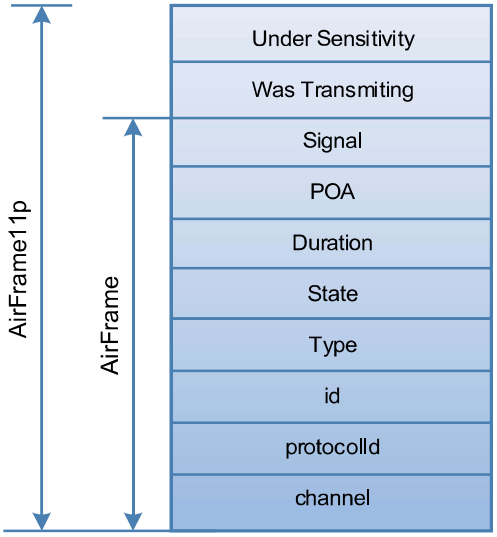

Figura 2.32. Atributos de las AirFrame y AirFrame11p.

Las Airframe11p difieren únicamente en dos atributos (campos), esto implica que los modelos de propagación (AnalogueModels) de MiXiM están disponibles en Veins.

2.4.3.2 ChannelState

Esta clase permite almacenar el estado actual del canal. Sus atributos se muestran en la Tabla 2.22.

Tabla 2.22. Atributos de ChannelState.

Nombre	Tipo	Descripción
idle	bool	Define si el canal está inactivo
rssi	double	Almacena el valor RSSI del canal

2.4.3.3 ChannelSenseRequest

Es un paquete de control dirigido desde la capa MAC a la capa física. Solicita al decider de la capa física que escuche durante un periodo de tiempo antes de enviar de regreso en conjunto la solicitud con ChannelState en este periodo. Este comportamiento es distinto al del método getChannelState() de MacToPhyInterface.

Cada ChannelSenseRequest que se envía a la capa física, tiene que definir un SenseMode y un timeout. El timeout define después de cuántos segundos la solicitud debe ser enviada de regreso a la capa MAC como máximo. El SenseMode indica a la capa física el propósito de sensar. Se tiene los siguientes SenseModes:

- UNTIL_IDLE : La capa física debe informar a la MAC tan pronto como el canal esté disponible, enviando la solicitud de regreso. En otras palabras, escucha el canal hasta que se encuentre libre o se alcance el timeout. Se representa con una variable int igual a 1.
- UNTIL_BUSY: La capa física debe informar a la MAC tan pronto como el canal esté ocupado, enviando la solicitud de regreso. En otras palabras, escucha el canal hasta que se encuentre ocupado o se alcance el timeout. Se representa con un valor entero igual a 2
- UNTIL_TIMEOUT: Escucha el canal hasta que se alcance el timeout. Se representa con un valor entero igual a 3.

Los atributos de ChannelSenseRequest se muestran en la Tabla 2.23.

Tabla 2.23. Atributos de ChannelSenseRequest.

Nombre	Tipo	Descripción
Result	ChannelState	El resultado de la solicitud conteniendo el estado actual del canal.
senseMode	int (SenseMode enum)	define el modo de sensing
senseTimeout	simtime_t	Tiempo para escuchar en el canal.

2.4.4 COMPONENTES PRINCIPALES DE VEINS

2.4.4.1 BaseDecider

BaseDecider es una clase que hereda de Decider (MiXiM), su tarea principal es transmitir una AirFrame desde processSignal a processNewSignal, processSignalHeader o processSignalEnd dependiendo del estado de la AirFrame. También provee respuestas a las solicitudes ChannelSenseRequest.

BaseDecider define 2 tipos de mensajes de control:

- PACKET_DROPPED: se emplea si la capa física ha detectado un error en el paquete. Se representa con el entero 22100
- LAST_BASE_DECIDER_CONTROL_KIND: es útil para las subclases, las cuales deben empezar sus propios tipos desde este valor. Se representa con el entero 22101

BaseDecider define 3 estados para el procesamiento de una señal:

- NEW: Se emplea si la señal se recibe por primera vez.
- EXPECT_HEADER: Es utilizado si se está esperando el header de la señal.

– EXPECT_END: Si se espera por el final de la señal.

BaseDecider almacena adicionalmente información de la sensibilidad con la que se debe recibir una AirFrame, el estado del actual del canal e información de la solicitud ChannelSenseRequest que está en marcha.

2.4.4.2 Decider80211p

La Figura 2.33 describe la clase Decider80211p y sus atributos principales.

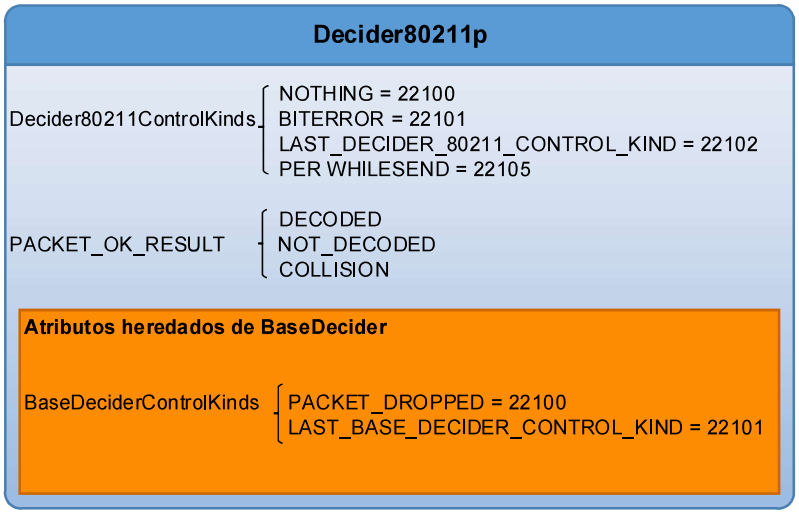

Figura 2.33. Atributos de Clase Decider80211p.

La clase Decider80211p hereda de BaseDecider. Define 4 tipos de mensajes de control (a través de un dato de tipo enum Decider80211ControlKinds):

– NOTHING=22100
– BITERROR=22101
– LAST_DECIDER_80211_CONTROL_KIND=22102
– RECWHILESEND=22103

Para conocer el resultado de un packet0K(), se emplean 3 tipos de resultados (representados con enum PACKET_OK_RESULT) :

– DECODED (correctamente decodificado)
– NOT_DECODED (descartado debido a una baja SNR)
– COLLISION (descartado debido a una colisión)

double snrThreshold es el valor de umbral para verificar un mapa SNR.

double ccaThreshold se usa para describir el nivel de potencia usado para declarar que el canal está ocupado si:

Se pierde una porción del preámbulo (802.11-2012 18.3.10.6 CCA requirements). Para determinar si el Preambulo se ha perdido se toma como referencia un umbral de CCA obligatorio de -65dBm para un canal de 10 MHz.

bool allowTxDuringRx activa o desactiva la interrupción de la recepción de mensajes.

Para un estándar 802.11 MAC empezar una transmisión mientras se recibe otra trama está prohibido.

Para propósito de investigación, se puede usar una capa MAC personalizada encima de 802.11p OFDM PHY y decidir interrumpir una recepción en marcha para iniciar una transmisión.

Si la variable allowTxDuringRx está en falso, la simulación terminará si esto ocurre.

double centerFrequency indica la frecuecia en la que el Decider escucha por señales.

bool collectCollisionStats se configure en True si se desea recolectar información de las colisiones. Se calcula el PER (Packet Error Rate) para valores SNR y SINR. Esto podría incrementar el tiempo de simulación. Por ello, este parámetro debe ser configurado como falso si es que las estadísticas de las colisiones no son necesarias. Int collisions almacena el número de colisiones.

2.4.4.3 PhyLayer80211p

En MiXiM, la clase BasePhyLayer es responsable de conectar todos los módulos [46]. PhyLayer80211p tiene un comportamiento similar, constituye una adaptación de la capa física de MiXiM para hacer posible modelar redes vehiculares [48].

PhyLayer80211p es una subclase de BasePhyLayer y AccessModuleWrap, es decir, hereda sus atributos, pero agrega funcionalidad [48]. En adición a los atributos heredados, está ProtocolIds de tipo enum que utiliza el entero 12123 para lidentificar al protocolo IEEE_80211.

La Figura 2.34 expone la estructura de la clase PhyLayer80211p. Dentro de los atributos heredados de BasePhyLayer tenemos ProtocolIds(otra vez), AirFrameStates, AnalogueModelList y ParameterMap. ProtocolIds, identifica a un protocolo genérico con un entero de valor cero. Se definió este tipo de protocolo, porque MiXiM fue pensado para ser

extendido acorde a las necesidades de los investigadores. Por eso PhyLayer80211p, identifica en su lugar al protocolo IEEE_80211. AirFrameStates, es de tipo enum y utiliza datos de tipo entero para representar los estados START_RECEIVE (1), RECEIVING (2) y END_RECEIVE (3). AnalogueModelList consituye una lista de todos los AnalogueModels que se van a aplicar a la señal. ParameterMap es una agrupación de parámetros que se pasan a los AnalogueModels. Estos parámetros varían de un modelo a otro.

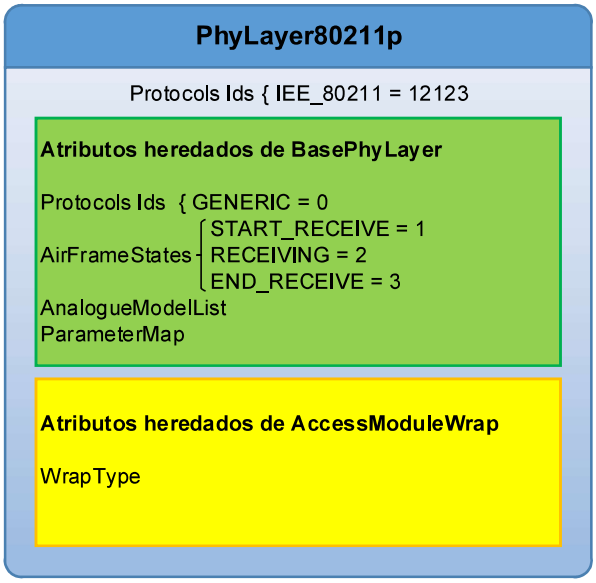

Figura 2.34. Atributos de Clase PhyLayer8011p.

El atributo WrapType es de tipo cModule y es usado a un nivel muy b ajo para asociar instancias de nodos con SUMO a través de TraCI. No debe ser manipulado por el usuario.

2.4.5 PROCESO DE TRANSMISIÓN Y RECEPCIÓN EN VEINS

2.4.5.1 Transmisión en Veins

El proceso de transmisión se muestra en Figura 2.35 .La capa Mac1609_4, requiere que se envíe desde capas superiores una PDU llamada WaveShortMessage, esta capa es incompatible con cualquier otro tipo de paquetes (si vienen de capas superiores). Cuando un WaveShortMessage llega a la Mac1609_4, se extrae información del canal y la prioridad del mensaje usando el método handleUpperMsg. Una vez que se finaliza este procedimiento, una cola es creada con el método createQueue. El último paso consiste en agregar los paquetes a la cola EDCA con el método queuePacket.

Para enviar los datos a la capa física hay que iterar sobre los paquetes de la cola hasta encontrar el que se desea enviar con el método initiateTransmit. El paquete encontrado se pasa a la capa física. Los paquetes que se envían desde la capa MAC se conocen como Mac80211Pkt.

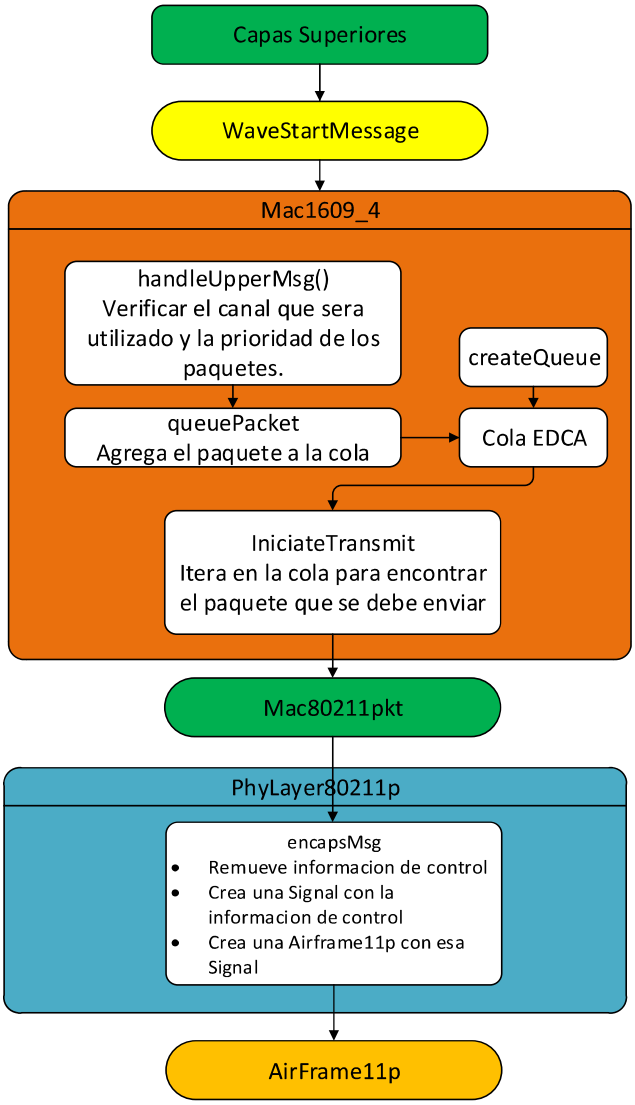

Figura 2.35. Proceso de Transmisión en Veins.

En la capa física (PhyLayer80211p) se recibe la trama MAC y se "encapsula" usando el método encapsMsg. En realidad, no se trata de un encapsulamiento real, solo se crea una AirFrame (que es un objeto) con información del paquete Mac80211Pkt a la que se agregan algunos atributos.

PhyLayer8011p depende de dos clases adicionales: ChannelAccess y ConnectionManager. El primero identifica al destino y calcula el retraso de propagación.

Para identificar un destino ChannelAccess hace uso de ConnectionManager para identificar los destinos. Para calcular el retraso de propagación primero se obtiene la ubicación de los nodos con el atributo de tipo Coord que poseen todas las AirFrame11p. Con las coordenadas de ambos nodos se calcula la distancia que hay entre ellos y se divide para la velocidad de la luz. Para todo esto, ChannelAccess utiliza el método sendToChannel().

A diferencia de MiXiM, Veins introduce un parámetro llamado máxima distancia de interferencia. En MiXiM se consideraba atenuación en espacio libre, en Veins el investigador puede introducir su propio modelo para modelar atenuación.

2.4.5.2 Recepción en Veins

La recepción de un mensaje es un proceso más complejo. Todo inicia con el método handleAirFrame() que Veins hereda de MiXiM. Las tramas que se manejan con este método tienen 3 estados posibles: START_RECEIVE, RECEIVING y END_RECEIVE que disparan los métodos handleAirFrameStartReceive(), handleAirFrameReceiving() y handleAirFrameEndReceive().

Las tramas reciben un tratamiento similar al de MiXiM, el método handleAirFrameStartReceive() de PhyLayer80211p agrega las tramas a ChannelInfo, luego agrega el retardo de propagación y los modelos de propagación y finalmente cambia el estado de la AirFrame11p a RECEIVING y luego se invoca al método handleAirFrameReceiving(), desde aquí, se hace una llamada al método processSignal() del Decider80211p; éste revisa el estado de la AirFrame11p (atributo Signal) que puede ser : NEW, EXPECT_HEADER o EXPECT_END. Estos estados ocasionan que sean invocados los métodos processNewSignal(), processSignalHeader() y processSignalEnd() respectivamente.

Si es la primera vez que se procesa la AirFrame11p, esta tendrá el estado de la señal (Signal) NEW, y por lo tanto se invoca el método processNewSignal() para ser tratada. Este método inicia con la extracción de la señal (Signal) de la AirFrame11p, de donde se obtiene la potencia para compararse con la sensibilidad de la antena de modo que se pueda decidir si la trama debe ser descartada o no. Si la potencia no supera la sensibilidad de la antena la trama es descartada y se revisa si el canal está ocupado, pero si la potencia excede el umbral definido por la sensibilidad de la antena, existen dos escenarios posibles. En el primero, la NIC no está sincronizada con la trama que se recibe, en este caso, se sincroniza e inicia la decodificación de la trama; en el segundo escenario, la trama debe ser

considerada como interferencia debido a que aún se está decodificando una trama anterior. Al final del proceso el estado de la Signal de AirFrame11p se cambia por EXPECT_END.

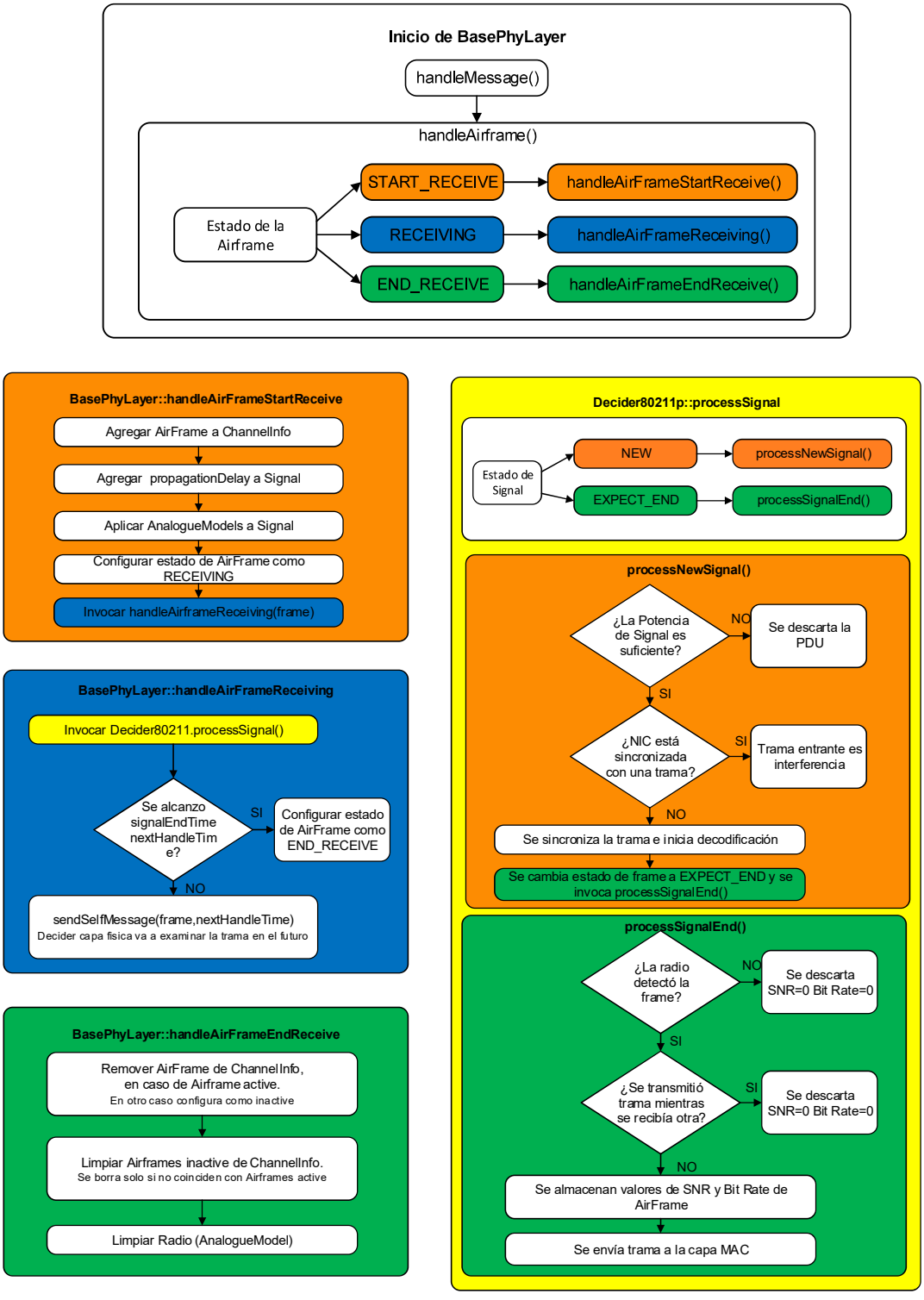

Figura 2.36. Proceso de recepción en Veins.

