Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Análisis de la norma 802.11p mediante simulación

Autor: Alfonso Tello Castillo

Tutor: Germán Madinabeitia Luque

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Trabajo Fin de Grado

Grado en Ingeniería de las Tecnologías de Telecomunicación

Análisis de la norma 802.11p mediante simulación

Autor:

Alfonso Tello Castillo

Tutor:

Germán Madinabeitia Luque

Profesor colaborador

Departamento de Ingeniería Telemática

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2017

Trabajo Fin de Grado: Análisis de la norma 802.11p mediante simulación

Autor: Alfonso Tello Castillo

Tutor: Germán Madinabeitia Luque

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2017

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

Quiero agradecer a todos los profesores del Grado de Ingeniería de Tecnologías de Telecomunicación el tiempo y el esfuerzo dedicado a mi formación, y en especial a mi tutor D. Germán Madinabeitia Luque por haberme guiado a lo largo de todo este proyecto. Por supuesto agradecer a mi familia las facilidades que siempre me han brindado para que nunca haya tenido impedimentos en el desarrollo de mi educación académica.

Este es uno de los frutos que da todo el tiempo dedicado por tantas personas.

Gracias.

Alfonso Tello Castillo

Sevilla, 2017

Resumen

En este trabajo se analiza la norma 802.11p, desarrollada para los entornos de comunicaciones vehiculares, mediante el software de simulación NS-3. Ésta es una tecnología aún en desarrollo, y de ser exitosa, promete ofrecernos grandes mejoras en nuestra vida cotidiana.

Comenzaremos con una introducción sobre el concepto en el que se basa su desarrollo, El Internet de las Cosas, posteriormente trataremos la teoría sobre la que se ha sustentado el proyecto y por último simularemos varios escenarios con el objetivo de entender el funcionamiento de éstas redes.

Para finalizar, se extraerán las conclusiones principales sobre cómo funcionan las comunicaciones vehiculares y se recomendarán las posibles líneas futuras de ampliación de este trabajo.

Abstract

In this project we will analyze the standard 802.11p, whose development was focused on the vehicular communiactions, using the simulation software NS-3. This technology is still being developed and, given the case it becomes a success, it can potentially improve our everyday life.

We will start with an introduction to the concept behind its development, the Internet of Things. Then, we will address the theory in which the building of the project is based and finally we will simulate some scenarios in order to understand how this networks work.

Lastly we will collect the main conclusions about the vehicular comunications and we will suggest some posibles ways to continue in future works.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

Internet facilita la información adecuada, en el momento adecuado, para el propósito adecuado.

- Bill Gates -

Internet ha sido, sin ninguna duda, uno de los grandes inventos llevados a cabo por la humanidad. Hoy en día estamos tan acostumbrados a interactuar con él que la mayoría de las veces ni siquiera nos percatamos de su uso. Sin embargo, si echamos la vista atrás y nos acordamos (o preguntamos) cómo era la vida hace unos 30 años, podemos llegar a comprender los profundos cambios que ha experimentado nuestra sociedad gracias a él.

Hoy, Internet está sufriendo una evolución que, poco a poco y desde hace ya unos años, está consiguiendo que volvamos a experimentar nuevos cambios en nuestra sociedad. Para ello muchas tecnologías se están desarrollando actualmente y otras tantas ya son usadas en nuestro día a día.

Con este trabajo se pretende aportar un pequeño paso más a ésta evolución de Internet, mediante el análisis y la experimentación de una de las citadas tecnologías aún en desarrollo, que con suerte, podrá ser implantada en el futuro en nuestra sociedad.

1.1 Motivación

La motivación que hay detrás del desarrollo de este trabajo es la comentada evolución que está sufriendo Internet, y que nos ha llevado a hablar del **Internet de las Cosas** (en adelante IoT) a día de hoy.

IoT es por sí mismo un campo con miles de aplicaciones, y sobre el cuál se están basando muchos trabajos en la actualidad. Introduciremos brevemente qué significa y cuáles son algunas de sus repercusiones, para posteriormente aclarar sobre qué campo de todos los que abarca vamos a trabajar en este proyecto.

1.1.1 El Internet de las cosas

Lo primero que hay que mencionar sobre el IoT es su definición. Aunque no existe una definición formal, según el Grupo de soluciones empresariales basadas en Internet (IBSG), IoT no es más que el momento en el tiempo en el que se conectaron más objetos que personas a Internet.

De acuerdo a esta definición IoT comenzó a existir en un punto entre 2008 y 2009, y la proporción persona a objetos conectados a Internet ha seguido incrementándose desde entonces, y según las previsiones lo seguirá haciendo durante muchos años (1).

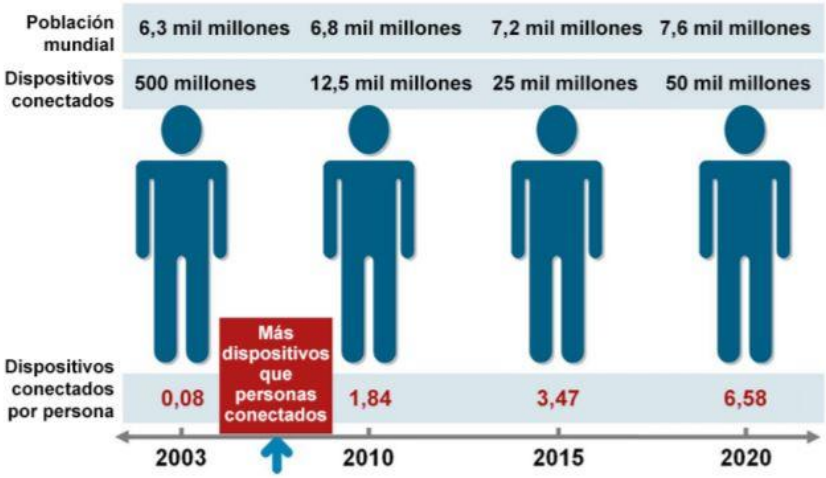


Figura 1-1. Punto en el tiempo en el que nació IoT

Fuente: Cisco IBSG, abril de 2011

Sin embargo, más allá de la definición, es necesario entender el concepto de IoT con un poco de perspectiva. Para ello vamos a comentar qué es lo que teníamos cuando ni siquiera la idea había surgido.