Cuando el Decider11p detecta el estado EXPECT_END en la Signal de la AirFrame11p, se invoca processSignalEnd(), aquí se revisa el parámetro underSensitivity para comprobar si la radio detectó la trama y se comprueba si la trama se recibió mientras se transmitía otra distinta. En ambos escenarios se considera un proceso de recepción fallida, se almacenan valores de SNR y Bitrate en cero en un tipo de dato para resultados llamado DeciderResult80211. El método analiza luego el mapping SNR para comprobar que la trama se recibió correctamente; si es así, se almacena un estado indicando que la recepción fue exitosa. Veins verifica adicionalmente que la trama que está siendo procesada es aquella con la que se ha sincronizado, de no ser así también se almacena un DeciderResult80211 con SNR y tasa de datos en 0.

2.5 SIMULACIONES

Para las simulaciones se realizan modificaciones a los ejemplos ya definidos en INETMANET y Veins, estableciendo configuraciones de acuerdo con los estándares IEEE 802.11a e IEEE 802.11p respectivamente.

2.5.1 SIMULACIONES EN INETMANET

En INETMANET se utiliza el ejemplo layered80211 ubicado en home/tesis/omnetpp-5.0/samples/inetmanet/examples/wireless/layered80211 y se lo modifica para trabajar con el estándar IEEE 802.11a y con n host.

2.5.1.1 Escenario de simulación

El escenario de simulación contiene un nodo estático y n host móviles que se comunican con el nodo. Los nodos móviles se mueven aleatoriamente con el modelo Random Way Point. Tanto el nodo estático, como los móviles se configuran con el estándar IEEE 802.11a y todos los procesos definidos en capa física en la transmisión y recepción.

Los nodos envían tráfico ICMP que se encapsula hasta formar la trama MAC la cual se envía a capa física formando la trama PLCP.

El entorno de simulación contiene los módulos mostrados en la Figura 2.37:

- inet.environment.common.PhysicalEnvironment
- inet.networklayer.configurator.ipv4.IPv4NetworkConfigurator
- inet.node.inet.AdhocHost
- inet.physicallayer.ieee80211.bitlevel.Ieee80211LayeredScalarRadioMedium

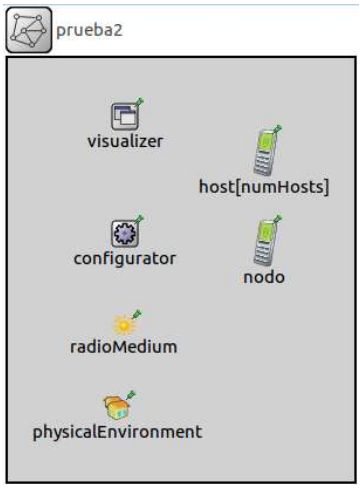

Figura 2.37. Entorno de simulación de INETMANET.

Y cada uno de estos módulos constituyen un submodulo de la red definido como:

- configurator: IPv4NetworkConfigurator
- radioMedium: Ieee80211LayeredScalarRadioMedium
- physicalEnvironment: PhysicalEnvironment
- host[numHosts]: AdhocHost
- nodo: AdhocHost

El submodulo configurator permite realizar la configuración del rango de direccionamiento del protocolo IPv4 para lo que se utiliza la siguiente línea

```
1       config = xml("<config><interface hosts='*' address='10.0.0.x'
        netmask='255.255.255.0'/></config>");
```

Código 2.1. Configuración del submodulo configurator.

El submodulo radioMedium permite configurar el medio físico y por medio del cual se podra realizar la configuración de cada uno de los modelos de propagación como se indica en las subsecciones del archivo omnet.ini:

```
1               [General]
2               # Free Space
3               [Config ModelTwoRayGroundReflection]
4               description = "TwoRayGroundReflection"
5               *.physicalEnvironment.groundType = "FlatGround"
6               *.radioMedium.pathLossType = "TwoRayGroundReflection"
7
8               [Config ModelRicianFading]
9               description = "RicianFading"
```

```
*.radioMedium.pathLossType = "RicianFading"

[Config ModelRayleighFading]
description = "RayleighFading"
*.radioMedium.pathLossType = "RayleighFading"

[Config ModelLogNormalShadowing]
description = "LogNormalShadowing"
*.radioMedium.pathLossType = "LogNormalShadowing"

[Config ModelNakagamiFading]
description = "NakagamiFading"
*.radioMedium.pathLossType = "NakagamiFading"
```

Código 2.2. Configuración Radio Medium.

El submodulo physicalEnvironment permite definir un ambiente físico para el modelo TwoRayGroundReflection.

El submodulo host[numHosts] permite definir un determinado número de nodos que enviaran información hacia nodo. Los dos submodulos se configuran para que trabaje con el estándar 802.11a utilizando los parámetros de la Tabla 2.24.

Tabla 2.24. Parámetros definidos por el estándar 802.11a.

Parámetro	**Configuración**
Modo de operación	**.wlan[*].opMode = "a"
Velocidad	**.wlan[*].bitrate = 36Mbps
Espaciamiento de canal	**.wlan[*].radio.receiver.channelSpacing = 20MHz **.wlan[*].radio.transmitter.channelSpacing = 20MHz
Frecuencia de portadora	**.wlan[*].radio.carrierFrequency = 5.765GHz
Potencia de transmisión	**.wlan[*].radio.transmitter.power = 250mW
Sensibilidad	**.wlan[*].radio.receiver.sensitivity = -70dBm

2.5.1.2 Configuración de la simulación en INETMANET

INETMANET contiene una capa física del estándar IEEE 802.11 para OFDM detallado. En el transmisor se realizan procesos de aleatorización, codificación, modulación, entrelazado y formación del símbolo OFDM y en el receptor se realiza el proceso inverso. Estas características se pueden definir en los escenarios de simulación planteados.

El tráfico que se envía en estas simulaciones corresponde al generado por el protocolo ICPM en capa de red, el cual es enviado por todos los hosts en secuencia ascendente.

Se define un nodo estático y varios nodos móviles que se mueven en base al modelo Random Way Point aleatoriamente, por lo que es necesario la ejecución de varias pruebas para realizar un promedio.

Los parámetros definidos para la simulación se muestran en la Tabla 2.25.

Tabla 2.25. Parámetros de simulación.

Parámetro	**Configuración**
Área de simulación	**.constraintAreaMinX = 0m **.constraintAreaMinY = 0m **.constraintAreaMinZ = 0m **.constraintAreaMaxX = 300m **.constraintAreaMaxY = 300m **.constraintAreaMaxZ = 0m
ARP	**.arpType = "GlobalARP"
Numero de hosts móviles	*.numHosts = 7 # *variable 3,4,5,6,7*
Movilidad en hosts	**.host*.mobilityType = "RandomWPMobility" **.host*.mobility.initFromDisplayString = **false** **.host*.mobility.initialX = uniform(0m,300m) **.host*.mobility.initialY = uniform(0m,300m) **.host*.mobility.initialZ = 0m **.host*.mobility.speed = 12mps # *variable 4,8,12,16,20*
Nodo fijo	**.nodo.mobilityType = "StationaryMobility" **.nodo.mobility.initFromDisplayString = **false** **.nodo.mobility.initialX = 150m **.nodo.mobility.initialY = 150m **.nodo.mobility.initialZ = 0m
Parámetros básicos del estándar IEEE 802.11a	**.opMode = "a" # *IEEE 802.11a* **.bitrate = 36Mbps # *Velocidad* **.carrierFrequency = 5.765GHz # *Frecuencia*
Capa MAC	**.wlan[*].typename = "Ieee80211Nic" **.wlan[*].macType = "Ieee80211CompatibleMac"
Radio	**.wlan[*].radioType = "APSKScalarRadio" **.wlan[*].radio.transmitter.power = 200mW **.wlan[*].radio.receiver.energyDetection = -62dBm **.wlan[*].radio.receiver.channelSpacing = 20MHz **.wlan[*].radio.bandwidth = 20MHz **.wlan[*].radio.receiver.sensitivity = -65dBm **.wlan[*].radio.receiver.snirThreshold = 4dB
ICMP	*.nodo.numPingApps = 0 *.host[*].numPingApps = 1 *.host[*].pingApp[*].destAddr = "nodo"

La configuración detallada del transmisor y del receptor se muestran en el ANEXO E.

2.5.2 SIMULACIONES EN VEINS

Las simulaciones se llevan a cabo usando el modelo cliente-servidor local. TraCI es un middleware que conecta SUMO y Veins (que corre dentro de OMNeT++) internamente a través del puerto 9999. OMNeT++ provee un script de Python denominado sumo-launchd.py, que provee la conexión cliente servidor con SUMO. Cuando una simulación inicia, OMNeT++ envía una solicitud al servidor TRaCI que contesta con la ubicación de los nodos en tiempo real. SUMO modela la movilidad de los vehículos y OMNeT++ modela el intercambio de mensajes entre los vehículos (nodos) usando Veins.

2.5.2.1 Parámetros de las simulaciones con Veins

En las tablas que se muestran a continuación, se exponen varios parámetros necesarios para efectuar simulaciones en OMNeT++. Estos parámetros deben ser especificados en el archivo omnetpp.ini. La Tabla 2.26 muestra parámetros específicos del escenario como su tamaño o el tiempo de simulación.

Tabla 2.26. Parámetros específicos de escenarios.

Parámetros de simulación	Valores Posibles (Ejemplos)
debug-on-errors	true
print-undisposed	false
sim-time-limit	6000s
**.scalar-recording	true
**.vector-recording	true
**.debug	false
**.coreDebug	false
*.playgroundSizeX	2500m
*.playgroundSizeY	2500m
*.playgroundSizeZ	50m

La Tabla 2.27 muestra los parámetros de configuración de WaveAppLayer usados para la comunicación entre nodos (o vehículos), applType representa el protocolo usado para el intercambio de información.

Tabla 2.27. Parámetros de configuración para configuración entre nodos.

Parámetros de WaveAppLayer	Valores Posibles (Ejemplos)
*.node[*].applType	“TraCIDemo11p”
*.node[*].appl.debug	false
*.node[*].appl.headerLength	256 bit
*.node[*].appl.sendBeacons	false
*.node[*].appl.dataOnSch	false
*.node[*].appl.sendData	true

Parámetros de WaveAppLayer	Valores Posibles (Ejemplos)
*.node[*].appl.beaconInterval	1s
*.node[*].appl.beaconPriority	3
*.node[*].appl.dataPriority	2
*.node[*].appl.maxOffset	0.005s

La Tabla 2.28 muestra los parámetros específicos de 802.11p para NICs (Network Interface Cards) como son: potencia de transmisión, tasa de datos y frecuencia de portadora.

Tabla 2.28. Parámetros específicos de las NICs.

Parámetros 802.11p	Valores posibles (Ejemplos)
*.connectionManager.pMax	20mW
*.connectionManager.sat	-89dBm
*.connectionManager.alpha	2.0
*.connectionManager.carrierFrequency	5.890e9 Hz
*.connectionManager.sendDirect	true
*.**.nic.mac1609_4.useServiceChannel	false
*.**.nic.mac1609_4.txPower	20mW
*.**.nic.mac1609_4.bitrate	18Mbps
*.**.nic.phy80211p.sensitivity	-89dBm
*.**.nic.phy80211p.useThermalNoise	true
*.**.nic.phy80211p.thermalNoise	-110dBm
*.**.nic.phy80211p.decider	xmldoc(“config.xml”)
*.**.nic.phy80211p.analogueModels	xmldoc(“config.xml”)
*.**.nic.phy80211p.usePropagationDelay	true

La Tabla 2.29 muestra parámetros específicos de TraCI como el servidor (normalmente localhost) y el Puerto.

Tabla 2.29. Parámetros específicos de TraCI.

Parámetros TraCIScenarioManager	Valores posibles (Ejemplos)
*.manager.updateInterval	0.1s
*.manager.host	“localhost”
*.manager.port	9999
*.manager.moduleType	“org.car2x.veins.nodes.Car”
*.manager.moduleName	“node”
*.manager.margin	25
*.manager.launchConfig	xmldoc(“erlangen.launchd.xml”)

La Tabla 2.30 muestra parámetros de movilidad, desde aquí se pueden configurar el número de accidentes que le pueden suceder a cada nodo, el tiempo de cada accidente y el tiempo de resolución del accidente.

Tabla 2.30. Parámetros de Movilidad de Veins.

Parámetros de movilidad	Valores posibles (Ejemplos)
*.node[*].veinsmobilityType	"org.car2x.veins.modules.mobility. traci.TraCIMobility"
" *.node[*].mobilityType	"TraCIMobility"
*.node[*].mobilityType.debug	true
" *.node[*].veinsmobilityType.debug	true
*.node[*0].veinsmobility.accidentCount	1s
*.node[*0].veinsmobility.accidentStart	75s
*.node[*0].veinsmobility.accidentDuration	30s

Nuestro estudio se enfocará en la capa física de Veins, o mas bien, en como Veins extiende la capa física de MiXiM para simulaciones de VANETs (802.11p).

Para realizar simulaciones con Veins, se requiere la ejecución de dos componentes esenciales: un ejemplo de Veins en OMNeT++ y SUMO.

Se realizaron 1200 simulaciones para obtener resultados veraces.

Ejecutar simultáneamente OMNeT++ y SUMO, y realizar centenas de simulaciones una y otra vez es prácticamente imposible. Por este motivo, se desarrollaron scripts para automatizar 3 actividades: simulaciones, procesamiento de datos, y extracción de datos.

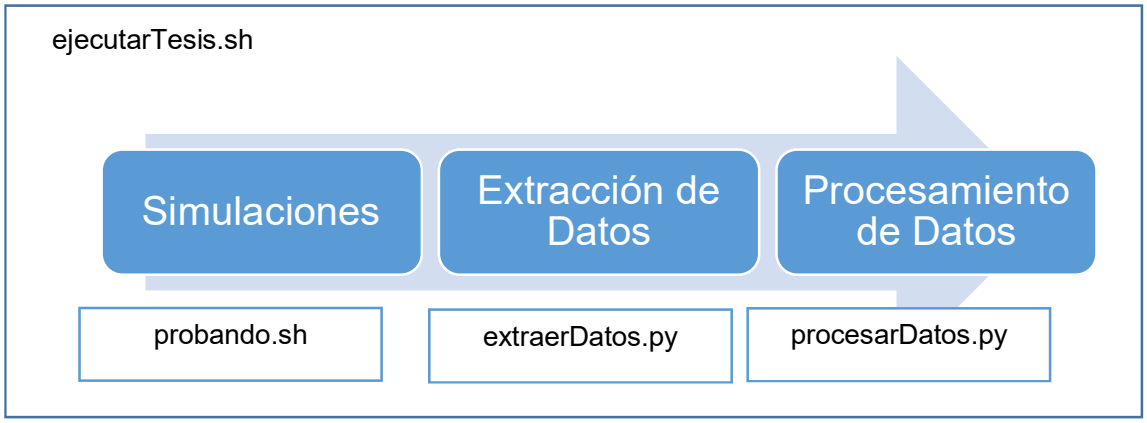

Figura 2.38. Estructura de script ejecutarTesis.sh.

Para ejecutar simulaciones, extraer sus datos y procesar sus resultados se utilizan 3 scripts respectivamente: probando.sh, extraerDatos.py y procesarDatos.py, los cuales se muestran en la Figura 2.38.

El script probando.sh itera sobre el número de vehículos y las velocidades deseadas. Estos valores se insertan en archivos de configuración de SUMO, luego procede a crear trips con duarouter (parte de SUMO) y finaliza levantando OMNeT++ y enlazando con SUMO para

ejecutar la simulación en consola. Durante este proceso no se puede visualizar la interfaz gráfica de OMNeT++, solo son evidentes los eventos que se solían mostrar en la consola de OMNeT++. Para ejecutar una simulación desde Bash en Linux, es necesario cambiar de directorio a la ubicación del archivo de simulación y luego ejecutar el comando de ejecución de simulaciones. Este procedimiento se muestra en el Código 2.3. EL código completo del scipt empleado para automatizar las simulaciones se encuentra en el ANEXO F.

```
1  #Cambia de directorio y Corre la simulacion en Omnet
2  cd /home/tesis/omnetpp-5.0/samples/veins/examples/veins
   opp_run -r 0 -u Cmdenv -n .:../../src/veins --image-path=../../images
3  -l ../../src/veins --debug-on-errors=false omnetpp.ini
```

Código 2.3. Código para ejecución de aplicaciones vía Consola de Bash.

Para extraer datos, fue desarrollado extraerDatos.py, un script de Python que convierte los archivos .vec que resultaron de la simulación en archivos de Excel. Este proceso es necesario porque los resultados de los archivos .vec no son evidentes. El código del script de extracción de datos se encuentra en el ANEXO G.

Para obtener un análisis de los resultados y presentarlos de la forma que se propuso para este proyecto de titulación se desarrolló procesarDatos.py. Este script analiza los archivos de Excel del paso anterior uno por uno (existe un archivo de Excel por cada nodo de la simulación) y exporta 5 archivos: el primero llamado resultado.xlsx que resume todas las combinaciones posibles de throughput y retardo con número de nodos y velocidad de vehículos; los 4 archivos restantes almacenan los resultados propuestos para este proyecto de titulación: throughput vs número de nodos, throughput vs velocidad de vehículo, retardo vs número de nodos y retardo vs velocidad de vehículo. El código del script de procesamiento de datos se encuentra en el ANEXO H.

2.6 MÉTRICAS DE EVALUACIÓN

Para la evaluación del rendimiento de una red, dos parámetros muy importantes son el throughput y el retardo. Estos parámetros son los que se medirán para evaluar el comportamiento de nuestros escenarios de simulación al utilizar diferentes modelos de propagación, distintas velocidades y diferente numero de nodos.

2.6.1 THROUGHPUT

Está definido como el numero de bits recibidos con éxito por el nodo destino dividido por el total de tiempo de transmisión.

2.6.2 RETARDO

Generalmente este retardo se lo conoce como retardo extremo a extremo y constituye el tiempo promedio que transcurre desde que se inicia la transimision en el nodo fuente hasta que se recibe la información en el nodo destino.

2.6.3 CÁLCULO DEL THROUGHPUT Y RETARDO EN INETMANET

En INETMANET se tienen medidores de throughput definidos en capa de aplicación, pero como nuestro análisis es en capa física se realiza modificaciones en el código fuente para su cálculo. El retardo no se encuentra implementado a nivel de capa física por lo que también se debe agregar líneas de código. La obtención de estas métricas de rendimiento de la red se realiza en el modelo de radio debido a que en ese modelo es donde se realiza todo el proceso de transmisión/recepción.

El throughput y retardo se obtienen en el método que finaliza la recepción de la trama de radio en el archivo radio.cc: Radio::endReception(). Y en los métodos ejecutados previamente, por defecto se va midiendo el tiempo de recepción de la trama. Estos métodos son Radio::startReception() y Radio::continueReception(). Estas métricas se guardan en vectores para luego ser procesados fácilmente por medio de scripts.

Para el cálculo del throughput se define en el archivo Radio.h los siguientes atributos:

```
1       double throug;
2       long numbits;
3       cOutVector throughputVect;
```

Código 2.4. Atributos añadidos para calcular el throughput

En el archivo Radio.cc en el que inicializa todos los parámetros se define el nombre de los vectores y se inicializa en cero los atributos añadidos.

```
1       retardot.setName("End-to-End Delay");
2       throughputVect.setName("Throughput");
3       numbits = 0;
4       throug = 0;
```

Código 2.5. Inicialización de parámetros

Finalmente, se realiza el cálculo del throughput con la división del número de bits recibidos, sobre el tiempo total de simulación y se guarda como vector:

```
1       numbits += radioFrame->getBitLength();
2       throug = numbits/simTime();
3       throughputVect.record(throug);
```

Código 2.6. Cálculo del throughput

Para el cálculo del retardo se utiliza una forma alternativa de obtención de vectores. En el archivo Radio.h se define como protected al vector de salida:

```
1       cOutVector retardot;
```

Código 2.7. Atributos añadidos a Radio.h para calcular el retardo.

En el archivo Radio.cc en el método endReception() se calcula el retardo y se guarda como un vector:

```
1       simtime_t delay = simTime() - radioFrame->getCreationTime()  ;
                simtime_t duration = radioFrame->getDuration() +
2       delay;
3       retardot.record(duration);
```

Código 2.8. Cálculo del retardo.

2.6.4 CÁLCULO DE THROUGHPUT Y RETARDO EN VEINS

OMNeT++ dispone módulos con la capacidad de almacenar el throughput y el retardo. Sin embargo, Veins no posee esta funcionalidad; por este motivo, se procedió con la edición del código con la finalidad de obtener y procesar información de throughput y retardo.

Modificaciones a la clase BasePhyLayer se ven reflejadas en todas sus subclases. PhyLayer80111p posee todos los atributos y modificaciones efectuadas a los métodos debido a que se trata de una subclase de BasePhyLayer. En virtud de ello, se modificó el código de algunos métodos del archivo BasePhyLayer.c para que se realice el cálculo del throughput y retardo. El throughput se calcula como la razón entre la longitud de las tramas y el tiempo que se requiere para transmitirlas, y el retardo extremo-extremo es el tiempo transcurrido desde la salida del primer bit hasta el arrivo del último bit al receptor.

Los resultados del cómputo se almacenan en atributos de clase agregados en BasePhyLayer.h, en el Código 2.9, se muestran los atributos que se añadieron para almacenar el resultado de los cálculos de throughput y retardo.

Los vectores de OMNeT++ almacenan parejas de métricas y tiempo de ocurrencia. Las variables throughputVec y retrasoTotal son atributos de tipo cOutVector (vectores), que se emplearon para almacenar las variaciones de throughput y retardo en el tiempo.

```
1       private:
2           int numeroBits;
3           double tiempoSim;
4           double retardo;
5           cOutVector throughputVec;
6           cOutVector retrasoTotal;
```

Código 2.9. Atributos añadidos a BasePhyLayer.h para calcular throughput y retardo.

Los atributos numeroBits, tiempoSim y retardo se utilizan para realizar el cálculo del throughput y el retardo extremo a extremo.

Los métodos que fueron alterados para realizar la estimación de throughput y retardo son handleMessage(), handleAirFrameStartReceive() y handleAirFrameEndReceive(). Nótese que los métodos handleAirFrameStartReceive() y handleAirFrameEndReceive() son invocados desde handleMessage().

En el Código 2.10 se muestra una instrucción agregada sobre el código original en el método handleMessage() para almacenar la cantidad de bits que se han recibido: numeroBits+=PK(msg)->getBitLength();

```
1       ************Determino el numero de bits de la trama ********
2       numeroBits+=PK(msg)->getBitLength();
3       handleAirFrame(static_cast<AirFrame*>(msg));
4   ****************************************************
```

Código 2.10. Modificaciones al método handleMessage()

El método handleAirframe(), recibe la trama y revisa su estado. Existen 3 posibles estados: START_RECEIVE, RECEIVING y END_RECEIVE que disparan la ejecución de handleAirFrameStartReceive(), handleAirFrameReceiving() y handleAirFrameEndReceive() respectivamente.

Las AirFrame tienen el estado START_RECEIVE la primera vez que se reciben y por lo tanto son procesadas por handleAirFrameStartReceive(). En dicho método se optó por introducir una instrucción para capturar información del retardo en un atributo de la clase del mismo nombre propio de la clase BasePhyLayer. El retraso total consiste en la combinación del retardo de propagación y la duración de la AirFrame. Antes de que la ejecución de este método finalice, se cambia el estado de START_RECEIVE a RECEIVING y por lo tanto, la trama se procesa con handleAirFrameReceiving(), método en el que no se insertó código y que cambia el estado de RECEIVING a END_RECEIVE.

```
simtime_t delay = simTime() - s.getSendingStart();

**********Almaceno el retardo de propagación*****************

//simtime_t tiempoTx=s.getDuration();

retardo=delay.dbl()+s.getDuration().dbl();
s.setPropagationDelay(delay);
```

Código 2.11. Modificaciones al método handleAirFrame

Las tramas que tienen el estado END_RECEIVE son procesadas por handleAirFrameEndReceive(). En el método mencionado se agregó el código necesario para estimar el retardo y el throughput y registrar sus cambios en el tiempo. Sus modificaciones se muestran en el Código 2.12.

El tiempo de recepción de la trama se almacena en el atributo tiempoSim con la instrucción: tiempoSim= simTime().dbl(). El throughput es el cociente entre la cantidad de bits transmitidos y el tiempo transcurrido. Se insertó este cálculo: double velocidad=numeroBits/tiempoSim;.

El método record() de la clase vector permite almacenar variables asociadas a su tiempo de ocurrencia. Se introdujo código para almacenar el throughput y el retardo en forma de vectores en los atributos throughputVec y retrasoTotal.

```
    tiempoSim= simTime().dbl();

    double velocidad=numeroBits/tiempoSim;

    throughputVec.record(velocidad);
    retrasoTotal.record(retardo);
```

Código 2.12. Modificaciones al método handleAirFrameEndReceive().

3. RESULTADOS Y DISCUSIÓN

3.1 CAPA FÍSICA DE INETMANET

La capa física de INETMANET para el estándar IEEE 802.11a se encuentra implementada a detalle y cumple con las especificaciones del estándar IEEE 802.11 para OFDM. Toda esta implementación se encuentra en el directorio bitlevel de ieee80211 de capa física.

En la Tabla 3.1 se muestra una comparación entre INETMANET y las definiciones de IEEE 802.11a.

Tabla 3.1. Comparación entre INETMANET y el estándar IEEE 802.11.

Elemento	INETMANET	Estándar
Transmisor	Se encapsula la trama MAC recibida formando la trama PLCP. Se realiza procesos de aleatorización, codificación, entrelazado, modulación y formación del símbolo OFDM en el campo de datos de la trama PLCP. Se realiza procesos de codificación convolucional con R=1/2, entrelazado, modulación con BPSK y formación del símbolo OFDM en el campo de señal de la trama PLCP.	Se encapsula la trama MAC recibida formando la trama PLCP. Se realiza procesos de aleatorización, codificación, entrelazado, modulación y formación del símbolo OFDM en el campo de datos de la trama PLCP. Se realiza procesos de codificación convolucional con R=1/2, entrelazado, modulación con BPSK y formación del símbolo OFDM en el campo de señal de la trama PLCP.
Receptor	Se desencapsula la trama PLCP para obtener la trama MAC. Se realiza los procesos inversos tanto en el campo de señal como en el campo de datos de la trama PLCP.	Se desencapsula la trama PLCP para obtener la trama MAC. Se realiza los procesos inversos tanto en el campo de señal como en el campo de datos de la trama PLCP.
Aleatorizador/ Desaleatorizador	Utiliza un generador polinomial $S(x) = x^7 + x^4 + 1$	Utiliza un generador polinomial $S(x) = x^7 + x^4 + 1$
Codificador/ Decodificador convolucional	Se utiliza un polinomio generador es G0: 133 y G1: 171.	Se utiliza un polinomio generador es G0: 133 y G1: 171.
Entrelazado/ Desentrelazado	Entrelazado y desentrelazado definen dos permutaciones.	Entrelazado y desentrelazado definen dos permutaciones.
Modulación/ Demodulación	Utiliza modulaciones BPSK, QPSK, 16QAM y 64QAM para el campo de datos de la trama PLCP. Utiliza modulacion BPSK para el campo de señal de la trama PLCP.	Utiliza modulaciones BPSK, QPSK, 16QAM y 64QAM para el campo de datos de la trama PLCP. Utiliza modulacion BPSK para el campo de señal de la trama PLCP.

Elemento	INETMANET	Estándar
OFDM	El símbolo OFDM se forma por subportadoras piloto y 48 subportadoras de datos (0-47)	El símbolo OFDM se forma por: 4 subportadoras piloto, 48 subportadoras de datos (0-47)

3.1.1 VALIDACIÓN DEL PROCESO DE TRANSMISIÓN/RECEPCIÓN.

La validación del proceso de transmisión/recepción se realiza en base a las simulaciones con la capa física detallada. En la Figura 3.1 se observa un ejemplo de los mensajes intercambiados entre hosts y nodo. El host[1] envía una solicitud de ping hacia host [0], host[2] y nodo. Y nodo envía un ACK y una respuesta al ping hacia todos los hosts. Finalmente el host[1] confirma la recepción de la respuesta con un ACK.

```
host[0] --> host[1]   ping10                PING req 10.0.0.1 to 10.0.0.3
host[0] --> nodo      ping10                PING req 10.0.0.1 to 10.0.0.3
nodo --> host[0]      ACK                   WLAN  ack 0A-AA-00-00-00-01
nodo --> host[1]      ACK                   WLAN  ack 0A-AA-00-00-00-01
nodo --> host[0]      parsed-ping10-reply   PING reply 10.0.0.3 to 10.0.0.1
nodo --> host[1]      parsed-ping10-reply   PING reply 10.0.0.3 to 10.0.0.1
host[0] --> host[1]   ACK                   WLAN  ack 0A-AA-00-00-00-03
host[0] --> nodo      ACK                   WLAN  ack 0A-AA-00-00-00-03
```

Figura 3.1. Mensajes intercambiados entre host y nodo.

Los mensajes enviados de PING request y PING reply son encapsulados en capa de red, capa MAC y capa física, en donde se forma la trama PLCP formada por 125 bytes y una duración de 48us como se observa en la Figura 3.2.

```
inet::physicallayer::Ieee80211OFDMPLCPFrame:125 bytes     duration=0.048ms
```

Figura 3.2. Duración de la trama PLCP.

Los ACK enviados también se encapsulan en las capas correspondientes, y en capa física forman una trama PLCP con 19bytes de longitud y 24us de duración.

```
inet::physicallayer::Ieee80211OFDMPLCPFrame:19 bytes      duration=0.024ms
```

Figura 3.3. Duración del ACK.

Para comenzar con la transmisión se verifica que el radio este en modo TRANSMITTER y el estado de transmisión este en IDLE como se muestra en la Figura 3.21.

```
Radio mode changed from RECEIVER to TRANSMITTER
```

```
Changing radio reception state from IDLE to UNDEFINED.
Changing radio transmission state from UNDEFINED to IDLE.
```

Figura 3.4. Modo de transmisión IDLE.

Luego que se verifica el canal libre se comienza con el envío de la trama de radio hacia el canal inalámbrico, esto se verifica en la Figura 3.5.

```
Sending (inet::physicallayer::RadioFrame)ping10 with 1000 bits in 48 us transmission duration
from (inet::physicallayer::APSKRadio)radio, antenna = { IsotropicAntenna }, transmitter = {
Ieee80211LayeredOFDMTransmitter }, receiver = { Ieee80211LayeredOFDMReceiver } ...
Sending (inet::physicallayer::RadioFrame)ping10 from (inet::physicallayer::APSKRadio)radio,
antenna = { IsotropicAntenna }, transmitter = { Ieee80211LayeredOFDMTransmitter }, receiver
= { Ieee80211LayeredOFDMReceiver } ...
Sending (inet::physicallayer::RadioFrame)ping10 from (inet::physicallayer::APSKRadio)radio,
antenna = { IsotropicAntenna }, transmitter = { Ieee80211LayeredOFDMTransmitter }, receiver
= { Ieee80211LayeredOFDMReceiver } ...
```

Figura 3.5. Envío de la trama al canal inalámbrico.

Mientras se envía la información, el modo de radio cambia a TRANSMITING y el indicador de la señal transmitida cambia de estado a WHOLE, este procedimiento se puede verificar en la Figura 3.6.

```
Transmission started: (inet::physicallayer::RadioFrame)ping10 WHOLE as
LayeredTransmission, ...
Changing radio transmission state from IDLE to TRANSMITTING.
Changing radio transmitted signal part from NONE to WHOLE.
```

Figura 3.6. Estado TRANSMITTING.

Para que la trama de radio sea transmitida se debe realizar los procesos de aleatorización, codificación, entrelazado, modulación en el campo de datos, y en el campo de señal los mismos procesos excepto la aleatorización. La Figura 3.7 muestra los resultados de estos procesos y comprueba que en capa física se están realizando los procesos indicados según el estándar.

```
Encoding the following bits: 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
The encoded bits are: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 1 0 0 1 1 0
1 0 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
FEC encoded bits are: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 1 0 0 1 1 0
1 0 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
Interleaving the following bits: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 1
0 0 1 1 0 1 0 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
The interleaved bits are: 0 1 0 0 1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0
0 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0
Interleaved bits are: 0 1 0 0 1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0
1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0
Modulated OFDM symbol: (-1,0) (1,0) (-1,0) (-1,0) (1,0) (1,0) (1,0) (-1,0)
(1,0) (1,0) (-1,0) (-1,0) (1,0) (-1,0) (-1,0) (-1,0) (-1,0) (1,0) (-1,0)...
```

```
Scrambling the following bits: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 ...
The scrambled bits are: 0 1 1 0 1 1 0 0 0 0 0 1 1 0 0 1 1 0 1 1 1 0 0 1 1 1 0
0 1 1 1 1 0 1 1 0 1 0 0 0 0 1 0 1 0 1 0 1 1 0 1 0 0 1 0 0 1 1 1 1 0 1 1 0 ...
Scrambled bits are: 0 1 1 0 1 1 0 0 0 0 0 1 1 0 0 1 1 0 1 1 1 0 0 1 1 1 0 0 1
1 1 1 0 1 1 0 1 0 0 0 0 1 0 1 0 1 0 1 1 0 1 0 0 1 0 0 1 1 1 1 0 1 1 0 0 1 ...
Encoding the following bits: 0 1 1 0 1 1 0 0 0 0 0 1 1 0 0 1 1 0 1 1 1 0 0 1 1
1 0 0 1 1 1 1 0 1 1 0 1 0 0 0 0 1 0 1 0 1 0 1 1 0 1 0 0 1 0 0 1 1 1 1 0 1 ...
The encoded bits are: 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1
0 0 1 0 1 1 0 0 0 0 1 1 0 0 0 1 0 1 0 1 1 0 0 1 1 0 1 1 1 0 0 0 0 0 0 1 1 ...
FEC encoded bits are: 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1
0 0 1 0 1 1 0 0 0 0 1 1 0 0 0 1 0 1 0 1 1 0 0 1 1 0 1 1 1 0 0 0 0 0 0 1 1 ...
Interleaving the following bits: 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0
1 0 0 0 1 0 0 1 0 1 1 0 0 0 0 1 1 0 0 0 1 0 1 0 1 1 0 0 1 1 0 1 1 1 0 0 0 ...
The interleaved bits are: 0 1 1 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 1 1 1 0 0 1 1 1
0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 1 1 0 0 0 0 0 0 ...
Interleaved bits are: 0 1 1 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 1 1 1 0 0 1 1 1 0 0
0 0 1 0 0 0 0 0 0 0 1 0 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 1 1 0 0 0 0 0 0 0 1 ...
Modulated OFDM symbol: (-0.316228,0.316228) (0.948683,0.948683) (0.948683,-
0.316228) (-0.948683,-0.316228) (0.316228,0.316228) (1,0) (0.948683,-0.316228)
(0.316228,-0.948683) (-0.948683,0.948683) (-0.948683,-0.948683) ...
Modulated OFDM symbol: (0.316228,-0.316228) (0.948683,0.316228)
(0.316228,0.316228) (0.316228,-0.316228) (-0.316228,-0.316228) (1,0)
(0.316228,0.316228) (-0.948683,0.316228) (-0.948683,-0.948683) ...
Modulated OFDM symbol: (-0.948683,0.948683) (0.948683,-0.316228) (-
0.316228,0.316228) (-0.948683,0.948683) (0.948683,0.948683) (1,0) (0.316228,-
0.948683) (0.948683,-0.316228) (-0.948683,0.316228) (-0.316228,-0.316228) ...
Modulated OFDM symbol: (-0.948683,-0.948683) (0.948683,0.316228) (-0.316228,-
0.316228) (-0.316228,-0.948683) (0.316228,0.948683) (1,0) (-0.316228,-
0.316228) (0.948683,-0.948683) (0.948683,0.316228) (-0.948683,0.316228) ...
Modulated OFDM symbol: (0.948683,0.316228) (-0.316228,-0.948683)
(0.948683,0.948683) (0.316228,0.948683) (-0.316228,-0.316228) (-1,0)
(0.948683,0.948683) (-0.948683,0.948683) (0.948683,0.316228) ...
Modulated OFDM symbol: (0.948683,0.316228) (0.316228,0.948683) (-
0.316228,0.948683) (0.948683,0.948683) (0.316228,0.316228) (-1,0) (-
0.948683,0.316228) (-0.316228,-0.316228) (0.316228,0.948683)...
Modulated OFDM symbol: (0.316228,0.316228) (-0.948683,-0.948683)
(0.316228,0.316228) (0.948683,0.316228) (-0.316228,-0.948683) (-1,0) (-
0.948683,0.948683) (0.948683,0.316228) (0.316228,-0.316228)...
```

Figura 3.7. Procesos ejecutados en transmisión.

Luego se genera un evento que indica que la trama de radio ha sido enviada como se muestra en la Figura 3.8.

```
** Event #2132 t=4.219684158702 prueba2.host[0].wlan[0].radio (APSKScalarRadio,
id=52)  on parsed-ping10-reply (inet::physicallayer::RadioFrame, id=3852)
```

Figura 3.8. Trama enviada

Finalizado el envío el temporizador expira y el modo de radio cambia a IDLE.

```
** Event #2134  t=4.219731779814  prueba2.nodo.wlan[0].radio (APSKScalarRadio,
id=163)  on selfmsg transmissionTimer (omnetpp::cMessage, id=20)
Transmission ended: (inet::physicallayer::RadioFrame)parsed-ping10-reply WHOLE
as LayeredTransmission, ...
```

```
Changing radio transmission state from TRANSMITTING to IDLE.
prueba2.nodo.wlan[0].mac.tx: Tx: radioTransmissionFinished()
prueba2.nodo.wlan[0].mac.upperMac: Transmission complete
prueba2.nodo.wlan[0].mac.upperMac: Doing step 2
prueba2.nodo.wlan[0].mac.upperMac:     doStep()     in     step=2     performed
expectReplyStartTxWithin()
prueba2.nodo.wlan[0].mac.rx: Setting NAV to 0.00006
prueba2.nodo.wlan[0].mac: changing radio to receiver mode
Changing radio transmitted signal part from WHOLE to NONE.
```

Figura 3.9. Finalización de la transmisión

Para realizar la recepción el radio debe estar en modo RECEIVER y el indicador de señal transmitida está en estado NONE.

```
prueba2.nodo.wlan[0].mac: changing radio to receiver mode
Changing radio transmitted signal part from WHOLE to NONE.
** Event #2135  t=4.219731779814  prueba2.nodo.wlan[0].radio (APSKScalarRadio,
id=163)  on configureRadioMode (omnetpp::cMessage, id=3862)
Radio mode changed from TRANSMITTER to RECEIVER
```

Figura 3.10. Estado RECEIVER

Entonces el medio comienza a escuchar el canal y para la recepción el estado de recepción cambia a IDLE mientras que el de trasmisión a UNDEFINED.

```
Radio mode changed from TRANSMITTER to RECEIVER
Noise power begin
Noise at 4.219731779814 = 1e-14 W
Noise at 4.219731779815 = 0 W
Noise power end
Computing listening possible: maximum power = 1e-14 W, energy detection =
6.30957e-10 W -> listening is impossible
Listening with BandListening, carrierFrequency = 5.765e+09 Hz, bandwidth = 2e+07
Hz, receiverId = 2, startTime = 4.219731779814, endTime = 4.219731779815, ...
Changing radio reception state from UNDEFINED to IDLE.
Changing radio transmission state from IDLE to UNDEFINED.
```

Figura 3.11. Estado UNDEFINED.

Cuando se comienza con la recepción el estado de recepción cambia a RECEIVING y el estado de la señal recibida cambia a WHOLE.

```
Receiving LayeredTransmission, packetModel = { SignalPacketModel }, bitModel =
{ SignalBitModel }, symbolModel = { SignalSymbolModel }, ...
Computing reception possible: minimum reception power = 3.31818e-10 W,
sensitivity = 3.16228e-10 W -> reception is possible
Reception started: attempting (inet::physicallayer::RadioFrame)parsed-ping10-
reply WHOLE as LayeredReception, ...
Changing radio reception state from IDLE to RECEIVING.
prueba2.host[0].wlan[0].mac.ctn[0]:     handleWithFSM:     processing     event
MEDIUM_STATE_CHANGED
```

```
prueba2.host[0].wlan[0].mac.ctn[0]: FSM fsm: processing event in state IDLE
prueba2.host[0].wlan[0].mac.ctn[0]:         FSM         fsm:         condition
"event==MEDIUM_STATE_CHANGED" holds, staying in current state
Changing radio received signal part from NONE to WHOLE.
```

Figura 3.12. Estado RECEIVING.