Antes de IoT, cada información que veíamos en la Web había sido recolectada, procesada y presentada por una o varias personas. Esto por supuesto conllevaba un trabajo (muchas veces duro) por parte de estas personas. Por supuesto, la velocidad con la que podíamos acceder a esta información era lenta, es decir, desde que se comenzaba a recopilar la información hasta que podíamos visualizarla en nuestras pantallas podían pasar meses.

Con IoT todo esto cambia. IoT es un cambio en el paradigma de Internet, en el que pasamos de una red donde prácticamente todos los datos eran recopilados y publicados por las personas, a una red donde la inmensa mayoría de éstos serán proporcionados por objetos. Estos, mediante una serie de sensores, serán capaces de conseguirlos de una forma totalmente autónoma y podrán publicarlos en la Web de manera prácticamente instantánea.

La cantidad de objetos a los que se les puede conectar un sensor para obtener información relevante es casi igual a la cantidad de objetos que existen. Por lo que podemos empezar a hacernos una idea del cambio que está suponiendo IoT en nuestras vidas. Sin embargo, para entender aún mejor porque este cambio es tan importante, vamos a comentar brevemente la diferencia entre dos términos que a menudo son confundidos: Internet y la Web.

Internet es una red de dispositivos físicos (switches, routers, etc.) cuya principal funcionalidad es el transporte de datos de una forma rápida, segura y confiable. La Web, por otro lado, se encarga de proporcionar una interfaz al usuario que le permita interactuar con la información que circula por Internet.

Figura 1-2. Internet vs Web

La Web además ha pasado por diferentes etapas a lo largo de su historia: desde una primera fase de investigación y desarrollo hasta su fase actual, la web "social", que permite a las personas comunicarse, conectarse y compartir información entre ellos. No es necesario pararse y explicar cómo ha cambiado la manera que tenemos de interactuar con la Web desde su comienzo hasta hoy en día.

Figura 1-3. Evolución de la Web desde sus inicios hasta la etapa actual

Por el contrario, Internet nunca había sufrido un "cambio de etapa" que le hubiese permitido evolucionar de una forma notable (aunque por supuesto haya estado en constante desarrollo todo este tiempo). Surgió como medio para que las personas pudiésemos transportar información de una forma rápida, segura y fiable, y no se había movido de ahí. Esto cambia con IoT, ahora Internet tiene que evolucionar hacia una estructura que nos permita manejar la cantidad de información que, no solo las personas, sino también los objetos, nos van a proporcionar.

Con el tiempo, esta información, nos va a permitir ser mucho más conscientes del entorno que nos rodea, comprenderlo mejor y por lo tanto ser capaz de gestionarlo de una manera mucho más eficiente. El potencial de IoT es enorme, tanto que, en pocos años se nos antojará impensable la cantidad de datos que estábamos pasando por alto en nuestros día a día.

Figura 1-4. Evolución de Internet desde sus inicios hasta el IoT

En la actualidad ya se están llevando a cabo proyectos de IoT y se espera que, entre otras cosas, éste acabe siendo un recurso para mejorar la gestión de prácticamente todo, como pueden ser las ciudades, las fábricas, las granjas, las carreteras, etc., y para poder mejorar la calidad de vida las personas entre otras cosas.

1.1.2 Comunicaciones V2X

Dentro de todos los campos de aplicación del IoT, en este proyecto nos vamos a centrar en las comunicaciones vehículo a vehículo (V2V) y vehículo a infraestructura (V2I), o V2X para referenciar a ambas. Estas comunicaciones son una tecnología actualmente en desarrollo donde los vehículos y unas balizas situadas en lugares de interés, serán los nodos de la red.

A grandes rasgos lo que se busca es que tanto las balizas como los vehículos sean capaces de recoger información a través de sensores y que, tras el procesamiento de ésta, ambas entidades tengan la capacidad de intercambiarse información entre ellas para poder ser subida a Internet, o bien para ser gestionada

individualmente si se trata de información de interés para el vehículo.

No es objetivo de las comunicaciones V2X manejar la información en Internet, pues las opciones que surgen cuando una cierta cantidad de datos es subida a la red son prácticamente infinitas.

Aun así, algunas de las aplicaciones más interesantes y de las que se pretenden desarrollar en primer lugar son el poder proveer información para la seguridad de los conductores, información relacionada con el tráfico o mediciones sobre la calidad del aire en las ciudades, todo ello prácticamente en tiempo real.

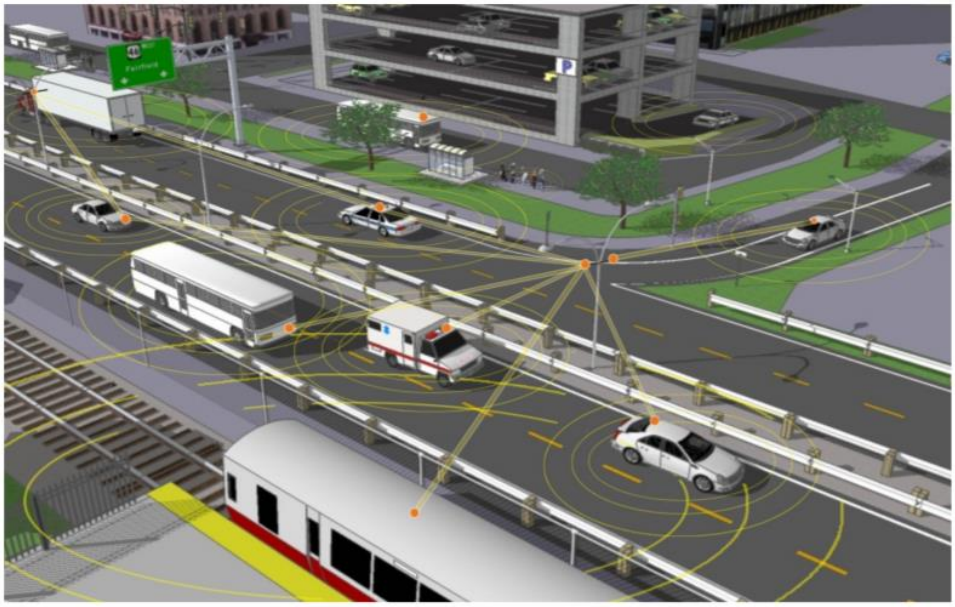


Figura 1-5. Comunicaciones V2V

1.2 Objetivos

Como hemos comentado, las comunicaciones V2X es una tecnología aún en desarrollo en la que hay que investigar numerosos factores.

Concretamente en este proyecto vamos a estudiar el comportamiento de este tipo de redes y cuáles son los límites de estas comunicaciones. Es decir, cuántos bytes somos capaces de enviar entre vehículo y vehículo o baliza y vehículo, en el tiempo en que ambos permanecen lo suficientemente cerca uno de otro, de forma que el intercambio de mensajes vía inalámbrica sea posible.

Para ello vamos a codificar y simular dos escenarios:

1. En el primer escenario vamos a estudiar cual es la tasa eficaz que podemos enviar entre una baliza y un vehículo en función de la velocidad de éste último.

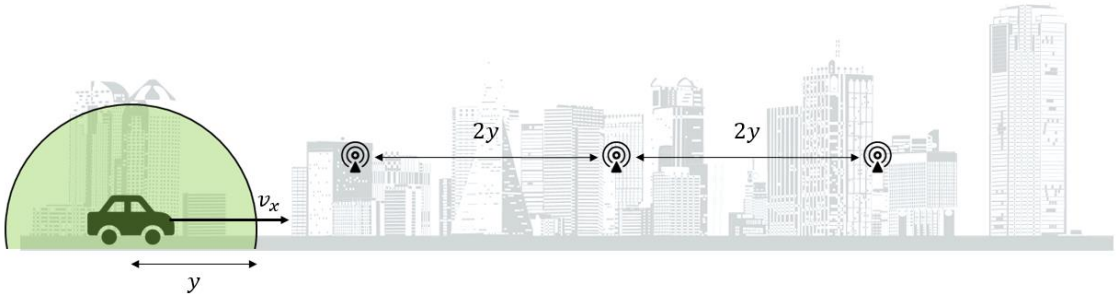


Figura 1-6. Escenario 1

2. En el segundo escenario volveremos a estudiar la tasa eficaz que podemos enviar pero en este caso entre dos vehículos que se aproximan entre ellos.

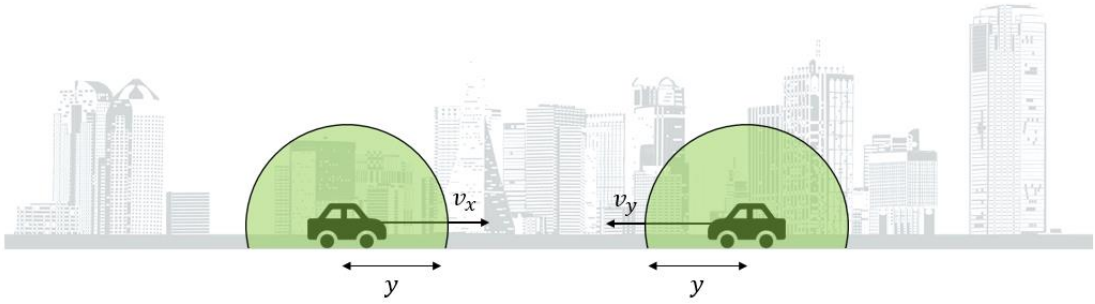


Figura 1-7. Escenario 2

1.3 Metodología

Para la realización de este proyecto se han llevado a cabo las siguientes etapas:

1. **Documentación.** Una primera fase que consiste en la lectura y análisis de la norma 802.11p, enfocándose sobre todo en las características más importantes de ésta. Además, se estudiará como NS-3 (ver capítulo 2) ha implementado la norma 802.11p y los distintos modelos de movilidad que existen en este software de simulación, de forma que podamos desarrollar un modelo con garantías del escenario que se desea simular.
2. **Codificación, validación del modelo.** En esta fase codificaremos en NS-3 el modelo de los escenarios que deseamos simular. Además, comprobaremos que los modelos usados son fiables para poder obtener unos datos fiables.

3. **Diseño de experimentos.** En la que se decidirán que experimentos llevar a cabo mediante las simulaciones. Definiendo los parámetros que se modificarán y los resultados que se esperan obtener.
4. **Simulación.** Una vez verificado el modelo y diseñado los experimentos, esta fase consistirá en extender el tiempo de simulación a días para poder obtener unos resultados precisos.
5. **Interpretación de los resultados.** Donde se analizarán los resultados de la simulación para poder interpretarlos correctamente y de esta forma comprender el funcionamiento de estas redes y obtener unas conclusiones finales, completando de este modo el objetivo principal de este trabajo.