Al comenzar la recepción se inicia el temporizador y se verifica si la recepción es posible en base a la potencia de recepción y la sensibilidad. Si esta potencia esta sobre la sensibilidad es posible la recepción.

```
**    Event    #2136       t=4.219732158702       prueba2.host[0].wlan[0].radio
(APSKScalarRadio,  id=52)    on  selfmsg  receptionTimer  (omnetpp::cMessage,
id=3859)
Noise power begin
Noise at 4.219684158702 = 1e-14 W
Noise at 4.219732158702 = 0 W
Noise power end
Computing  reception  possible:  minimum  reception  power  =  3.31818e-10  W,
sensitivity = 3.16228e-10 W -> reception is possible
Computing  reception  possible:  minimum  reception  power  =  3.31818e-10  W,
sensitivity = 3.16228e-10 W -> reception is possible
Receiving LayeredTransmission, packetModel = { SignalPacketModel }, bitModel =
{ SignalBitModel }, symbolModel = { SignalSymbolModel }, ...
Reception  ended:  successfully  for  (inet::physicallayer::RadioFrame)parsed-
ping10-reply WHOLE as LayeredReception, ...
```

Figura 3.13. Comienzo de la recepción.

En recepción los procesos realizados para el campo de señal y el campo de datos son los inversos a los indicados en la transmisión. En la Figura 3.14 es posible comprobar que todos estos procesos se realizan y los resultados son mostrados.

```
The field symbols has been demodulated into the following bit stream: 0 1 0 0
1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0
0 0 0 1 0
Deinterleaving the following bits: 0 1 0 0 1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 1 0 0
0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0
The deinterleaved bits are: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 1 0 0
1 1 0 1 0 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
Decoding the following bits: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 1 0 0
1 1 0 1 0 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
Recovered message: 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
Number of errors: 0 Cumulative error (Hamming distance): 0 End state: 0
The field symbols has been demodulated into the following bit stream: 0 1 1 1
1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 1 1 1 0 0 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 ...
The field symbols has been demodulated into the following bit stream: 1 1 0 1
1 0 1 1 1 1 1 1 1 1 0 1 0 1 0 1 1 1 1 1 1 0 0 1 1 0 0 0 0 0 0 1 1 1 0 0 0 0 ...
The field symbols has been demodulated into the following bit stream: 0 0 1 0
1 0 0 1 0 1 1 1 0 0 1 0 1 0 1 0 1 1 0 0 1 0 0 1 0 0 1 1 0 1 0 1 1 0 1 1 1 ...
```

```
The field symbols has been demodulated into the following bit stream: 0 0 0 0
1 0 1 1 0 1 0 1 0 1 0 0 1 1 1 0 0 1 0 1 1 0 0 0 1 0 1 1 0 0 1 1 1 0 1 1 0 ...
The field symbols has been demodulated into the following bit stream: 1 0 1 1
0 1 0 0 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 0 0 1 0 1 0 1 1 1 0 1 1 1 0 1 0 0 ...
The field symbols has been demodulated into the following bit stream: 1 0 1 1
1 1 1 0 0 1 1 0 1 0 1 0 1 1 1 1 0 0 1 1 0 1 0 1 1 1 1 0 0 1 1 1 0 0 0 0 1 ...
The field symbols has been demodulated into the following bit stream: 1 1 1 1
0 0 0 0 1 1 1 1 1 0 1 1 0 1 0 0 0 0 1 0 1 0 1 1 1 1 0 1 0 0 1 1 1 1 1 0 0 ...
Deinterleaving the following bits: 0 1 1 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 1 1 0
0 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 1 1 0 0 ...
The deinterleaved bits are: 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 1 0
0 0 1 0 0 1 0 1 1 0 0 0 0 1 1 0 0 0 1 0 1 0 1 1 0 0 1 1 0 1 1 1 0 0 0 0 0 ...
Decoding the following bits: 0 0 1 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1
0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 1 0 0 ...
Recovered message: 0 1 1 0 1 1 0 0 0 0 0 1 1 0 0 1 1 0 1 1 1 0 0 1 1 1 0 0 1 1
1 1 0 1 1 0 1 0 0 0 0 1 0 1 0 1 0 1 1 0 1 0 0 1 0 0 1 1 1 1 0 1 1 0 0 1 1 ...
Number of errors: 0 Cumulative error (Hamming distance): 0 End state: 12
Scrambling the following bits: 0 1 1 0 1 1 0 0 0 0 0 1 1 0 0 1 1 0 1 1 1 0 0 1
1 1 0 0 1 1 1 1 0 1 1 0 1 0 0 0 0 1 0 1 0 1 0 1 1 0 1 0 0 1 0 0 1 1 1 1 0 ...
The scrambled bits are: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 1 0 1 ...
Calculated CRC: 3425127028, received CRC: 3425127028
```

Figura 3.14. Proceso de recepción.

Una vez recibida esta información con todos los procesos indicados se desencapsula, y como resultado se obtiene la trama MAC que es enviada hacia la capa correspondiente.

```
Receiving LayeredTransmission, packetModel = { SignalPacketModel }, bitModel =
{ SignalBitModel }, symbolModel = { SignalSymbolModel }, ...
Sending up (inet::ieee80211::Ieee80211DataFrameWithSNAP)parsed-ping10-reply
```

Figura 3.15. Envío a la capa superior

Un evento es generado al enviar esta trama a la capa MAC y el receptor de capa MAC indica la recepción de la información de capa física.

```
**    Event    #2137        t=4.219732158702        prueba2.host[0].wlan[0].mac
(Ieee80211CompatibleMac,       id=51)             on       parsed-ping10-reply
(inet::ieee80211::Ieee80211DataFrameWithSNAP, id=3871)
Received frame from PHY: (inet::ieee80211::Ieee80211DataFrameWithSNAP)parsed-
ping10-reply
```

Figura 3.16. Trama recibida.

3.2 CAPA FÍSICA DE VEINS

Veins captura con precisión la duración de las tramas, modulación y modelos físicos, procesos como interleaving y aleatorización se sustituyen con retrasos. La comparación entre Veins y el Estándar 802.11p se muestra en la Tabla 3.2.

Tabla 3.2. Comparación entre Veins e IEEE 802.11p

Elemento	Veins	Estándar IEEE 802.11p
Transmisor	Se representa las PLCP con tiempos de duración. La representación física de una señal se encuentra en la clase Signal.	Se realiza procesos de aleatorización, codificación, entrelazado, modulación y formación del símbolo OFDM en el campo de datos de la trama PLCP.
Receptor	En el receptor se utilizan las clases ChannelInfo Y Signal para estimar cuales son las tramas erradas y cuales están correctas.	Se desencapsula la trama PLCP para obtener la trama MAC. Se realiza los procesos inversos tanto en el campo de señal como en el campo de datos de la trama PLCP.
Aleatorizador/ Desaleatorizador	Este proceso no se lleva a cabo, para sustituirlo se toma en cuenta un retraso (este retraso está incluido en el tiempo de duración de la PLCP)	Utiliza un generador polinomial
Entrelazado	Este proceso no se lleva a cabo, para sustituirlo se toma en cuenta un retraso (este retraso está incluido en el tiempo de duración de la PLCP)	Polinomios generadores G0: 133 y G1: 171.
Modulación/Demodulación	Se emula un proceso de modulación con "mappings". No se trata de un proceso de modulación real.	BPSK, QPSK, 16QAM y 64QAM

3.2.1 VALIDACIÓN DEL PROCESO DE TRANSMISIÓN/RECEPCIÓN.

Se evaluó la ejecución de las simulaciones donde se diferencian 3 pasos. En el primero se configuran los parámetros de SUMO y Veins, en el segundo se inicia con el mismo y finalmente se ejecuta OMNeT++ en modo consola. Todos estos procedimienetos fueron automatizados y evitan a necesidad de abrir las GUI.

En la Figura 3.17 se puede apreciar la ejecución de SUMO y OMNeT++ en consola a través de un script desarrollado para la ejecución simultánea de ambas aplicaciones sin intervención manual. El script en mención elige una semilla, inicia duaraouter (usado para crear rutas en SUMO), levanta el puerto de comunicación 9999 e inicia OMNeT++.

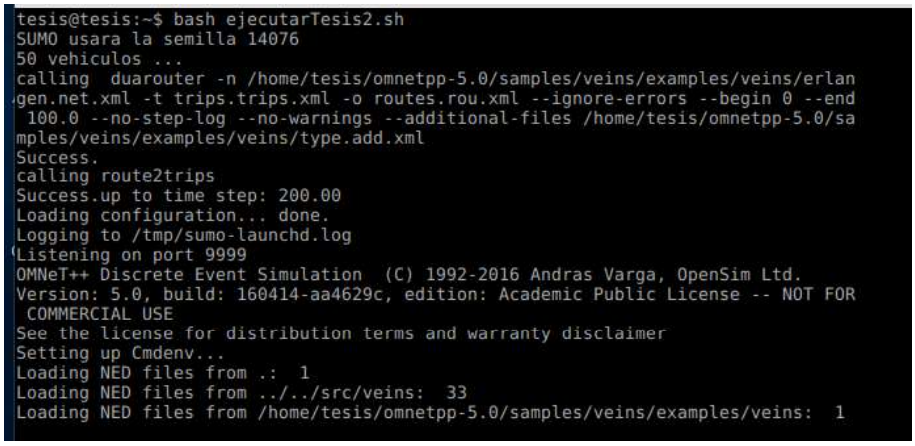

Figura 3.17. Ejecución de script para simulaciones on Veins.

En la Figura 3.18 se expone el inicio de la comunicación entre SUMO y OMNeT++ y el registro de eventos durante la ejecución de la simulación. Cada mensaje está identificado con un número de evento y el instante de ourrencia. El porcentaje de progreso de la simulación también es visible en el extremo derecho de cada línea.

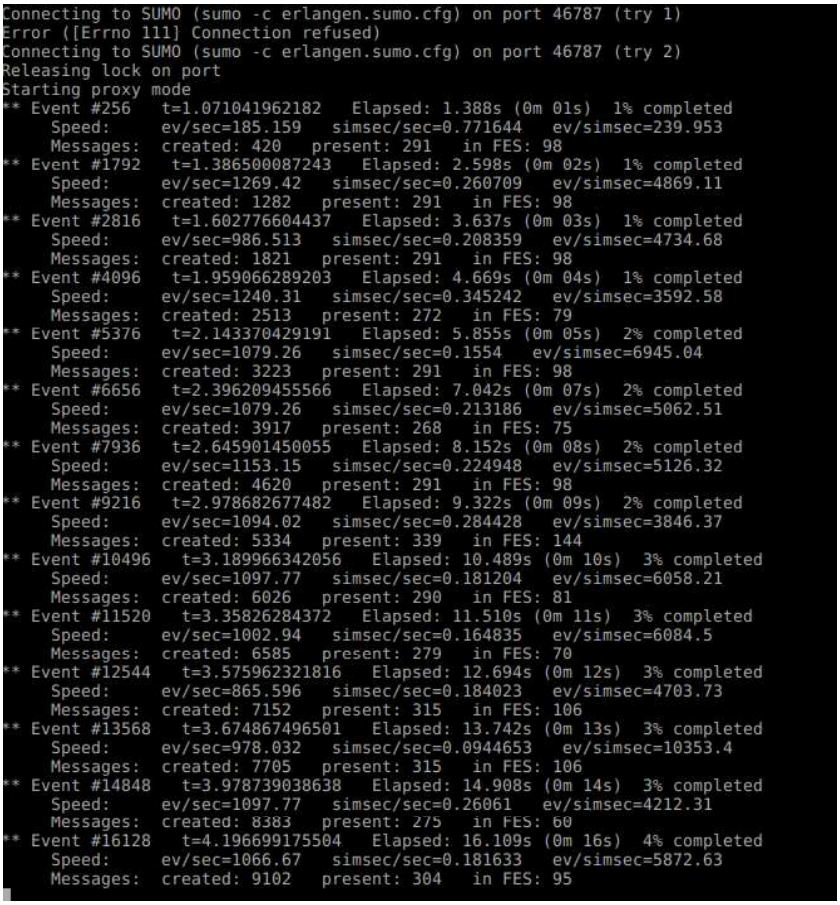

Figura 3.18. Simulación de OMNeT++ a través de consola de Linux.

3.3 SIMULACIONES EN INETMANET

Se ejecutan varias simulaciones de prueba con tráfico TCP y UDP pero no es posible su transformación a bits debido a que no se tiene implementado el serializador correspondiente. Adicionalmente se hacen pruebas con 10, 20, 30, 40, 50, 60, 80 y 100 nodos y las simulaciones se detienen en menos de un segundo.

Debido a la gran cantidad de procesos realizados tanto en el transmisor como en el receptor no se logra capturar las tramas MAC en el receptor. Se probaron 18 simulaciones con los parámetros mencionados para cada modelo de propagación con las velocidades de 4, 8, 12, 16 y 20 m/s, en total 1080 simulaciones.

Puesto que no fue posible enviar trafico TCP y UDP, se ejecutan las mismas simulaciones para el protocolo ICPM de capa de red con 2, 4, 5, 7, 8, 10, 20, 30, 40, 50, 60, 80 y 100 nodos. A partir de los 8 nodos se tiene los mismos problemas al capturar las tramas MAC en el receptor, por lo que se define 2, 3, 4, 6 y 8 nodos móviles para las simulaciones a realizarse.

Nuestro escenario de simulación parte del ejemplo Layered80211, el mismo que está configurado para una comunicación de dos hosts en modo ad hoc con el estándar IEEE 802.11g y con 5s de simulación, el cual envía trafico ICMP entre hosts y permite configurar la capa física de IEEE 802.11 con OFDM de forma detallada. Se debe destacar que el ejemplo indica que IEEE 802.11 con OFDM está diseñado para soportar una comunicación nodo a nodo y no con gran cantidad de nodos. Igualmente, el tiempo de simulación es corto debido a todo el procesamiento realizado en capa física.

Para estas simulaciones se configuran escenarios con diferentes números de nodos, diferentes velocidades, se obtienen valores de throughput y retardo para los modelos de propagación. El retardo y throughput se almacenan en archivos .vec para cada uno de los nodos.

Los archivos.vec obtenidos son procesados por los scripts desarrollados, los cuales permiten el cálculo del promedio y su graficación.

3.3.1 THROUGHPUT VS VELOCIDAD

En la Tabla 3.3. Throughput vs. Velocidad para distintos modelos de propagación. se muestran los resultados obtenidos para throughput en las distintas velocidades para todos los modelos de propagación utilizados. Como se muestra en la Figura 3.19 se observa que al aumentar la velocidad el throughput decae. Entre modelos la variación va decreciendo

ya que en los modelos se considera más fenomenos de propagación, pero no existe mucha diferencia entre el modelo FreeSpace y de dos rayos.

Tabla 3.3. Throughput vs. Velocidad para distintos modelos de propagación.

Modelo / Vel. (m/s)	Throughput Free Space (Kbps)	Throughput Two Ray (Kbps)	Throughput Rician (Kbps)	Throughput Rayleigh (Kbps)	Throughput Log Normal (Kbps)	Throughput Nakagami (Kbps)
4	21,32	21,32	16,91	13,34	19,39	12,42
8	20,96	20,96	16,69	12,08	18,56	15,45
12	20,81	20,81	15,86	11,91	17,66	14,07
16	20,07	20,07	15,00	12,20	17,44	13,71
20	17,39	18,80	14,62	12,31	14,22	13,92

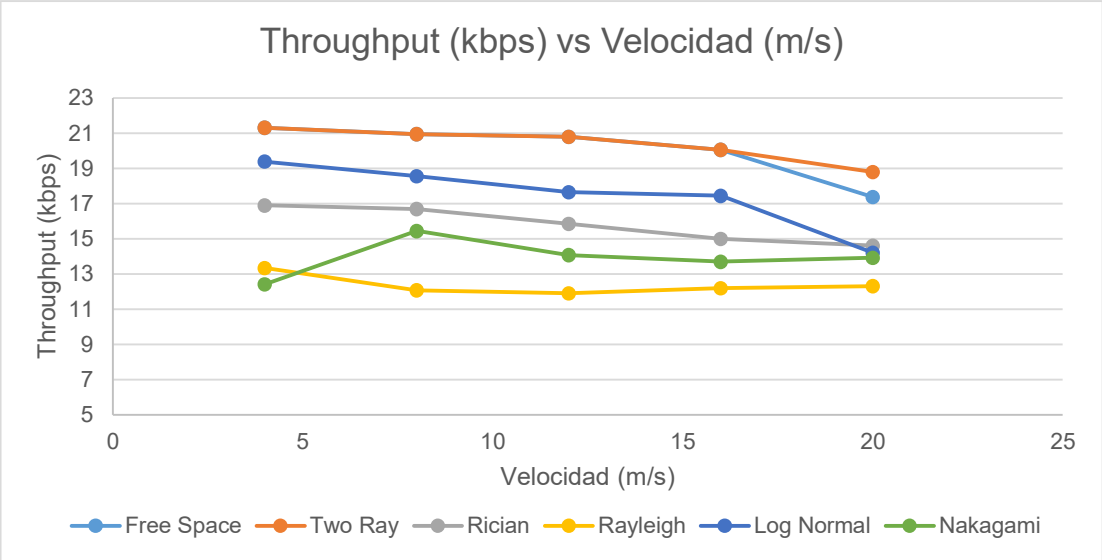

Figura 3.19. Gráfica de Throughput vs. Velocidad.

3.3.2 THROUGHPUT VS NÚMERO DE NODOS

En la Tabla 3.4. Throughput vs. Cantidad de nodos para distintos modelos de propagación. se muestran los resultados obtenidos para throughput en base al número de nodos. Como se muestra en la Figura 3.20 se observa que al aumentar el número de nodos el throughput obviamente aumenta que existe mayor cantidad de nodos que envían información.

Entre modelos la variación entre el FreeSpace y TwoRayGround no es representativa, mientras que entre los demás modelos al considerar más fenómenos el throughput decae.

Tabla 3.4. Throughput vs. Cantidad de nodos para distintos modelos de propagación.

Modelo / Nodos	Throughput Free Space (Kbps)	Throughput Two Ray (Kbps)	Throughput Rician (Kbps)	Throughput Rayleigh (Kbps)	Throughput Log Normal (Kbps)	Throughput Nakagami (Kbps)
3	13,68	15,63	8,31	7,34	9,01	9,58
4	21,43	21,43	16,81	13,71	18,44	14,62
5	16,20	16,20	12,69	10,19	14,70	11,15
6	23,13	23,13	17,30	12,61	20,95	13,14
7	26,11	25,57	23,96	18,00	24,19	19,66

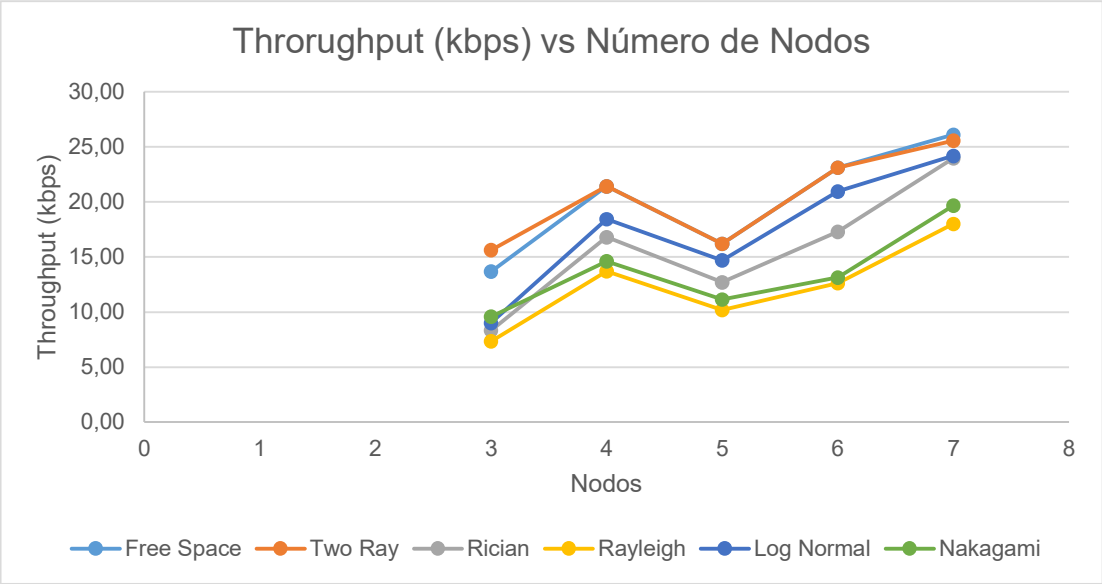

Figura 3.20. Gráfica de Throughput vs. Número de nodos.