2 NS

NS (más concretamente su versión actual NS-3), es un **software de simulación de redes** basado en eventos discretos, usado principalmente en la investigación y la enseñanza. Desarrollado en un lenguaje de programación orientado a objetos (C++) y con las interfaces programadas en Python.

Se trata de un software libre y abierto distribuido bajo la licencia GNU GPLv2 que cuenta con una amplia comunidad de desarrolladores que lanzan una versión estable cada tres meses, con nuevos módulos desarrollados, documentados, validados y mantenidos por investigadores.

Es el software que hemos elegido para realizar este proyecto. A lo largo del capítulo comentaremos por qué y mencionaremos cuáles son las características más importantes de este simulador.

Figura 2-1. Logo del simulador NS-3

2.1 Conceptos teóricos

Pero antes de comentar las principales características de NS-3, vamos a explicar brevemente los fundamentos teóricos más importantes en los que se sustenta su desarrollo.

- La simulación.
- Simulaciones basadas en eventos discretos.
- Programación orientada a objetos.

2.1.1 ¿Qué es una simulación? ¿Para qué sirve?

La simulación es una de las herramientas más poderosas que tienen los ingenieros a la hora de estudiar un sistema, pues hace posible el estudio, análisis y evaluación de éste, que de otra forma sería imposible. Bien porque manejar y manipular el sistema real puede ser extremadamente caro o complicado, o bien porque ni siquiera existe aún.

Definiremos la simulación como el proceso de **diseñar el modelo** de un sistema real y realizar experimentos sobre éste, con el propósito de **entender su comportamiento** y/o **evaluar** varias estrategias para optimizar su funcionamiento. (2)

En esta definición nos encontramos con los conceptos más importantes de una simulación:

1. El **modelado.** Es decir, una simplificación del sistema real que nos permite responder a preguntas sobre éste sin tener que experimentar sobre dicho sistema. Es básico en una simulación, si el modelo del sistema que queremos estudiar no es correcto cualquier información que este nos vaya a proporcionar será también incorrecta.
2. **La obtención de conclusiones**. Hablar de realizar una simulación sin tener como objetivo final la obtención de unos datos que nos permitan tomar decisiones no tiene sentido. Puede parecer obvio

pero hay que remarcar que existe una diferencia entre obtener unos resultados numéricos y analizar estos para obtener unas conclusiones.

Para abordar el proceso de una simulación, existen diferentes estrategias. Sin embargo, la más extendida y la que utiliza NS es la simulación basada en eventos discretos.

2.1.2 Simulación basada en eventos discretos

Las simulaciones basadas en eventos discretos se caracterizan (frente a las simulaciones basadas en tiempo continuo) por un control del tiempo que permite hacer avanzar éste a intervalos de una longitud variable.

El elemento principal de estas simulaciones son los eventos. Un evento es un suceso que hace cambiar el estado del sistema. Por ejemplo, si nos encontramos simulando un sistema que representase una estación de ferrocarriles, un evento podría ser la llegada de un tren, su salida o incluso la compra de un billete por parte de un pasajero.

El tiempo de simulación es otro concepto importante. En este tipo de simulaciones, es un valor común a todo el sistema que marca el momento del tiempo en el que nos encontramos, avanza en intervalos discretos y variables en longitud, y sólo ante la ocurrencia de eventos.

Existen varias formas de tratar los eventos en este tipo de simulaciones. A continuación, explicaremos cuál es la que usa NS-3, que es además el método más extendido. (3)

2.1.2.1 El método de planificación de eventos discretos

Como acabamos de mencionar, dentro de las simulaciones basadas en eventos discretos, el método de planificación de eventos discretos es el más extendido. Dicho método se basa en planificar la ocurrencia de eventos en el futuro (ya sea de una forma estocástica o determinista). (3)

Para cada evento se programa (en el momento en el que se define) en que tiempo se va a ejecutar, y se guarda en una estructura de datos denominada **calendario de eventos,** con una cola con política de prioridades. Esta política para este caso consistirá en tratar antes los eventos que hayan sido programados con un tiempo de ejecución menor.

Una vez el tiempo de simulación coincide con el tiempo en el que un evento se ejecuta, éste se marca como ejecutado en el calendario de eventos y se llevan a cabo todas las acciones que conlleve (entre las que se pueden incluir la generación de nuevos eventos).

Este proceso se repite hasta que no quedan más eventos que procesar, o cuando se alcanza un tiempo máximo prefijado que hace que se la simulación se detenga.

De una forma esquematizada podríamos resumir el método con la siguiente tabla:

PASOS	Acciones	Condición
0	Iniciación del tiempo de simulación t_s ($t_s = 0$)	
1	Finalización de la simulación	El calendario de eventos está vacío o se ha alcanzado el tiempo máximo de simulación
2	Se extrae el evento E con menor el menor tiempo de ejecución (t_e) del calendario	
3	Se actualiza el tiempo de simulación ($t_s = t_e$)	
4	Se ejecutan todas las acciones relacionadas con la ocurrencia del evento E	
5	Vuelta al paso 1	

Tabla 2-1. Algoritmo del método de planificación de eventos discretos

2.1.2.2 Ejemplo

Comentemos un breve ejemplo para poder entender mejor los beneficios de estas simulaciones y su funcionamiento.

Supongamos que queremos simular un sistema que represente una red real. En este caso nuestro sistema será simple, contaremos con dos equipos (A y B) conectados directamente mediante un cable donde el equipo A deseará enviar un mensaje al equipo B. Por lo tanto, en nuestro sistema tendremos tres entidades, los dos PCs y el cable.

Antes de empezar la simulación programaremos la ocurrencia del primer evento E_0 (con tiempo $t_{e0} = t_0$): el equipo A genera un mensaje que debe ser enviado al equipo B:

1. Una vez comenzada la simulación, el tiempo de simulación avanzará al tiempo en el que nuestro primer evento fue programado, $t_s = t_0$.
2. Se marcará como ejecutado dicho evento y se procesará. Supondremos en este caso que el evento E_0 genera otro evento E_1 con tiempo t_{e1} y que representa el tiempo que el mensaje tarda en recorrer todas las capas de protocolo de un equipo estándar hasta que está listo para ser enviado por la capa física (que conecta directamente con el cable).
3. Al no haber más eventos entre t_{e_0} y t_{e_1} el tiempo de simulación avanzará a $t_s = t_{e_1}$, se marca como ejecutado dicho evento (E1) de la cola y se procesa.
4. Este evento genera otro evento E_2 con tiempo t_{e2} que será igual al tiempo que el mensaje tarde en recorrer el cable hasta llegar al equipo B (nótese que dicho tiempo puede ser generado de manera aleatoria).

5. Por último, t_s avanza hasta t_{e_2} y se procesa el evento. Este último evento no genera otro (pues el mensaje ya ha llegado a su destino) por lo tanto el calendario de eventos se vacía y la simulación concluye.

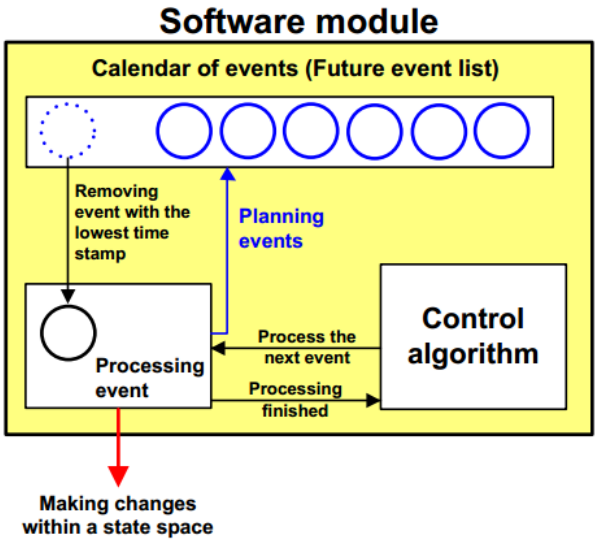


Figura 2-2. Ciclo de vida de un evento en simulaciones basadas en eventos discretos

Aunque este ejemplo está muy simplificado nos sirve para ilustrar cual es el funcionamiento básico de NS-3. La principal ventaja de este tipo de simulaciones es que nos permite abstraernos de todo lo que pasa en un sistema mientras no ocurran eventos que nos interesen, reduciendo de forma notable los costes de estudiar sistemas mediante simulaciones. A la hora de simular redes, esta característica es muy beneficiosa pues en éstas solo nos interesa estudiar una serie de eventos muy definidos, que son los que afectan a los parámetros de los que queremos obtener información, siendo todos los demás eventos totalmente innecesarios.

2.1.3 Programación orientada a objetos

NS-3 está programado en C++, un lenguaje de programación orientado a objetos. Vamos a comentar brevemente qué ventajas tienen este tipo de lenguajes y por qué es el que usa NS-3.

La programación orientada a objetos (POO) fue una evolución de la programación procedural basada en funciones, debido a que esta última hacía que para aplicaciones muy grandes, la modificación de una sola de sus líneas implicaba la modificación de muchas otras líneas de otras funciones que estuviesen relacionadas, haciendo que fuese un trabajo arduo y laborioso.

La POO no implica necesariamente que el código sea mejor ni más avanzado, sino simplemente más estratégico pues hace que el mantenimiento de este sea mucho más flexible.

El concepto de la POO es "partir" el código de forma que una serie de objetos con una lógica interna, puedan comunicarse con otros objetos con otra lógica interna distinta, encapsulando de esta forma el código, de manera que la modificación de estos objetos no afecte a otros. Además, esto lleva a una de las ventajas más importantes de la POO: la reutilización de código, ya que es extremadamente simple coger un objeto programado por un tercero, con una lógica totalmente independiente, e incorporarlo a tu programa. A estos objetos independientes se les llama clases.

El hecho de que detrás de NS-3 haya una comunidad de programadores muy amplia, y que además cualquier persona pueda desarrollar su propio código de forma independiente para que pueda ser implementado en una de sus versiones, podría ser uno de los motivos por los que se decidió desarrollar el software en este tipo de lenguajes.

2.2 Características principales del simulador

Comprendido ahora cuáles son los fundamentos teóricos de NS-3 mencionaremos cuáles son sus principales virtudes y algunas de sus características más interesantes para este proyecto. (4)

- La clase Simulator
- Las clases Helper
- Los atributos en NS-3
- Las trazas

2.2.1 La clase Simulator

Una de las clases más importante de NS-3 es la clase Simulator. Es la que se encarga de la simulación en sí, ofreciendo un reloj de simulación que proporciona un tiempo común para todo el sistema, mecanismos para arrancar y parar el planificador (el equivalente al calendario de eventos antes visto), y mecanismos de gestión de eventos que permiten añadir, eliminar y cancelar éstos.

2.2.1.1 Gestión de eventos

La clase Simulator nos permite mediante el método Schedule programar un evento basándonos en tres parámetros: el instante en el que se ejecutará ese evento, el objeto que debe gestionar el evento, y el método con el que se debe gestionar el evento. Por ejemplo, podríamos realizar la siguiente declaración:

```
Schedule (20ms, tarjetaFisicaWiFi, llegadaPaquete);
```

con la que estaríamos programando un evento que se produciría a los 20ms del tiempo simulado, que además se encargaría de gestionar la clase tarjetaFisicaWiFi con el método llegadaPaquete.