3.3.3 RETARDO VS VELOCIDAD DEL NODO.

En la Tabla 3.5. Retardo vs. Velocidad para distintos modelos de propagación. se muestran los resultados obtenidos de throughput en bps para todos los modelos de propagación utilizados a diferentes velocidades.

Como se ve en la Figura 3.21, el retardo es casi constante y entre modelos no hay ninguna variación. Al aumentar la velocidad el retardo disminuye ya que los nodos se van aproximando uno a otro más rápidamente existiendo una distancia menor para su comunicación.

Tabla 3.5. Retardo vs. Velocidad para distintos modelos de propagación.

Modelo / Vel. (m/s)	Retardo Free Space (us)	Retardo Two Ray (us)	Retardo Rician (us)	Retardo Rayleigh (us)	Retardo Log Normal (us)	Retardo Nakagami (us)
4	96,19	96,19	95,58	95,19	96,21	94,30
8	96,19	96,19	95,60	94,76	95,92	94,50
12	95,57	95,57	95,16	94,19	95,42	94,23
16	94,80	94,80	94,31	94,76	94,21	95,25
20	92,72	95,00	92,50	94,99	94,57	94,68

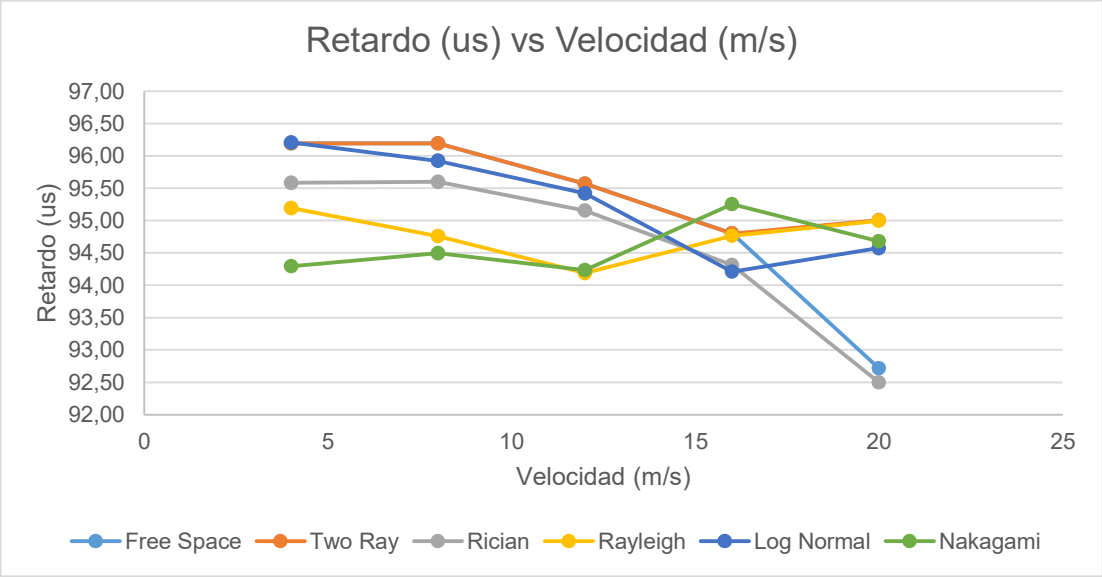

Figura 3.21. Gráfica de Retardo vs. Velocidad.

3.3.4 RETARDO VS NÚMERO DE NODOS.

En la Tabla 3.6 se muestran los resultados obtenidos de retardo en us para todos los escenarios con diferentes números de nodos. El retardo es casi constante durante la simulación y entre modelos no hay variación alguna como se muestra en la Figura 3.22.

Al aumentar el número de nodos la variación es muy pequeña, aunque se puede notar que al aumentar el número de nodos su retardo va reduciéndose debido a que existen más nodos dentro del rango de cobertura que se encontrarán a una distancia más pequeña de separación.

Tabla 3.6. Retardo vs. Cantidad de nodos para distintos modelos de propagación.

Modelo / Nodos	Retardo Free Space (us)	Retardo Two Ray (us)	Retardo Rician (us)	Retardo Rayleigh (us)	Retardo Log Normal (us)	Retardo Nakagami (us)
3	91,48	94,29	88,71	89,72	92,45	89,72
4	96,22	96,22	96,26	95,87	96,24	95,19
5	96,15	96,14	96,20	96,19	96,19	95,06
6	96,15	96,15	95,80	95,92	96,16	96,14
7	95,47	94,95	96,17	96,19	95,29	95,72

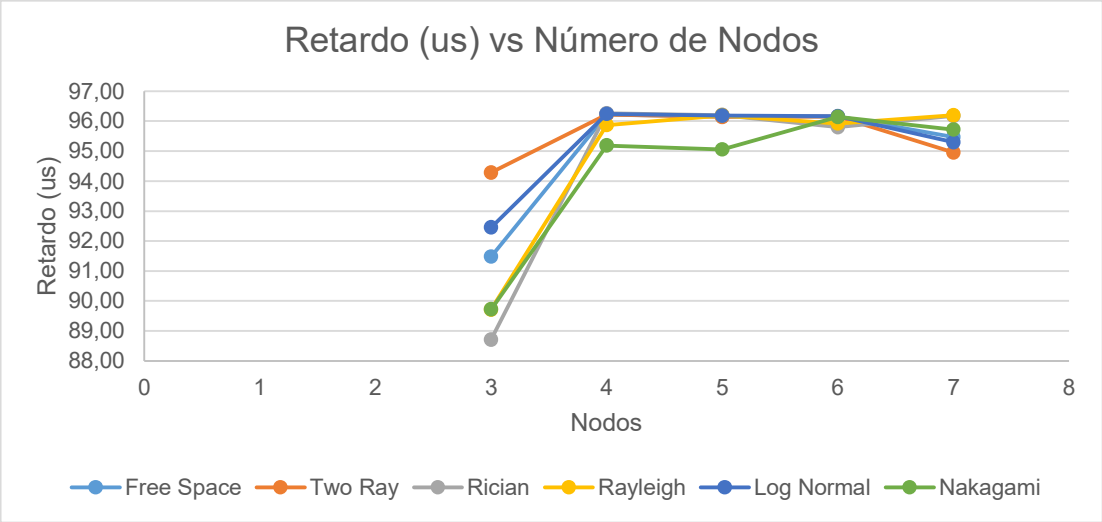

Figura 3.22. Gráfica de Retardo vs. Cantidad de nodos.

3.4 SIMULACIONES EN VEINS

El objetivo de las simulaciones consiste en observar el comportamiento de la red mediante las gráficas throughput y el retardo de los nodos cuando se utilizan diferentes combinaciones de cantidad de nodos y velocidad de estos. Se obtendrán graficas de: Throughput vs. Velocidad de desplazamiento, Throughput vs. Cantidad de nodos, retardo vs. Velocidad de desplazamiento y Retardo vs. Cantidad de Nodos. Para conseguir esto se agregó código para hacer el cálculo del throughput y el retardo con una cantidad variable de nodos a distintas velocidades de desplazamiento promedio (el detalle de las modificaciones se puede verificar en la sección 2.6.4).

Seis modelos se emplearon para realizar un total de 1200 simulaciones (200 simulaciones por modelo). Ejecutar una cantidad de simulaciones como esta manualmente resulta prácticamente imposible por métodos tradicionales. Por este motivo, se desarrolló una serie de scripts para automatizar la realización de las simulaciones y el análisis de los resultados.

Se utilizó una muestra de 25 simulaciones para hacer el análisis por limitaciones en el procesamiento de una cantidad de datos como la que genera una cantidad de simulaciones como esta. La muestra de las simulaciones incluye todas las combinaciones posibles de: 20,40,60,80 y 100 nodos a velocidades de 4,8,12,16 y 20 m/s. Para cada uno de estos escenarios, se evaluó: Throughput vs Velocidad, throughput vs número de nodos, retardo vs Velocidad y retardo vs número de nodos.

Entre los modelos de propagación (Analogue Models) empleados tenemos: SimplePathlossModel, BreakpointPathlossModel, TwoRayInterferenceModel, JakesFading, LogNormal Shadowing y NakagamiFading.

SimplePathloss Model es un modelo que considera atenuación en un espacio libre sin reflexiones o shadowing.

BreakpointPathlossModel usa SimplePathlossModel con parámetros que varían si se supera o no un umbral de distancia específico (llamado Breakpoint). Cuando no se supera el umbral, se usa closeRangeModel y cuando se excede, se emplea farRangeModel. Ambos modelos no son mas que SimplePathlossModel en configuraciones distintas para pequeña y larga distancia. El problema con esta aproximación es que aquellos valores que se encuentran justo debajo del umbral se someten a una atenuación muy distinta comparada con aquellos que se encuentran justo por encima de él. Otro defecto del modelo es que una señal que se encuentra muy distante se atenúa en la misma magnitud que una señal que apenas y supera el umbral de distancia. Por ejemplo, en un sistema con un umbral de 300m una señal que es transmitida a 900m del receptor experimenta la misma pérdida de potencia que una señal que se transmite a 301 metros de distancia.

TwoRayInterferenceModel es un modelo mas avanzado que considera la interferencia (constructiva o destructiva) de dos señales, una LOS que enlaza ambas antenas directamente en línea recta y una que se origina de la reflexión con el suelo. A diferencia del modelo tradicional que se menciona frecuentemente en libros de texto, Veins implementa una versión mas precisa que se desarrolló a partir de mediciones en campo por el creador de Veins, Cristoph Sommer. La versión de TwoRayInterferenceModel implementada en Veins no solo obtiene resultados mas realistas que la versión simplificada, también es capaz de capturar el efecto de una gran atenuación a corta distancia (normalmente causado por interferencia destructiva).

Rayleigh fading es una aproximación estadística que define la propagación de una señal en un medio inalámbrico, considerando que su magnitud varía, o mas bien, se atenúa de acuerdo a la distribución de Rayleigh. Esta aproximación es útil para estimar la propagación

en ambientes urbanos. El modelo de Jakes simula señales que se atenúan de acuerdo a la distribución de Rayleigh pero arrivan al receptor con la misma potencia y describiendo un círculo, separadas por un ángulo que les permita ser equidistantes una de otra.

Log Normal Shadowing es un modelo donde los niveles de potencia siguen una distribución lognormal. En una distribución lognormal, el logaritmo, está normalmente distribuido. Este modelo es útil para simular atenuación causada por obstáculos. Constituye una extensión del modelo de Friis (que está estructurado para un entorno libre de obstáculos).

Nakagami se utiliza para modelar la atenuación de señales inalámbricas que siguen múltiples caminos (paths). Es útil para simular escenarios con un relativamente alto retraso de propagación.

3.4.1 THROUGHPUT VS VELOCIDAD

En la Tabla 3.7 se tiene un resumen de los resultados de las simulaciones, y en la Figura 3.23 se muestra un gráfico comparativo del throughput calculado con distintos modelos de propagación.

Tabla 3.7. Throughput vs. Velocidad para distintos modelos de propagación.

Modelo / Vel. (m/s)	Throughput Jakes (Kbps)	Throughput Log Normal (Kbps)	Throughput Nakagami (Kbps)	Throughput Two Ray (Kbps)	Throughput BreakPoint (Kbps)	Throughput SimplePathLoss (Kbps)
4	510.61	588.04	76.83	5.70	2062.75	185.76
8	300.69	63.71	159.83	1.66	485.28	615.43
12	566.81	475.31	553.71	27.05	768.40	65.33
16	230.23	196.85	9.66	1380.45	2166.42	394.95
20	37.50	193.76	682.42	556.88	398.03	158.60

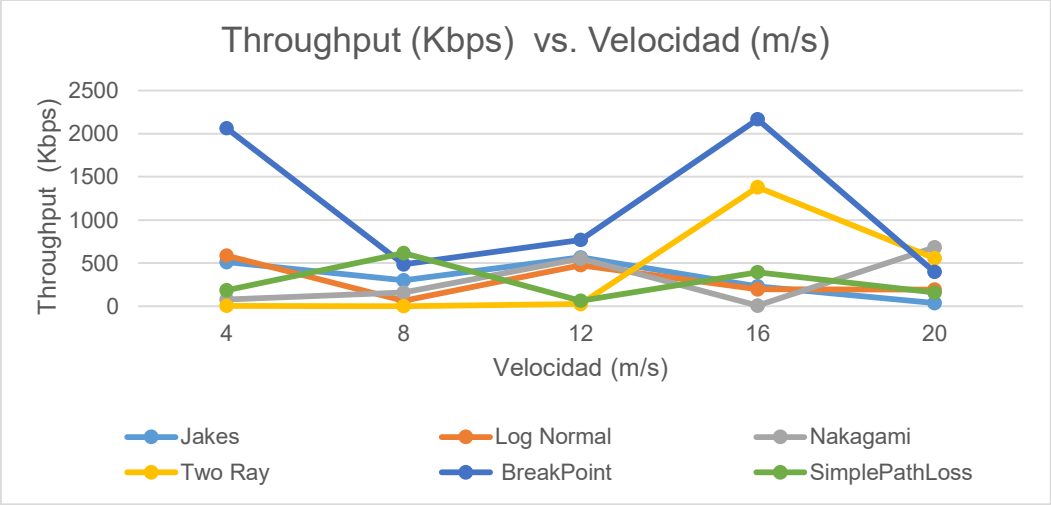

Figura 3.23. Gráfica de Throughput vs. Velocidad de varios modelos de propagación.

SimplePathlossModel carece de una curva semejante a los modelos restantes. Se espera que los resultados de este modelo sean mas erróneos porque solo toma en cuenta la atenuación en espacio libre de una sola señal.

Para todos los modelos se encontraron variaciones agresivas en el throughput cuando excedían 12 m/s. En la Figura 3.24 se muestran Breakpoint Pathloss Model y Two Ray Interference Model, los cuales reflejan curvas similares que se distancian por al menos 500 Kbps de offset; cuando estas curvas alcanzan (o exceden) el valor umbral (12 m/s), el thoughput se dispara a unos 2 Mbps para el caso de Breakpoint y 1.4 Mbps para Two Ray Interference pero decaen rápidamente a 556 y 398 Kbps respectivamente. Two Ray Interference presenta valores tan bajos como 5.7Kbps (7 veces menos que Jakes Fading) o tan altos como 1380 Kbps (7 veces más que Log Normal). Las simulaciones con Breakpoint Pathloss tienen los valores más extravagantes, siendo su valor mas bajo 485 Kbps y el valor mas alto igual a 2166 Kbps.

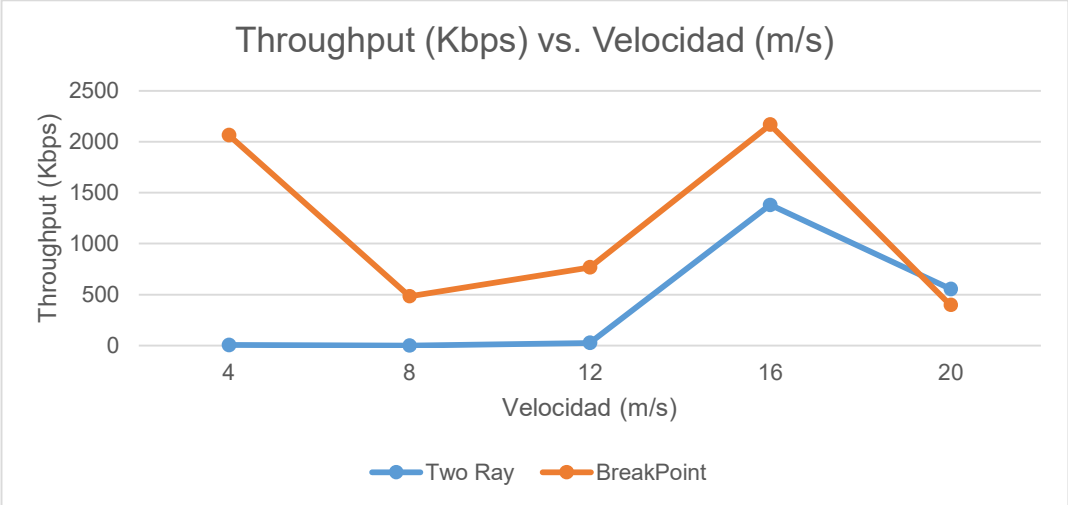

Figura 3.24. Gráfica de Throughput vs. Velocidad de modelos Breakpoint y Two Ray Interference.

Por otro lado, en la Figura 3.25 se pueden apreciar curvas Jakes Fading, Nakagami y Log Normal, las cuales siguen un patrón semejante entre sí. Presentan un máximo relativo en al valor umbral de aproximadamente 500 Kbps. Los resultados de Jakes Fading y LogNormal son consistentes con el concepto de un throughput que se reduce con el aumento de la velocidad de los vehículos. Cuando la velocidad supera los 12 m/s el rendimiento decae rápidamente hasta 37 Kbps (Jakes Fading) y 193 Kbps (Log Normal). El modelo Nakagami tiene un comportamiento afín al de Jakes Fading y Log Normal antes de alcanzar una velocidad de 12 m/s. Superando este umbral, la curva de throughput tiene

un comportamiento polar, es decir, se tienen valores muy bajos o muy altos, lo que hace imposible identificar tendencias; este comportamiento podría ser causado por señales que se benefician de interferencia constructiva modelada en Nakagami.

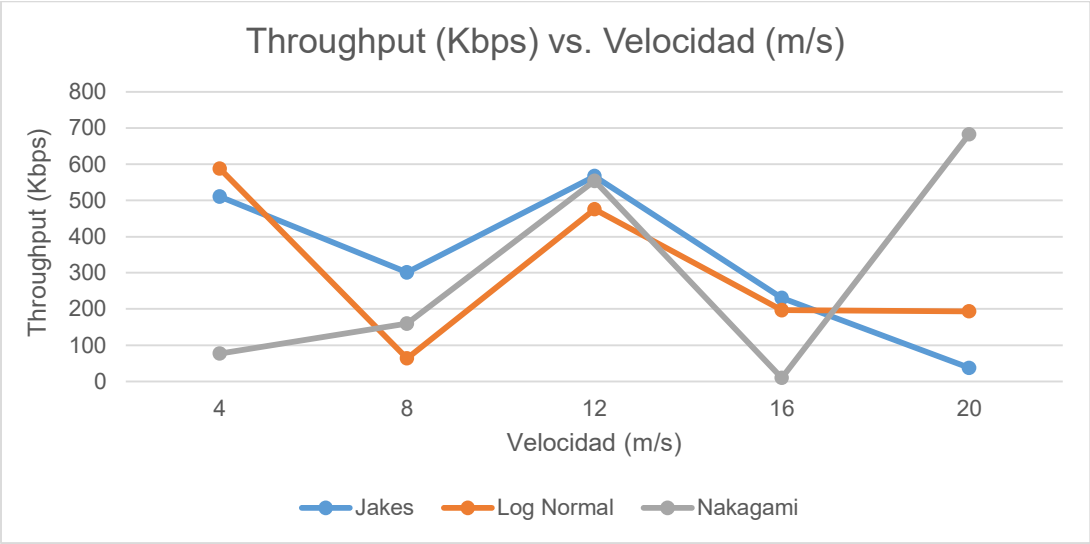

Figura 3.25. Gráfica de Throughput vs. Velocidad de modelos Jakes, Log Normal y Nakagami.

3.4.2 THROUGHPUT VS NÚMERO DE NODOS

En la Tabla 3.8 se resumen los resultados de las simulaciones, y en la Figura 3.26 se muestra una gráfica comparativa del throughput vs el número de nodos de cada modelo de propagación.

Tabla 3.8. Throughput vs. Número de Nodos para distintos modelos de propagación.

Modelo / Nodos	Througput de Jakes (Kbps)	Throughput de Log Normal (Kbps)	Thoughput de Nakagami (Kbps)	Throughput de Two Ray (Kbps)	Throughput de BreakPoint (Kbps)	Throughput de SimplePathLoss (Kbps)
20	22.72	66.75	412.51	25.01	2845.51	612.50
40	229.12	565.18	215.99	736.26	1268.37	725.29
60	441.33	711.17	254.41	1252.23	281.15	181.20
80	933.90	91.17	347.54	579.47	275.99	372.09
100	124.39	483.45	14.22	265.80	655.75	44.03

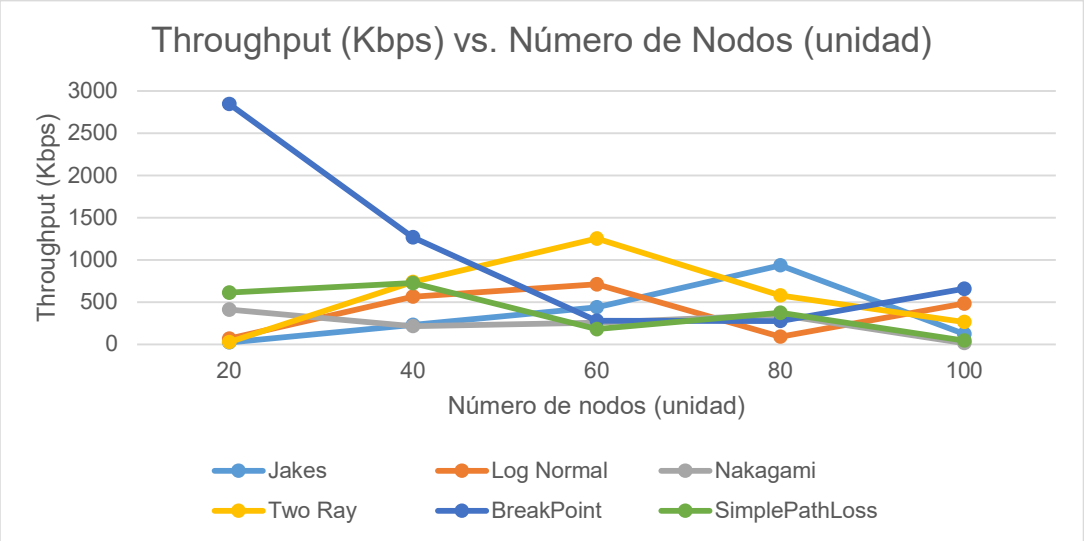

Figura 3.26. Gráfica de Throughput vs. Cantidad de Nodos de varios modelos de propagación.

LogNormal y TwoRayInterference tienen patrones comparables. El throughput crece con la cantidad de nodos debido a que un incremento en la cantidad de vehículos implica un aumento en la cantidad de mensajes que se intercambian con la RSU. Del experimento sabemos que esto aplica únicamente a una cantidad igual o inferior a 60 nodos (máximo relativo). Cuando el número es superior, factores como interferencia y colisiones degradan la comunicación.

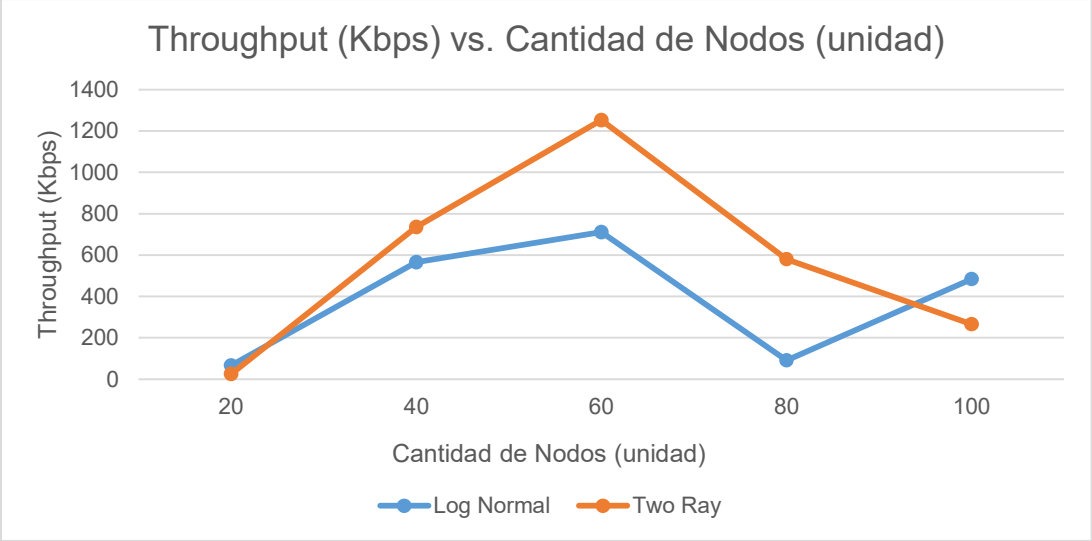

Figura 3.27. Gráfica de Throughput vs. Cantidad de nodos de modelos Log Normal y Two Ray.

En la Figura 3.28 se muestran los modelos Jakes Fading y Nakagami. Se puede apreciar un comportamiento análogo al de Log Normal y TwoRayInterference: que el valor del throughput generalmente asciende con un incremento en el número de vehículos con la diferencia que número de nodos debería ser igual o inferior a 80, el desempeño de la comunicación decae sustancialmente mas allá de este punto. El throughput calculado con Jakes Fading alcanza 933Kbps cuando están activos 80 nodos y decrece hasta 124 Kbps si se tienen 100 vehículos en la red. Para el caso de Nakagami tenemos un pico de 347 Kbps con 80 nodos y 14 Kbps con 100 vehículos.

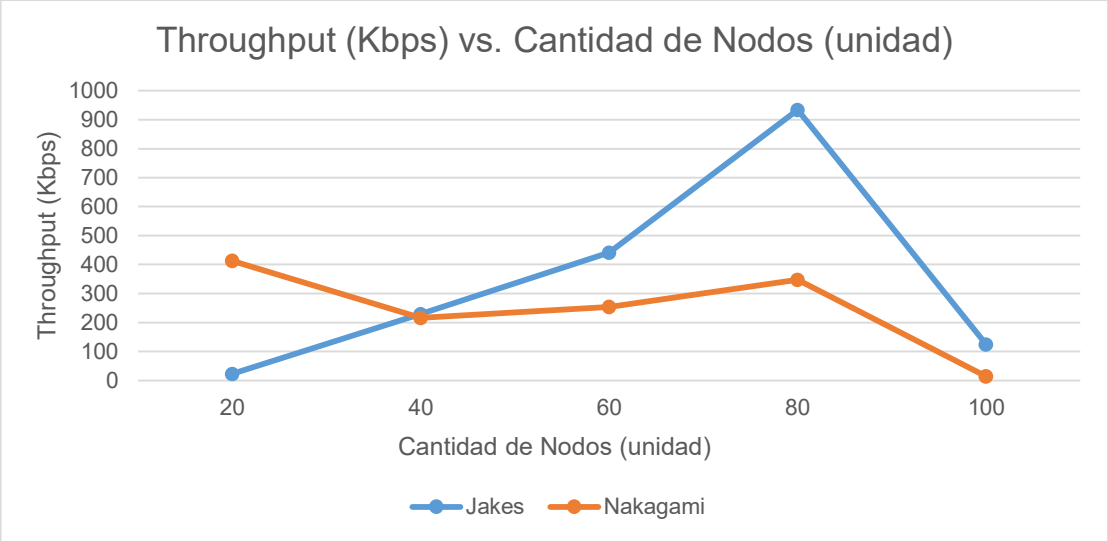

Figura 3.28. Gráfica de Throughput vs. Cantidad de nodos de modelos Jakes y Nakagami.

Con Breakpoint Pathloss se obtuvo un throughput máximo de 2845 Kbps con 20 vehículos (más de dos veces el valor máximo de los modelos restantes), esta cifra decae rápidamente hasta alcanzar 275 Kbps con 80 vehículos y se eleva nuevamente a 655 Kbps con 100 nodos.

3.4.3 RETARDO VS VELOCIDAD DEL VEHÍCULO.

En la Tabla 3.9 se resumen los resultados de las simulaciones. En la Figura 3.29 se muestra una gráfica comparativa para el retardo vs la velocidad de los vehículos para cada modelo de propagación.

Tabla 3.9. Retardo vs. Velocidad de vehículos para distintos modelos de propagación.

Modelo / Velocidad	Retardo de Jakes (us)	Retardo de Log Normal (us)	Retardo de Nakagami (us)	Retardo de Two Ray (us)	Retardo de BreakPoint (us)	Retardo de Simple PathLoss (us)
4	108.92	108.85	108.71	108.22	107.70	108.34
8	109.15	108.90	108.84	108.31	107.96	108.18
12	108.91	108.92	108.89	108.48	107.86	108.18
16	109.01	108.64	108.64	108.19	107.81	108.30
20	108.98	108.79	108.95	108.12	107.85	108.19

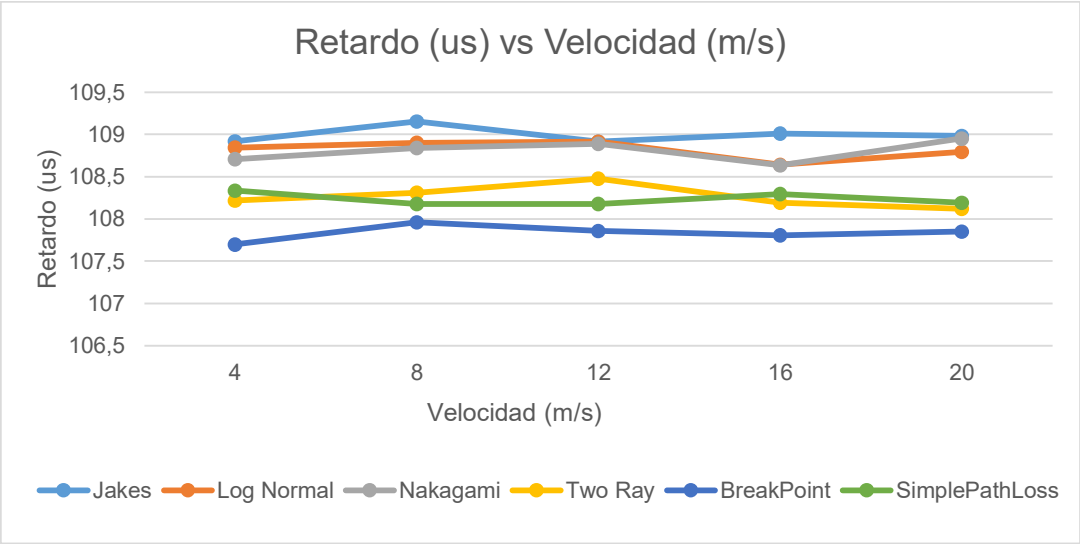

Figura 3.29. Gráfica de Retardo vs. Velocidad para varios modelos.

El retardo se mantiene prácticamente constante sin importar las variaciones de velocidad. Esto se cumple para todos los modelos de propagación. Sin embargo, algunos modelos oscilan en torno a un valor de retardo distinto a los demás. Los modelos Jakes Fading, Log Normal y Nakagami tienen un retardo aproximado de 109 microsegundos. Two Ray Interference oscila en torno a 108 microsegundos y Breakpoint Pathloss tiene un retardo promedio de 107 microsegundos.

3.4.4 RETARDO VS NÚMERO DE NODOS.

En la Tabla 3.10 se resumen los resultados de las simulaciones. En la Figura 3.30 se muestra una gráfica comparativa para el retardo vs la Número de vehículos para cada modelo de propagación.

En este caso, todos los modelos (exceptuando Nakagami) tuvieron resultados consistentes con la idea de que el retardo debería incrementarse con el aumento del número de vehículos.

Two Ray Interference es el único modelo que tiene una tendencia solo creciente para el retardo con el incremento de vehículos en la red. Las simulaciones que emplearon Jakes Fading tienen los retardos mas altos, entre 178.42 a 109.33 microsegundos.

Los modelos Log Normal y Breakpoint Pathloss tienen curvas idénticas que oscilan entre rangos distintos. Log normal oscila entre valores de retardo de 108.67 a 109.01 microsegundos y Breakpoint Pathloss oscila entre 107.51 a 107.87 microsegundos.

Tabla 3.10. Retardo vs. Cantidad de nodos para distintos modelos de propagación.

Modelo / Nodos	Retardo de Jakes (us)	Retardo de Log Normal (us)	Retardo de Nakagami (us)	Retardo de Two Ray (us)	Retardo de BreakPoint (us)	Retardo de SimplePathLoss (us)
20	108.74	108.67	108.83	107.77	107.51	107.86
40	108.91	108.78	108.84	108.25	107.81	108.21
60	109.09	108.66	108.38	108.32	107.82	108.33
80	108.90	108.98	108.99	108.47	108.00	108.40
100	109.33	109.01	108.98	108.50	107.87	108.39

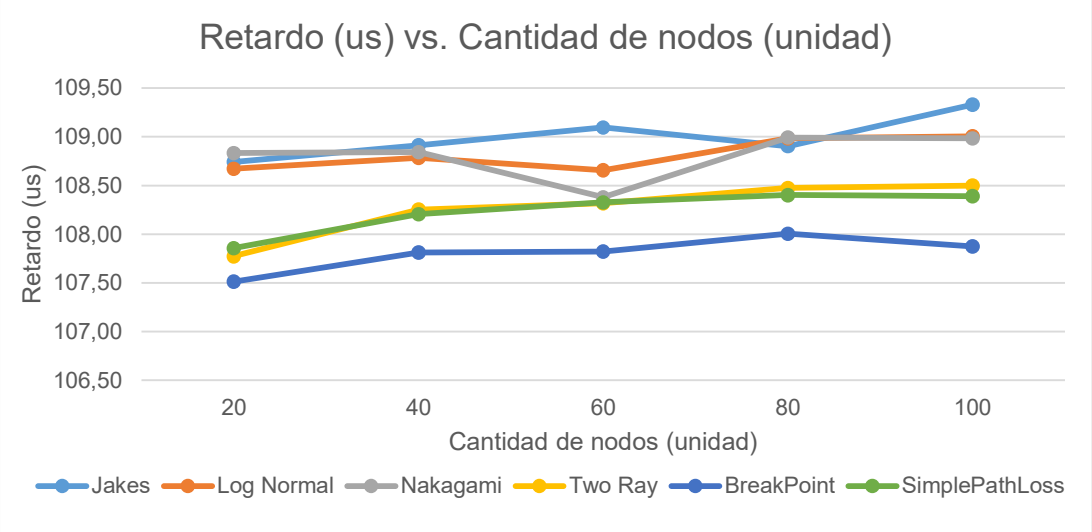

Figura 3.30. Gráfica de Retardo vs. Cantidad de nodos para varios modelos de propagación.

4. CONCLUSIONES Y RECOMENDACIONES

4.1. CONCLUSIONES

OMNeT++ es un simulador de eventos discretos que explota el principio de herencia de programación orientada a objetos. En virtud de esto, investigadores pueden utilizar, modificar o inclusive implementar nuevas funciones en el simulador dependiendo de sus intereses. Este proyecto se benefició de esta cualidad de OMNeT++, haciendo posible la implementación de dos nuevas características: el cálculo del throughput y retardo.

La capa física de INETMANET comprende dos componentes clave: los modelos de radio y los modelos de medio físico. Los modelos de radio describen los procesos realizados en el transmisor y en el receptor para el envío y recepción de mensajes. Por otro lado, los modelos del medio físico permiten realizar una caracterización del canal inalámbrico. Es con estos modelos que se desarrollan varios módulos para simular las características de capa física del estándar IEEE 802.11, dentro del que se incluye el estándar IEEE 802.11a estudiado en este proyecto.

INETMANET implementa la capa física del estándar IEEE 802.11 en OFDM de forma detallada, cumpliendo todas sus especificaciones, por lo que es posible realizar un estudio preciso y realista de los estándares basados en este esquema de modulación, tales como IEEE 802.11a. En la capa física de INETMANET se realizan los procesos de: encapsulación y desencapsulacion, aleatorización y desaleatorizacion, codificación y decodificación, entrelazado y desentrelazado, modulación y demodulación.

La implementación de la capa física IEEE 802.11p de Veins, tiene seis elementos clave: AnalogueModels, módulo Radio, ChannelInfo, Decider, BasePhyLayer y AirFrames. Los AnalogueModels constituyen modelos de propagación que se aplican a las señales recibidas. El módulo Radio emula las propiedades físicas de los transceivers como esquemas de modulación y modos de transmisión simplex/duplex. ChannelInfo identifica todas las AirFrames que están en el medio. Finalmente, Decider puede clasificar las tramas como mensajes o interferencia usando esta información.

Al comparar la implementación de la capa física de Veins con el estándar IEEE 802.11p, se encontró que algunos de los procesos definidos en el estándar no se ejecutan y otros son virtuales. Aleatorización y entrelazado no están presentes y se sustituyen con retrasos. Modulación y demodulación se representan simplemente con indicadores del tipo de modulación en lugar de realizar una modulación real, esto debido a que las AirFrames son abstracciones y no señales reales.

INETMANET posee métodos para el cálculo del throughput a nivel de capa de aplicación, por esta razón fue necesario implementar esta funcion a nivel de capa física. En cuanto al retardo, INETMANET no posee métodos para su cálculo y también fue necesario realizar su implementación. Los métodos donde es posible introducir código para extraer los datos de throughput y retardo son: Radio::startReception(), Radio::continueReception() y Radio::endReception().

Veins es un framework que no posee métodos para el cálculo del throughput y el retardo. Sin embargo, su estructura facilita la expansión de este con nuevas funcionalidades. Motivo por el cual se implementó manualmente el cálculo del throughput, retardo y el almacenamiento de esta información en vectores para que sea posible generar las gráficas de este estudio. Los puntos clave para introducir código con la finalidad de capturar datos en Veins son los métodos: BasePhyLayer::handleAirFrameStartReceive, BasePhyLayer::handleAirFrameReceiving y BasePhyLayer::handleAirFrameEndReceive, estos se ejecutan secuencialmente cada vez que se recibe una trama.

En los resultados que se obtuvieron de las simulaciones realizadas en INETMANET se pudo observar que el retardo no es afectado considerablemente por variaciones en el desplazamiento de los nodos. Sin embargo, existe un incremento sustancial en el retardo cuando el número de nodos en el escenario alcanza 4 unidades, a partir de este punto no se ven cambios de importancia. En el análisis del throughput, se percibe un crecimiento del throughput que coincide con un incremento de la cantidad de nodos. A pesar de ello, el throughput no se ve afectado considerablemente por cambios en la velocidad de desplazamiento. Este comportamiento es común a todos los modelos de propagación que se evaluaron en este estudio.

De los resultados obtenidos en las simulaciones con Veins se puede inferir que el retardo prácticamente no es afectado por cambios en velocidad de los nodos o la cantidad de estos. En el estudio de las variaciones de throughput se encontró que los modelos Jakes, Log Normal y Nakagami responden de manera similar a los cambios en la velocidad de los vehículos, asimismo, Two Ray y Breakpoint producen curvas comparables entre ellas de Throughput vs. Velocidad de vehículos. SimplePathloss no produjo un patrón equiparable, sin embargo, este comportamiento se esperaba debido a que este modelo se caracteriza por considerar sólo pérdidas en espacio libre siendo sus resultados, de menor veracidad. Del análisis de curvas de Throughput vs. Cantidad de nodos en Veins, se pudo observar en todos los modelos una tendencia creciente que se invierte en un punto de inflexión (este punto difiere dependiendo del modelo). La tendencia creciente a la que se hace mención se podría explicar como un aumento natural debido a que se tienen más dispositivos que

pueden recibir datos. Sin embargo, el punto de inflexión marca el umbral donde la interferencia de las señales tiene mayor impacto en la degradación de la comunicación que el incremento de vehículos.

4.2. RECOMENDACIONES

Las simulaciones en OMNeT++ consumen una gran cantidad de recursos computacionales. Para conseguir una mejora considerable en el tiempo de ejecución se recomienda utilizar una distribución de Linux que use pocos recursos como Lubuntu y una PC con 16 GB de RAM y un procesador capaz de alcanzar 2.3GHz.

En caso de requerir una gran cantidad de simulaciones para fines de investigación, es recomendable automatizar la ejecución de estas con scripts de bash. Evitando de ese modo la necesidad de iniciar manualmente las interfaces de SUMO y OMNeT++ y reduciendo el tiempo necesario para ejecutar simulaciones.