Además, la clase Simulator nos ofrece otros tres métodos para programar eventos:

- **ScheduleNow (equivalente a Schedule (0, …))**. Programa un evento en el tiempo de simulación en el que se encuentre la simulación.
- **ScheduleDestroy**. Programa un evento que se ejecutará al finalizar la simulación.
- **Cancel**. Nos permite cancelar un evento ya programado.

Con todo esto se puede tener un control total de todos los eventos que ocurren en una simulación.

2.2.2 Las clases Helper

NS-3 proporciona una serie de clases muy útiles para la programación, las clases Helper. En NS-3 todo se programa a bajo nivel. Por poner un ejemplo, si se quiere programar un equipo con una aplicación simple, se ha de definir un contenedor que almacenará la capa física (en realidad nos referimos a la clase que modela la capa), la capa MAC, la capa IP y todas las que sean necesarias, que por supuesto hay también que definir. Además, en cada una de estas capas habrá que configurar una serie de parámetros y después, interconectar estas capas entre ellas.

Toda esta configuración se hace muy tediosa y (en la mayoría de las situaciones) innecesaria. Por ello NS-3 ofrece las clases Helper, estas clases realizan todos los procesos que hemos comentado de una forma automática, siendo solo necesario por nuestra parte invocar un par de métodos y configurar un par de parámetros. Con esto conseguimos que la programación del código sea mucho más ágil. (5)

2.2.3 Atributos

En cualquier lenguaje POO un concepto muy importante son los atributos de una clase, es decir, las variables que definen los parámetros de nuestra clase. Una de las características que diferencian a NS-3 de otros softwares es la forma que tiene de tratar con estos atributos.

Normalmente para poder modificar los atributos de una clase hay dos opciones: o bien modificamos el código en función de que valor queramos que tenga, o programamos nuestro software para que acepte valores de esos argumentos a la hora de arrancarlo (es decir, por línea de comandos). Esto último tiene una desventaja, será necesario conocer la lista de parámetros que son modificables y asignarle un valor a cada uno de ellos en cada ejecución.

NS-3 ofrece otra forma de tratar estos atributos. En NS-3 podemos modificar sus valores por línea de comandos con la diferencia de que en este caso podemos elegir cuántos atributos modificar, sin tener que especificar una lista entera. Además, todo esto se hace de una forma que apenas tiene consecuencias sobre el rendimiento de la ejecución.

Esta característica es muy atractiva en simulaciones de redes, pues aquí es siempre muy interesante observar como varían los resultados en función de uno solo de los parámetros o varios de ellos dejando los demás constante.

2.2.4 Trazas

Antes de introducir el concepto de trazas debemos explicar brevemente que son los callbacks. Los callbacks son una utilidad que nos permite programar qué método queremos invocar ante la ocurrencia de un evento (llamado disparador en NS-3), desde cualquier punto del programa y sin introducir dependencias adicionales.

Para que los callbacks sean útiles debe existir un método que se subscriba a ellos, es decir, que quiera ser avisado cuando se produzca un evento E. Por ejemplo, sería interesante suscribir el método *contadorPaquetesTirados* a un callback que le invocase cuando se produjera el descarte de un paquete por cualquier motivo.

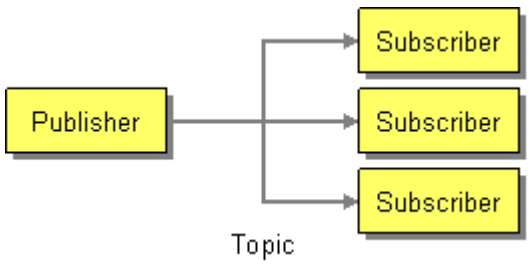


Figura 2-3. Patrón publicador suscriptor.

Entendido este concepto no es difícil extenderlo al concepto de traza. Las trazas no son más que callbacks ya programados en NS-3 que permiten actuaciones ante eventos muy específicos de la red como, por ejemplo:

- Inicio de transmisión de un paquete
- Fin de recepción de un paquete
- Asociación a un punto de acceso
- Etc.

Con las trazas y una serie de métodos que habrá que programar para realizar las acciones necesarias según los objetivos de la simulación, seremos capaces de tener bajo control todos los eventos que están ocurriendo en nuestro sistema, y podremos analizarlos, procesarlos, presentarlos en forma de estadísticos y, el objetivo final de toda simulación, obtener conclusiones.

2.3 Los módulos WiFi y movilidad

Vamos a introducir de forma superficial los dos modelos más importantes de NS-3 para la realización de este trabajo: el modelo WiFi y el modelo Movilidad

2.3.1 WiFi

2.3.1.1 Descripción

En NS-3 el módulo WiFi se modela con diferentes clases que simulan el funcionamiento de una tarjeta WiFi desde su nivel físico hasta su nivel MAC. (6)

Concretamente en NS-3 podemos encontrarnos modelados en clases los siguientes componentes:

- En canal físico
- La capa física (PHY).
- La capa MAC baja, encargada de las funciones de acceso al medio. En NS-3 se añade además la capa MAC media, encargada del acceso al canal compartido.
- La capa MAC alta, que implementa las acciones no críticas en términos de tiempo como puede ser la generación de Beacons, la asociación o el control de la tasa de transmisión.