Los resultados de las simulaciones se almacenan en archivos de extensión .vec. La información almacenada en estos archivos se puede visualizar en la GUI de OMNeT++ con facilidad. Sin embargo, solo se muestra la información de una simulación a la vez. En caso de requerir un análisis mas sofisticado que permita establecer una comparación entre los resultados de múltiples simulaciones se recomienda el uso de Python y la librería openpyxl. Con estas herramientas es posible presentar los datos en un formato mas convencional como Workbooks de Excel mientras se provee la funcionalidad para analizar estadísticamente los resultados.

5. REFERENCIAS BIBLIOGRÁFICAS

[1] R. Kaur, R. Singla, B. Kaur y S. Singh, «MANETs: Overview, Tools, Security and Applications in Health Care,» *Australian Journal of Basic and Applied Science,* pp. 1-6, 2017.

[2] J. Wu y I. Stojmenovic, «Ad Hoc Networks,» pp. 29-31, 2004.

[3] S. u. Rehman, Vehicular Ad-Hoc Networks (VANETs) - An Overview, Sydney, 2013.

[4] M. Molina y R. Silva, «Arquitectura de las Redes Ad-Hoc,» *Polibits,* vol. 36, pp. 8-13, 2007.

[5] L. Han, «Wireless Ad-hoc Networks,» 2004.

[6] A. Bang y M. Ghorale, «Wireless Ad-Hoc Networks: Types, Applications, Security Goals,» *International Journal of Advent Research in Computer and Electronics (IJARCE),* pp. 128-132, 2015.

[7] T. Ledesma, W. Baluja y L. Coya, «Autoconfiguración en MANETs,» *Revista Telem@tica,* vol. 13, pp. 21-34, 2014.

[8] J. Areito Bertolin, «Análisis de riesgos y contramedidas en Redes MANET,» *REE,* pp. 62-73, 2012.

[9] F. Ruiz y M. Solera, «Simulación de Protocolos de Enrutamiento para Redes Móviles AdHoc Mediante la Herramienta de Simulación NS-3.,» Red Nacional de Investigación y Educación del Ecuador, Loja, 2014.

[10] S. Vijayalakshmi y M. Sweatha, «A Survey on History and Types of Manet,» *International of Emerging Trends in Science and Technology,* vol. 03, nº 07, pp. 4310-4315, 2016.

[11] K. Waseem, K. Faheem, H. Zeeshan, A. Syed, K. Zaidullah, A. Muhammad y A. Muhammad, «An Overview of Mobile Ad-hoc Network Simulators and Associated Simulation Techniques,» *International Journal od Computer Science and Telecommunications,* vol. 3, nº 6, pp. 39-43, 2012.

[12] S. Al-Sultan, M. M. Al-Doori, A. H. Al-Bayatti y H. Zedan, A comprehensive survey on vehicular Ad Hoc network, Leicester: Elsevier, 2013.

[13] J. Hoebeke, I. Moerman, B. Dhoedt y P. Demeester, «An Overview of Mobile Ad Hoc Networks: Applications and Challenges,» pp. 60-66, 2004.

[14] M. Kumar, «An Overview of MANET: History, Challenges and Applications,» *Indian Journal of Computer Scienceand Engineering ,* p. 5, 2012.

[15] M. Ramia , M. A. Amin y O. A. Ashraf, «A Comparison between IEEE 802.11a, b, g, n and ac Standards,» *IOSR Journal of Computer Engineering,* vol. 17, nº : 2278-0661, p. 4, 2015.

[16] Tektronix, Wi-Fi: Overview of the 802.11 Physical Layer and Transmitter Measurements, 2013.

[17] H. Yomo, C. Huan, P. Kyritsi, T. Duc, S. Chakraborty y R. Prasad, «PHY and MAC Performance Evaluation of IEEE 802.11a WLAN over Fading Channels,» *IETE Journal of Research,* vol. 51, pp. 1-10, 2015.

[18] E. Herrera, A. Diaz y C. Calafate, «Desarrollando el estándar IEEE 802.11n, un paso adelante en WLAN,» *CiComp07,* pp. 1-9, 2007.

[19] Cisco Systems, Inc., Voice over Wireless LAN 4.1 Design Guide, San Jose, USA: Cisco, 2010.

[20] Motorola, 5Ghz IEEE 802.11a for Interference Avoidance, 2009.

[21] T. S. Rappaport, Wireless Communications: Principles and Practice, Michigan: Prentice Hall, 2012.

[22] O. Orozco y G. Llano, VANET Applications focused on environmental sustainability, Cali: Imprenta de la Universidad Icesi, 2014.

[23] A. . M. Abeldime y L. Wu, The Physical Layer of the IEEE 802.11p WAVE Communication Standard: The Specifications and Challenges, San Francisco, 2014.

[24] M. Emmelman, . . Bochow y C. Kellum, Vehicular Networking: Automotive Applications and Beyond., Chichester: Wiley, 2010.

[25] P. Kukolev, «Comparison of 802.11a anda 802.11p over Fading Channels,» *elektrorevue,* vol. 4, nº 1, pp. 7-11, 2013.

[26] IEEE, IEEE Standard for Information technology--Telecommunications and information exchange between systems Local and metropolitan area networks--Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE, 2016.

[27] V. Garg, Wireless Communications and Networking, San Francisco: Elsevier, 2007.

[28] M. Gast, 802.11 Wireless Networks, O'Reilly, 2005.

[29] S. Grafling, P. Mahonen y J. Riihijarvi, «Performance evaluation of IEEE 1609 WAVE and IEEE 802.11 p for vehicular communications. in Ubiquitous and Future Networks (ICUFN),» *IEEE Second International Conference on 2010 ,* 2010.

[30] J. Mittag, Enabling accurate cross-layer PHY/MAC/NET simulation studies of vehicular communication networks, Proceedings of the IEEE, 2011.

[31] L. W. Couch, Digital and Analog Commnication Systems, New Jersey: Pearson, 2013.

[32] B. Carlson y P. B. Crilly, Communication Systems: An Introduction to Signals and noise in Electrical Communication, New York: McGrawHill, 2013.

[33] . X. Shao-Qiu, Z. Ming-Tuo y Z. Yan, Millimeter Wave Technology in Wireless PAN, LAN and MAN, Washington: Taylor & Francis, 2008.

[34] F. Ros, J. Martinez y P. Ruiz, «A survey on modeling and simulation of vehicular networks: Communications, mobility, and tools,» *Computer Communications,* vol. 43, pp. 1-15, 2014.

[35] L. Urquiza, C. Tripp, I. Matrin y M. Aguilar, «Propagation and Packet Error Models in VANET Simulations,» *IEEE Latin American Transactions,* vol. 12, pp. 499-507, 2014.

[36] D. Tse, Fundamentals of Wireless Communication, Cambridge: Cambridge University Press, 2012.

[37] P. Dent, G. Bottomley y T. Croft, «Jakes Fading Model Revisted,» *Electronics Letters,* vol. 29, pp. 1162-1163.

[38] OpenSim Ltd., «OMNeT++,» 2019. [En línea]. Available: https://omnetpp.org/.

[39] OpenSim Ltd., OMNeT++ Simulation Manual, 2016.

[40] F. Moreno, Introducción a la OOP, Grupo EIDOS.

[41] J. Finochietto, «Programación Orientada a Objetos en C++,» pp. 1-68, 2010.

[42] R. L. Fonseca, «Introducción a la Programación Orientada a Objetos,» 2014.

[43] J. L. Castillo, «Programación Orientada a Objeto,» pp. 1-65.

[44] R. Nagel y S. Eichler, «Efficient and Realistic Mobility and Channel Modeling for,» Technische Universitat Munchen, Munich, 2014.

[45] C. Lochert, A. Barthels, A. Cervantes y M. Mauve, «Multiple simulator interlinking environment for ivc.,» *ACM International Workshop,* pp. 87-88, 2010.

[46] P. v. Wijngaarden, FRAME CAPTURE IN 802.11P VEHICULAR NETWORKS., Enschede, 2011.

[47] K. Wessel, M. Swigulski, A. Kopke y D. Wilkomm, «Mixim - The Physical Layer An Architecture Overview,» de *International Workshop onOMNeT++*, Rome, 2009.

[48] R. Tomar, G. Deep Singh y M. Prateek, «Modelling and Simulation of Vehicle to Vehicle Communication,» *International Journal of Control Theory and Applications,* vol. 10, nº 0974-5572, pp. 245-253, 2017.

[49] INET Framework, INET Framework for OMNeT++ Manual, 2016.

[50] INET Framework, «API INET Framework,» [En línea]. Available: https://doc.omnetpp.org/inet/api-current/neddoc/index.html.

[51] INET Framework, «INET Framework User's Guide,» [En línea]. Available: https://inet.omnetpp.org/docs/users-guide/.

[52] A. Kopke , K. Haneveld y H. Lichte, «Simulating Wireless and Mobile Networks in OMNeT++ The MiXim Vision.,» de *International Workshop on OMNeT++*, 2008.

[53] J. Treviño, «Propagación de RF en las bandas: LF, MF, HF, VHF, UHF y VHF,» Universidad de las Americas Puebla, Mexico, 2003.

ANEXOS

ANEXO A. Instalacion de OMNeT++.

ANEXO B. Instalacion de INET.

ANEXO C. Instalacion de INETMANET.

ANEXO D. Instalacion de Veins y SUMO.

ANEXO E. Configuracion del transmisor y del receptor en INETMANET.

ANEXO F. Script de automatización de Simulaciones con Veins.

ANEXO G. Script de extracción de datos de archivos .vec.

ANEXO H. Script de procesamiento de datos de archivos .vec.

ANEXO A

Instalacion de OMNeT++

Las simulaciones de OMNeT++ se van a realizar en una máquina virtual con Linux. Las máquinas virtuales tienen una eficiencia/velocidad menor comparadas con sus contrapartes de instalación nativa. Las simulaciones con OMNeT++ pueden ser exigentes en términos de recursos del sistema; las limitaciones de rendimiento de las máquinas virtuales pueden entorpecer nuestro trabajo y por eso es de suma importancia elegir una distribución que garantice compatibilidad y la máxima velocidad posible en el entorno de virtualización.

Ubuntu, Fedora 25, Red Hat y OpenSUSE son distribuciones compatibles con OMNeT++ 5.0.

Elegimos Ubuntu porque tiene el mayor soporte, pero ejecutarlo es muy costoso en términos de procesamiento; la solución a este problema: Lubuntu, la versión ligera de Ubuntu.

Lubuntu se diferencia de su hermano mayor por un consumo mucho más bajo de recursos que mantiene la misma funcionalidad. Lubuntu prescinde del entorno de escritorio Unity y utiliza LXDE. Unity tiene una apariencia mucho más agradable que LXDE, pero eso viene con un costo: requerimientos del sistema más altos. LXDE limita el uso de animaciones y controles con 3D. En la tabla a continuación, se comparan los requerimientos mínimos de Ubuntu con los de Lubuntu.

	Ubuntu	**Lubuntu**
CPU	2 GHz de doble núcleo	Pentium 4 (1 núcleo)
RAM	2 GB	512 MB
HDD	25 GB	5 GB

La intalacion se la realiza a traves del terminal, por medio de comandos. A continuacion se describe paso a paso los comandos para la intalacion:

1.-Instalación de Prerrequisitos de OMNET++

Primero se requiere actualizar la base de datos de los paquetes disponibles:

```
$ sudo apt-get update
```

Luego instalar todos los paquetes requeridos

```
$ sudo apt-get install build-essential gcc g++ bison flex perl \
qt5-default tcl-dev tk-dev libxml2-dev zlib1g-dev default-jre \
doxygen graphviz libwebkitgtk-3.0-0
```

Para agregar soporte 3D con Qtenv, se necesitan los paquetes de desarrollo de OpenSceneGraph(3.2) y osgEarth (2.5 o superior).

```
$ sudo apt-get install libopenscenegraph-dev \
openscenegraph-plugin-osgearth libosgearth-dev
```

Se instalan los paquetes MPI para permitir soporte de simulaciones paralelas

```
$ sudo apt-get install openmpi-bin libopenmpi-dev
```

2.-Descargar y desempacar OMNeT++

Descargar OMNeT++ 5.0 de http://omnetpp.org . El archivo tiene una extensión .tgz. Luego copiar el archivo al directorio donde se desea instalar (normalmente /home/<usuario>).En la terminal, usar el siguiente comando para extraer el archivo .tgz

```
$ tar xvfz omnetpp-5.1.1-src.tgz
```

3.-Establecer el las variables de entorno de forma permanente (Agregar al PATH)

Es recomendable que las variables se establezcan de forma permanente. Editar el el archivo .bashrc.

```
$ gedit ~/.bashrc
```

Agregar esta línea al final del archivo.

```
export PATH=$HOME/omnetpp-5.1.1/bin:$PATH
```

4.-Configurar y Construir OMNeT++

En la raíz del directorio, ejecutar

```
$ ./configure
```

Cuando este proceso haya terminado, escribir en la terminal:

```
$ make
```

ANEXO B.

Instalación de INET

Para la instalacion se descarga el código fuente y luego se descomprime el directorio:

```
tar xvfz inet-<version>.tgz
```

Para la instalación se requiere que el IDE de OMNeT++ haya sido iniciado en donde iremos

```
File -> Import -> Existing Projects to the Workspace
```

Importamos el Proyecto de INET y vamos a

```
Build with Project -> Build
```

ANEXO C

Instalación de INETMANET

Para la instalacion se descarga el código fuente y luego se descomprime el directorio:

```
tar xvfz inetmanet-<version>.tgz
```

Para la instalación se requiere que el IDE de OMNeT++ haya sido iniciado en donde iremos

```
File -> Import -> Existing Projects to the Workspace
```

Importamos el Proyecto de INETMANET y vamos a

```
Build with Project -> Build
```

ANEXO D

Instalación de Veins y SUMO.

1.-Instalación de SUMO

Agregue los repositorios de SUMO, actualice la base de datos y solicite los paquetes usando estos comandos:

```
sudo add-apt-repository ppa:sumo/stable
sudo apt-get update
sudo apt-get install sumo sumo-tools sumo-doc
```

Es recomendable que las variables se establezcan de forma permanente. Editar el archivo .bashrc.

```
$ gedit ~/.bashrc
```

Agregar esta línea al final del archivo.

```
export SUMO_HOME=/usr/share/sumo/
```

Con el comando `sumo --version` se puede comprobar la versión de SUMO.

2.-Instalación de Veins

Para la instalacion se descarga el código fuente y luego se descomprime el directorio:

```
tar xvfz veins-<version>.tgz
```

Para la instalación se requiere que el IDE de OMNeT++ haya sido iniciado en donde iremos

```
File -> Import -> Existing Projects to the Workspace
```

Importamos el Proyecto de veins y vamos a

```
Build with Project -> Build
```

ANEXO E

Configuración del transmisor y del receptor en INETMANET.

```
## Transmisor
**.isCompliant = false
**.wlan[*].radio.transmitterType = "Ieee80211LayeredOFDMTransmitter"
**.wlan[*].radio.transmitter.channelSpacing = 20MHz
**.wlan[*].radio.transmitter.signalEncoderType = "Ieee80211OFDMEncoder"
**.wlan[*].radio.transmitter.signalModulatorType =
"Ieee80211OFDMModulator"
**.wlan[*].radio.transmitter.dataEncoderType = "Ieee80211OFDMEncoder"
**.wlan[*].radio.transmitter.dataModulatorType = "Ieee80211OFDMModulator"
**.wlan[*].radio.transmitter.dataEncoder.fecType = "ConvolutionalCoder"
**.wlan[*].radio.transmitter.dataEncoder.scramblerType =
"AdditiveScrambler"
**.wlan[*].radio.transmitter.dataEncoder.interleaverType =
"Ieee80211OFDMInterleaver"
**.wlan[*].radio.transmitter.signalEncoder.fecType = "ConvolutionalCoder"
**.wlan[*].radio.transmitter.signalEncoder.scramblerType = ""
**.wlan[*].radio.transmitter.signalEncoder.interleaverType =
"Ieee80211OFDMInterleaver"

**.wlan[*].radio.transmitter.dataModulator.pilotSubcarrierPolarityVectorO
ffset = 0
**.wlan[*].radio.transmitter.signalModulator.pilotSubcarrierPolarityVecto
rOffset = 0

# Aleatorizacion (Scrambler)
**.wlan[*].radio.**.seed = "1011101"
**.wlan[*].radio.**.generatorPolynomial = "0001001"

# Entrelazado de datos (Data interleaver)
**.wlan[*].radio.transmitter.dataEncoder.interleaver.numberOfCodedBitsPer
Subcarrier = 4
**.wlan[*].radio.transmitter.dataEncoder.interleaver.numberOfCodedBitsPer
Symbol = 192

# Entrelazado de señal (Signal interleaver)
**.wlan[*].radio.transmitter.signalEncoder.interleaver.numberOfCodedBitsP
erSubcarrier = 1
**.wlan[*].radio.transmitter.signalEncoder.interleaver.numberOfCodedBitsP
erSymbol = 48

# Codificacion de datos FEC
**.wlan[*].radio.transmitter.dataEncoder.fecEncoder.transferFunctionMatri
x = "133 171"
**.wlan[*].radio.transmitter.dataEncoder.fecEncoder.constraintLengthVecto
r = "7"
**.wlan[*].radio.transmitter.dataEncoder.fecEncoder.puncturingMatrix = "1
1 0; 1 0 1"
**.wlan[*].radio.transmitter.dataEncoder.fecEncoder.punctureK = 3
**.wlan[*].radio.transmitter.dataEncoder.fecEncoder.punctureN = 4

# Codificación de señal FEC
**.wlan[*].radio.transmitter.signalEncoder.fecEncoder.transferFunctionMat
rix = "133 171"
**.wlan[*].radio.transmitter.signalEncoder.fecEncoder.constraintLengthVec
tor = "7"
```

```
**.wlan[*].radio.transmitter.signalEncoder.fecEncoder.puncturingMatrix =
"1; 1"
**.wlan[*].radio.transmitter.signalEncoder.fecEncoder.punctureK = 1
**.wlan[*].radio.transmitter.signalEncoder.fecEncoder.punctureN = 2

# Modulacion
**.wlan[*].radio.transmitter.signalModulator.subcarrierModulation =
"BPSK"
**.wlan[*].radio.transmitter.signalModulator.pilotSubcarrierPolarityVecto
rOffset = 0
**.wlan[*].radio.transmitter.dataModulator.subcarrierModulation = "QAM-
16"
**.wlan[*].radio.transmitter.dataModulator.pilotSubcarrierPolarityVectorO
ffset = 1


## Receiver
**.wlan[*].radio.receiverType = "Ieee80211LayeredOFDMReceiver"
**.wlan[*].radio.receiver.dataDecoderType = "Ieee80211OFDMDecoder"
**.wlan[*].radio.receiver.errorModelType = "Ieee80211OFDMErrorModel"
**.wlan[*].radio.receiver.signalDecoderType = "Ieee80211OFDMDecoder"
**.wlan[*].radio.receiver.signalDemodulatorType =
"Ieee80211OFDMDemodulator"
**.wlan[*].radio.receiver.dataDemodulatorType =
"Ieee80211OFDMDemodulator"
**.wlan[*].radio.receiver.dataDecoder.fecType = "ConvolutionalCoder"
**.wlan[*].radio.receiver.dataDecoder.descramblerType =
"AdditiveScrambler"
**.wlan[*].radio.receiver.dataDecoder.deinterleaverType =
"Ieee80211OFDMInterleaver"
**.wlan[*].radio.receiver.signalDecoder.fecType = "ConvolutionalCoder"
**.wlan[*].radio.receiver.signalDecoder.descramblerType = ""
**.wlan[*].radio.receiver.signalDecoder.deinterleaverType =
"Ieee80211OFDMInterleaver"

# Data deinterleaver
**.wlan[*].radio.receiver.dataDecoder.deinterleaver.numberOfCodedBitsPerS
ubcarrier = 4
**.wlan[*].radio.receiver.dataDecoder.deinterleaver.numberOfCodedBitsPerS
ymbol = 192

# Signal deinterleaver
**.wlan[*].radio.receiver.signalDecoder.deinterleaver.numberOfCodedBitsPe
rSubcarrier = 1
**.wlan[*].radio.receiver.signalDecoder.deinterleaver.numberOfCodedBitsPe
rSymbol = 48

# Data FEC
**.wlan[*].radio.receiver.dataDecoder.fecDecoder.transferFunctionMatrix =
"133 171"
**.wlan[*].radio.receiver.dataDecoder.fecDecoder.constraintLengthVector =
"7"
**.wlan[*].radio.receiver.dataDecoder.fecDecoder.puncturingMatrix = "1 1
0; 1 0 1"
**.wlan[*].radio.receiver.dataDecoder.fecDecoder.punctureK = 3
**.wlan[*].radio.receiver.dataDecoder.fecDecoder.punctureN = 4

# Signal FEC
**.wlan[*].radio.receiver.signalDecoder.fecDecoder.transferFunctionMatrix
= "133 171"
```

```
**.wlan[*].radio.receiver.signalDecoder.fecDecoder.constraintLengthVector
= "7"
**.wlan[*].radio.receiver.signalDecoder.fecDecoder.puncturingMatrix = "1;
1"
**.wlan[*].radio.receiver.signalDecoder.fecDecoder.punctureK = 1
**.wlan[*].radio.receiver.signalDecoder.fecDecoder.punctureN = 2

# Modulation
**.wlan[*].radio.receiver.signalDemodulator.subcarrierModulation = "BPSK"
**.wlan[*].radio.receiver.dataDemodulator.subcarrierModulation = "QAM-16"
```