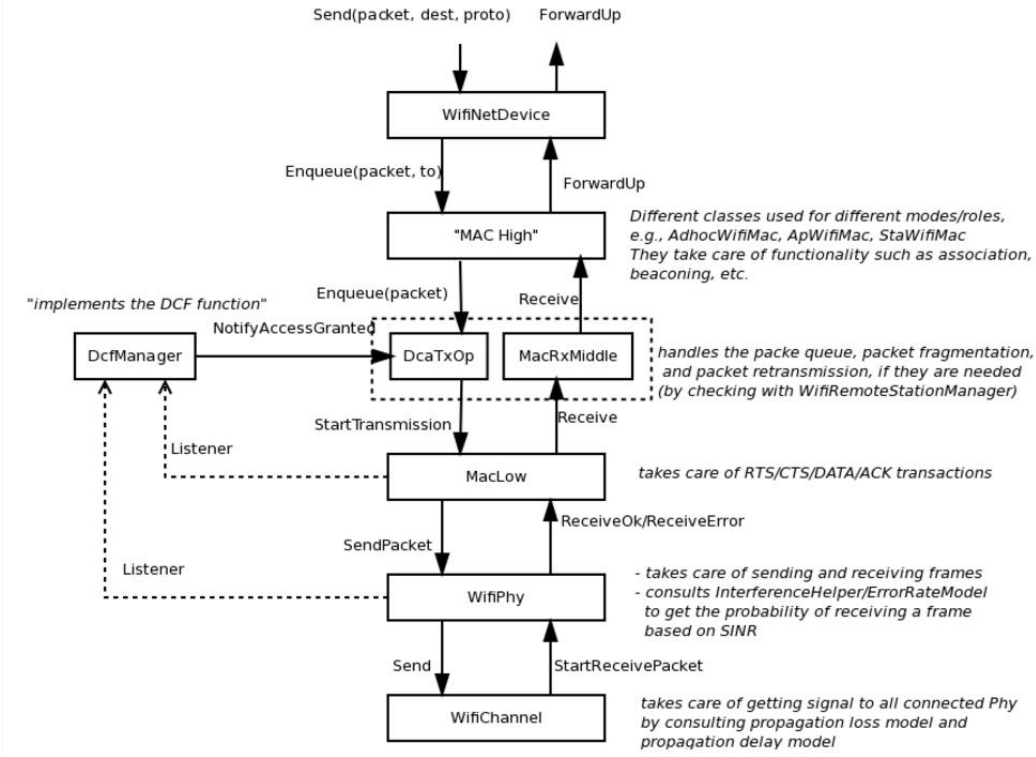


Figura 2-4. Módulo WiFi en NS-3

El objeto WifiNetDevice que vemos en la parte superior de la imagen se encarga de englobar todos los objetos comentados anteriormente en una sola entidad. En NS-3 un nodo puede contener varios objetos NetDevice, así como los ordenadores pueden contener varios tipos de dispositivos, como pueden ser un dispositivo WiFi, una tarjeta de red Ethernet, etc.

2.3.1.2 Capa física

La capa PHY es la que se encarga de modelar la recepción de los paquetes y el consumo de energía. Se tienen en cuenta tres factores a la hora de recibir un paquete:

1. Cada paquete recibido se evalúa probabilísticamente para aceptar o rechazar la recepción. Esta

probabilidad depende de la modulación, la relación señal a ruido (e interferencia) y el estado de la capa física.

2. Existe un objeto para rastrear todas las señales recibidas de modo que se pueda modelar la potencia de interferencia correcta para cada paquete recibido.
3. Uno o varios modelos de errores son usados para encontrar la probabilidad de recibir un paquete correctamente.

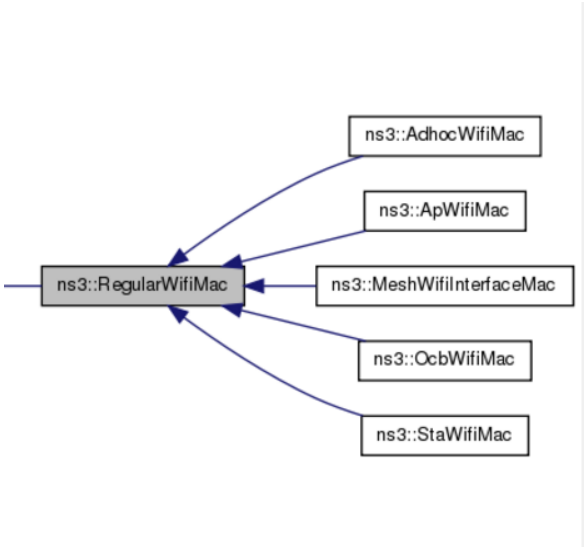
2.3.1.3 Capa MAC baja

La capa MAC baja se compone de estos elementos:

1. ns3::MacLow que se encarga de las transmisiones de RTS, CTS, DATA y ACK
2. ns3::DcfManager y ns3::DcfState que implementan las funciones DCF y EDCAF
3. ns3::DcaTxop y ns3::EdcaTxopN que maneja la cola de paquetes, la fragmentación de paquetes y las retransmisiones.

2.3.1.4 Capa MAC alta

Existen en la actualidad cinco modelos de capa MAC alta, dependiendo del tipo de red que se desee implementar:



En nuestro caso nos va a interesar usar el ns3::OcbWifiMac que es el que modela la capa MAC en las comunicaciones V2V.

Todas ellas derivan de la clase ns3::RegularWifiMac que implementa algunas características comunes a todos los tipos de WiFi usados como puede ser la posibilidad de ofrecer calidad de servicio (QoS).

2.3.2 Movilidad

2.3.2.1 Descripción

La movilidad en ns3 incluye lo siguiente:

- Un conjunto de modelos de movilidad usados para mantener y determinar la posición y la velocidad de un objeto.
- Trazas que notifican cuándo cambia el rumbo de un objeto.
- Una serie de clases Helper que facilitan la asignación de un modelo de movilidad a un objeto con su correspondiente posición inicial y velocidad. (7)

2.3.2.2 Diseño

El diseño del módulo se basa en tres clases: las clases MobilityModel, las clases PositionAllocator y las clases Helper.