ANEXO F

Script de automatización de simulaciones con Veins.

```
#!/bin/bash

# sintaxis probando <Numero de Nodos Maximo> <Velocidad Maxima>
#El script genera simulaciones para pasos de nodos de uno y de velocidad
de uno en uno.
# POr ejemplo: COn un numero de nodos maximo de 5 y una velocidad MAx de
50 habran simulaciones para
#1,2,3,4, y 5 nodos, y para cada uno, velocidades de
1km/h,2km/h,3km/h,4km/h,.....50km/h

contadorN=50
contadorK=1


while [ $contadorN -le $1 ]; do
    while [ $contadorK -le $2 ]; do
        cd
            sed -i "s/speed=\"[0-9]*/speed=\"$contadorK/"
/home/tesis/omnetpp-5.0/samples/veins/examples/veins/type.add.xml

        #Creacion de trips, con la velocidad especificada en el archivo
myType
        numero=$RANDOM
        echo  SUMO usara la semilla $numero
        echo  $contadorN vehiculos ...
        #Para poder generar el numero necesario de vehiculos, esta en el
comado de abajo se usa el
        coeficiente=$(( 100/contadorN ))

        /usr/share/sumo/tools/randomTrips.py -n /home/tesis/omnetpp-
5.0/samples/veins/examples/veins/erlangen.net.xml --trip-
attributes="type=\"myType\"" --additional-file /home/tesis/omnetpp-
5.0/samples/veins/examples/veins/type.add.xml -l -e 100 -p $coeficiente -
-seed $numero -o trips.trips.xml --validate

        #modificar script para que todos empiecen en t=0s
        sed -i 's/depart="[0-9]*/depart="0/' trips.trips.xml

        #Convertir trips a rutasss
        /usr/bin/duarouter -n /home/tesis/omnetpp-
5.0/samples/veins/examples/veins/erlangen.net.xml --route-files
trips.trips.xml -o /home/tesis/omnetpp-
5.0/samples/veins/examples/veins/erlangen.rou.xml --ignore-errors

        #Modificar el archivo de rutas para que deje de buscar en la web
        sed -i 's+<routes xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/routes_file.xsd">+<
routes>+' /home/tesis/omnetpp-
5.0/samples/veins/examples/veins/erlangen.rou.xml

        #matar procesos residuales de python
        pkill python

        #iniciar sumo
        python omnetpp-5.0/samples/veins/sumo-launchd.py -vv &
```

```
        #Correr SUMO en consola
        cd omnetpp-5.0/samples/veins/examples/veins
        #sumo-gui -c erlangen.sumo.cfg
        sumo -c erlangen.sumo.cfg

        #Cambia de directorio y Corre la simulacionen Omnet
        cd /home/tesis/omnetpp-5.0/samples/veins/examples/veins
        opp_run -r 0 -u Cmdenv -n .:../../src/veins --image-
path=../../images -l ../../src/veins --debug-on-errors=false omnetpp.ini
        #opp_run -u Cmdenv -n .:../../src/veins --image-path=../../images
-l ../../src/veins --debug-on-errors=false omnetpp.ini
        #opp_run -r 0 -n .:../../src/veins --image-path=../../images -
l ../../src/veins --debug-on-errors=false omnetpp.ini

        nomResultado=$contadorN
        nomResultado+="-"
        nomResultado+=$contadorK
        nomResultado+=".vec"

        cp /home/tesis/omnetpp-
5.0/samples/veins/examples/veins/results/General-0.vec
/home/tesis/Desktop/Auto-Results/$nomResultado

        contadorK=$((contadorK+1))
        #echo $contadorK
    done
    contadorK=1
    contadorN=$((contadorN+10))
    echo $contadorN
done
```

ANEXO G

Script de extracción de datos de archivos .vec.

```
# !/usr/bin/python

import os, sys ,glob, stat
import subprocess
from os import listdir
from os.path import isfile, join

from openpyxl import Workbook
from openpyxl.compat import range
from openpyxl.utils import get_column_letter


listaNodos=[]
seguir=False
path="/home/tesis/Desktop/Auto-Results"

def obtenerVectores(path):

    onlyfiles = [f for f in listdir(path) if isfile(join(path, f))]
    for f in onlyfiles:
        print(f)
        if len(f.split("."))==3:


crearDir="home/tesis/Desktop/Procesado"+"/"+(f.split("."))[0]+"."+(f.spli
t("."))[1]+"-p/"

crearDir2="Desktop/Procesado"+"/"+(f.split("."))[0]+"."+(f.split("."))[1]
+"-p/"
            extender=True
            #print(f.split("."))
        if len(f.split("."))==2:


crearDir="home/tesis/Desktop/Procesado"+"/"+(f.split("."))[0]+"-p/"
            crearDir2="Desktop/Procesado"+"/"+(f.split("."))[0]+"-p/"
            extender=False
            #print(crearDir2)
        #print(f)
        #print((f.split("."))[0])

        if not os.path.exists(crearDir2):
            os.mkdir(crearDir2,0o777)
            os.chmod(crearDir2,stat.S_IRWXO)
        vecEncontrado=False
        nombre=f
        pathArchivo=path+"/"+nombre

        #print pathArchivo
        # file-input.py
        lineList = open(pathArchivo,'r').readlines()

        for line in lineList:
            if not "rsu" in line:
                if "throughput" in line:
                    #print(line)
```

```
                    vecDividido=line.split()
                    if len(vecDividido) >= 2:
                        numVector=vecDividido[1]
                    if len(vecDividido) >= 3:
                        nodoDividido=vecDividido[2]
                        numNodo=nodoDividido.split(".")
                        if len(numNodo) >= 2:
                            nombreNodo=numNodo[1]
                            #print(numNodo)

listaNodos.append((numVector,nombreNodo,"throughput"))
                    if "retardo" in line:
                    #print(line)
                    vecDividido=line.split()
                    if len(vecDividido) >= 2:
                        numVector=vecDividido[1]
                    if len(vecDividido) >= 3:
                        nodoDividido=vecDividido[2]
                        numNodo=nodoDividido.split(".")
                        if len(numNodo) >= 2:
                            nombreNodo=numNodo[1]
                            #print(numNodo)
                    listaNodos.append((numVector,nombreNodo,"retardo"))

        for tupla in listaNodos:
            #print(tupla)
            numeroVector= tupla[0]
            nomNodo=tupla[1]
            tipo=tupla[2]
            dirArchivo=crearDir+nomNodo+".xlsx"

            #print(dirArchivo)
            #archivoTexto=(dirArchivo,w+)
            dest_filename = dirArchivo

            #Un workbook para el throughput
            wbThroughput = Workbook()
            wsThro = wbThroughput.worksheets[0]
            wsThro.title = "Throughput"
            wsThro.cell(column=1, row=1, value=nomNodo)
            wsThro.cell(column=1, row=2, value="Tiempo")
            wsThro.cell(column=2, row=2, value="Throughput (bits)")

            #Un workbook para el Retardo
            wbRetardo=Workbook()
            wsRet=wbRetardo.worksheets[0]
            wsRet.title="Retardo"
            wsRet.cell(column=1, row=1, value=nomNodo)
            wsRet.cell(column=1, row=2, value="Tiempo")
            wsRet.cell(column=2, row=2, value="Retardo (s)")

            #hay que llenar los datos desde la fila 3
            filaNum=3


            #EMpieza la iteracion sobre los archivos
            dirPython=dest_filename.split("/")

nuevoDir=dirPython[2]+"/"+dirPython[3]+"/"+dirPython[4]+"/"+dirPython[5]
            #print(nuevoDir)
```

```
            if tipo=="throughput":
                for line in lineList:
                    lineaDividida= line.split()
                    if len(lineaDividida)>=1:
                        if lineaDividida[0]==numeroVector:
                            wsThro.cell(column=1, row=filaNum,
value=lineaDividida[2])
                            wsThro.cell(column=2, row=filaNum,
value=lineaDividida[3])
                            filaNum=filaNum+1

                if not extender:
                    nuevoDir2=(nuevoDir.split("."))[0]+"-
Throughput.xlsx"
                if extender:

nuevoDir2=(nuevoDir.split("."))[0]+"."+(nuevoDir.split("."))[1]+"-
Throughput.xlsx"

                wbThroughput.save(nuevoDir2)
                            #print("1")
                            #print(ws1.cell(column=2, row=20).value)


            if tipo=="retardo":
                for line in lineList:
                    lineaDividida= line.split()
                    if len(lineaDividida)>=1:
                        if lineaDividida[0]==numeroVector:

                            wsRet.cell(column=1, row=filaNum,
value=lineaDividida[2])
                            wsRet.cell(column=2, row=filaNum,
value=lineaDividida[3])

                            filaNum=filaNum+1

                #print(nuevoDir)
                if not extender:
                    nuevoDir2=(nuevoDir.split("."))[0]+"- Retardo.xlsx"
                if extender:

nuevoDir2=(nuevoDir.split("."))[0]+"."+(nuevoDir.split("."))[1]+"-
Retardo.xlsx"

                #print(nuevoDir2)
                wbRetardo.save(nuevoDir2)

            #subprocess.call("./cambiarPermisos.sh")
            #wb.save('node[0].xlsx')
            #wb.save('Desktop/Procesado/10-11-
p/node[0].xlsx')#############
            #wb.save(nuevoDir)
            #print("2")
            #print(ws1.cell(column=2, row=20).value)

obtenerVectores(path)
```

ANEXO H

Script de procesamiento de datos de archivos .vec.

```
# !/usr/bin/python

import os, sys ,glob, stat
import subprocess
from os import listdir
from os.path import isfile, join

from openpyxl import load_workbook
from openpyxl import Workbook
from openpyxl.compat import range
from openpyxl.utils import get_column_letter

pathOrigen="/home/tesis/Desktop/Procesado"
pathResultado="/home/tesis/Desktop/Resultado/resultado.xlsx"
pathThVe="/home/tesis/Desktop/Resultado/Throughput vs Velocidad.xlsx"
pathReVe="/home/tesis/Desktop/Resultado/Retardo vs Velocidad.xlsx"

pathThNo="/home/tesis/Desktop/Resultado/Throughput vs Nodos.xlsx"
pathReNo="/home/tesis/Desktop/Resultado/Retardo vs Nodos.xlsx"

wb=Workbook()
wb.title="Resultado"
sheet=wb.active
sheet.cell(column=1, row=2, value="Numero de Nodos")
sheet.cell(column=2, row=2, value="Velocidad (m/s)")
sheet.cell(column=3, row=2, value="Throughput")
sheet.cell(column=4, row=2, value="Retardo")


carpetas = [nombre for nombre in listdir(pathOrigen) if
os.path.isdir(os.path.join(pathOrigen, nombre))]
for nombre in carpetas:
    nuevoPath=pathOrigen+"/"+nombre
    numeroNodos=(nombre.split("-"))[0]
    velocidad=(nombre.split("-"))[1]
    sumaThrough=0
    contThrough=0
    sumaRetardo=0
    contRetardo=0
    onlyfiles = [f for f in listdir(nuevoPath) if isfile(join(nuevoPath,
f))]
    for f in onlyfiles:
        nombre= f
        pathNombre=join(nuevoPath, f)
        print(pathNombre)
        wbAnalizar=load_workbook(pathNombre)
        first_sheet=wbAnalizar.get_sheet_names()[0]
        wsAnalizar=wbAnalizar.get_sheet_by_name(first_sheet)
        if "Throughput" in nombre:
            for fila in range(3,wsAnalizar.max_row+1):

sumaThrough=sumaThrough+float(wsAnalizar.cell(column=2,row=fila).value)
                contThrough=contThrough+1
        if "Retardo" in nombre:
            for fila in range(3,wsAnalizar.max_row+1):
```

```
sumaRetardo=sumaRetardo+float(wsAnalizar.cell(column=2,row=fila).value)
                contRetardo=contRetardo+1

    promThrough=sumaThrough/contThrough
    promRetardo=sumaRetardo/contRetardo
    nuevaFila=(numeroNodos,velocidad,promThrough,promRetardo)
    sheet.append(nuevaFila)


#-------------------------------------Throughput vs velocidad------------
-----------------------------------------------
valPromTh=0
thCont=0
wb.save(pathResultado)

wbThVe=Workbook()
shThVe=wbThVe.active
shThVe.cell(column=1,row=1,value="Throughput vs Velocidad")
shThVe.cell(column=1,row=2,value="Velocidad Vehiculo")
shThVe.cell(column=2,row=2,value="Throughput")

wbLeer=load_workbook(pathResultado)
sheetLeer=wbLeer.get_sheet_names()[0]
wsLeer=wbLeer.get_sheet_by_name(sheetLeer)

usados=list()
for fi in range(3,wsLeer.max_row+1):
    valPromTh=0
    thCont=0
    valorIniVel=wsLeer.cell(column=2,row=fi).value
    #print(valorIniVel in usados)
    #booleano=not valorIniVel in usados
    #print("Se va a ejecutar? : "+str(booleano) )
    if not valorIniVel in usados:
        #print("Se ejecuta")
        usados.append(valorIniVel)
        #print(valorIniVel)
        for indexB in range(3,wsLeer.max_row+1):

#print(valorIniVel+":::"+wsLeer.cell(column=2,row=indexB).value)
            if valorIniVel==wsLeer.cell(column=2,row=indexB).value:
                print("Se encontro el valor: "+ str(valorIniVel)+" en la
fila: "+ str(indexB))
                print("El valor prom hasta ahora es : "+ str(valPromTh)+"
y se le sumara: "+ str(wsLeer.cell(column=3,row=indexB).value))

valPromTh=valPromTh+float(wsLeer.cell(column=3,row=indexB).value)
                thCont=thCont+1

        valPromTh=valPromTh/thCont
        shThVe.append((valorIniVel,valPromTh))
wbThVe.save(pathThVe)

#_-----------------------------------------------Retardo vs Velocidad-----
-----------------------------------------------
valPromRe=0
thCont=0
wb.save(pathResultado)
```

```
wbThVe=Workbook()
shThVe=wbThVe.active
shThVe.cell(column=1,row=1,value="Retardo vs Velocidad")
shThVe.cell(column=1,row=2,value="Velocidad Vehiculo")
shThVe.cell(column=2,row=2,value="Retardo")

wbLeer=load_workbook(pathResultado)
sheetLeer=wbLeer.get_sheet_names()[0]
wsLeer=wbLeer.get_sheet_by_name(sheetLeer)

usados=list()
for fi in range(3,wsLeer.max_row+1):
    valPromRe=0
    thCont=0
    valorIniVel=wsLeer.cell(column=2,row=fi).value
    #print(valorIniVel in usados)
    #booleano=not valorIniVel in usados
    #print("Se va a ejecutar? : "+str(booleano) )
    if not valorIniVel in usados:
        #print("Se ejecuta")
        usados.append(valorIniVel)
        #print(valorIniVel)
        for indexB in range(3,wsLeer.max_row+1):

#print(valorIniVel+":::"+wsLeer.cell(column=2,row=indexB).value)
            if valorIniVel==wsLeer.cell(column=2,row=indexB).value:
                print("Se encontro el valor: "+ str(valorIniVel)+" en la
fila: "+ str(indexB))
                print("El valor prom hasta ahora es : "+ str(valPromRe)+"
y se le sumara: "+ str(wsLeer.cell(column=4,row=indexB).value))

valPromRe=valPromRe+float(wsLeer.cell(column=4,row=indexB).value)
                thCont=thCont+1

        valPromRe=valPromRe/thCont
        shThVe.append((valorIniVel,valPromRe))
wbThVe.save(pathReVe)

#-------------------------------------Throughput vs Nodos--------------
-----------------------------------------

valPromRe=0
thCont=0
wb.save(pathResultado)

wbThVe=Workbook()
shThVe=wbThVe.active
shThVe.cell(column=1,row=1,value="Throughput vs #Nodos")
shThVe.cell(column=1,row=2,value="# de Nodos")
shThVe.cell(column=2,row=2,value="Throughput")

wbLeer=load_workbook(pathResultado)
sheetLeer=wbLeer.get_sheet_names()[0]
wsLeer=wbLeer.get_sheet_by_name(sheetLeer)

usados=list()
for fi in range(3,wsLeer.max_row+1):
    valPromRe=0
    thCont=0
    valorIniVel=wsLeer.cell(column=1,row=fi).value######nodos/vel
```

```
    #print(valorIniVel in usados)
    #booleano=not valorIniVel in usados
    #print("Se va a ejecutar? : "+str(booleano) )
    if not valorIniVel in usados:
        #print("Se ejecuta")
        usados.append(valorIniVel)
        #print(valorIniVel)
        for indexB in range(3,wsLeer.max_row+1):

#print(valorIniVel+":::"+wsLeer.cell(column=2,row=indexB).value)
            if
valorIniVel==wsLeer.cell(column=1,row=indexB).value:################Nodos
/vel
                print("Se encontro el valor: "+ str(valorIniVel)+" en la
fila: "+ str(indexB))
                print("El valor prom hasta ahora es : "+ str(valPromRe)+"
y se le sumara: "+ str(wsLeer.cell(column=4,row=indexB).value))

valPromRe=valPromRe+float(wsLeer.cell(column=3,row=indexB).value)########
####thr/retar
                thCont=thCont+1

        valPromRe=valPromRe/thCont
        shThVe.append((valorIniVel,valPromRe))
wbThVe.save(pathThNo)

#---------------------------------------Retardo vs Nodos-----------------
-------------------------------------------------

valPromRe=0
thCont=0
wb.save(pathResultado)

wbThVe=Workbook()
shThVe=wbThVe.active
shThVe.cell(column=1,row=1,value="Retardo vs #Nodos")
shThVe.cell(column=1,row=2,value="# de Nodos")
shThVe.cell(column=2,row=2,value="Retardo")

wbLeer=load_workbook(pathResultado)
sheetLeer=wbLeer.get_sheet_names()[0]
wsLeer=wbLeer.get_sheet_by_name(sheetLeer)

usados=list()
for fi in range(3,wsLeer.max_row+1):
    valPromRe=0
    thCont=0
    valorIniVel=wsLeer.cell(column=1,row=fi).value######nodos/vel
    #print(valorIniVel in usados)
    #booleano=not valorIniVel in usados
    #print("Se va a ejecutar? : "+str(booleano) )
    if not valorIniVel in usados:
        #print("Se ejecuta")
        usados.append(valorIniVel)
        #print(valorIniVel)
        for indexB in range(3,wsLeer.max_row+1):

#print(valorIniVel+":::"+wsLeer.cell(column=2,row=indexB).value)
```

```
            if
valorIniVel==wsLeer.cell(column=1,row=indexB).value:################Nodos
/vel
                print("Se encontro el valor: "+ str(valorIniVel)+" en la
fila: "+ str(indexB))
                print("El valor prom hasta ahora es : "+ str(valPromRe)+"
y se le sumara: "+ str(wsLeer.cell(column=4,row=indexB).value))

valPromRe=valPromRe+float(wsLeer.cell(column=4,row=indexB).value)########
####thr/retar
                thCont=thCont+1

        valPromRe=valPromRe/thCont
        shThVe.append((valorIniVel,valPromRe))
wbThVe.save(pathReNo)
```

ORDEN DE EMPASTADO