En NS-3, las clases MobilityModel se usan para llevar un control de la posición y la velocidad de un objeto. Normalmente, se agregan a una clase Node y se accede a ellas usando el método GetObject<MobilityModel>() de la clase Node. Dependiendo del tipo de movilidad que deseemos, ns3 nos ofrece un subconjunto de clases derivadas de MobilityModel que veremos más adelante.

Para establecer la posición inicial de un objeto, normalmente se hace uso de la clase PositionAllocator. En un uso normal de ns3, esta clase no volverá a ser usada durante toda la aplicación, aunque en algunos casos como en el de movilidad aleatoria se volverá a usar para tomar nuevos parámetros.

```
MobilityHelper movilidadBaliza;

Ptr<ListPositionAllocator> positionBalizas = CreateObject<ListPositionAllocator> ();
positionBalizas->Add (Vector (distanciaBalizas * i, 0.0, 3.0));

movilidadBaliza.SetPositionAllocator (positionBalizas);
movilidadBaliza.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
movilidadBaliza.Install (nodes.Get(i));
```

Figura 2-5. Ejemplo típico de codificación de movilidad en NS-3

Además, normalmente se suele hacer uso de las clases helper, en este caso en concreto de la clase MobilityHelper, que nos facilita y nos hace mucho más visual la creación de un modelo de movilidad y su asociación a un objeto de nuestra simulación.

2.3.2.3 Sistema de coordenadas

De momento, una de las limitaciones de la movilidad en ns3, es el sistema de coordenadas. Mientras que existen numerosos sistemas de coordenadas para poder usar, y sus respectivas transformaciones entre ellos, ns3 solo implementa el sistema de coordenadas cartesiano.

Aun así, si se desea usar otro sistema de coordenadas, el usuario tiene libertad de implementar la posibilidad de convertir del sistema cartesiano a el que desee usar y viceversa.

En un futuro se espera que NS implemente esta posibilidad. Pero de momento, se hace necesario trabajar con el sistema cartesiano.

2.3.2.4 Coordenadas

La clase base usada para implementar las coordenadas es la clase Vector. Esta clase se usa tanto para las posiciones como para las velocidades.

Además, se pueden definir una serie de estructuras que facilitan la creación de los modelos de movilidad:

- Rectángulos
- Prismas rectangulares
- WayPoints: cada uno define conjunto tiempo - posición.

2.3.2.5 Subclases

Las distintas subclases que ns3 define para implementar la movilidad son las siguientes:

- **ConstantPositionMobilityModel.** La más sencilla de todas, en ella la posición de un objeto es constante hasta que se fije una nueva.
- **ConstantVelocityMobilityModel.** Modelo de movilidad para el cual la velocidad vectorial del objeto no cambia a no ser que se fije una nueva.
- **ConstantAccelerationMobilityModel.** Modelo en el que la aceleración vectorial no cambia a no ser

que se fije una nueva.

- **GaussMarkovMobilityModel.** Este es el único caso en el que el modelo de movilidad posee memoria y aleatoriedad, es decir, la velocidad del objeto en un momento t será escogida de una manera aleatoria, pero teniendo en cuenta la velocidad en el momento t inmediatamente anterior.
- **HierarchicalMobilityModel.** Aquí se combinan dos modelos, el padre y el hijo. La posición en este modelo siempre será la suma de las posiciones padre e hijo. Muy útil cuando se quiere simular un nodo dentro de otro nodo que se está moviendo.
- **RandomDirection2dMobilityModel.** En este modelo cada objeto espera por un tiempo específico, selecciona una velocidad y una dirección aleatorias y comienza a moverse en esa dirección hasta que alcanza uno de los límites definidos en el modelo. En ese momento se vuelve a pausar, selecciona una nueva dirección y velocidad y así cíclicamente.
- **RandomWalk2DMobilityModel.** En el que cada objeto se mueve con una velocidad y dirección aleatoria hasta que se recorre una distancia predeterminada o un tiempo predeterminado. Si el objeto alcanza alguno de los límites definidos en el modelo, rebotará con un ángulo reflexivo.
- **WaypointMobilityModel.** Aquí se definen una serie de WayPoints (cada uno de ellos contendrá un tiempo y una posición). El objeto comenzará en la posición del primer WayPoint y con velocidad 0, a partir de ahí se moverá al siguiente con velocidad constante y en un intervalo igual a la diferencia de tiempo entre el WayPoint origen y destino. Cuando haya alcanzado el último WayPoint el objeto se detendrá y permanecerá en esa posición hasta el final de la simulación.
- **RandomWaypointMobilityModel.** En este modelo cada objeto empezará pausado durante un tiempo aleatorio. Después de esta pausa seleccionará un punto de destino aleatorio y se comenzará a mover con una velocidad también aleatoria. Cuando alcance su destino el proceso comenzará de nuevo.
- **SteadyStateRandomWaypointMobilityModel.** Basado en el modelo anterior, pero en este caso los tiempos de pausa, la velocidad y la dirección son elegidos siguiendo una distribución uniforme.

3 802.11P

Los que se enamoran de la práctica sin la teoría son como los pilotos sin timón ni brújula, que nunca podrán saber a dónde van.

- Leonardo da Vinci -

Las redes V2X que comentamos brevemente en la introducción de esta memoria, plantean un reto que no plantea ningún otro tipo de redes: el tiempo del que se dispone para intercambiar información.

En otros tipos de comunicaciones, como la que se podría producir entre un equipo con WiFi y un router, el tiempo del que se dispone para intercambiar información no es un problema a la hora de diseñar la red y su funcionamiento. Esto es debido a que el usuario normalmente puede elegir, o sabe de antemano, cuánto tiempo va a estar dentro de la zona de cobertura. Este tiempo además suele ser elevado, y hace que sea inusual el caso en el que el usuario no disponga del tiempo necesario para conseguir la información deseada. Por tanto, renunciar a ciertas prestaciones que puede ofrecer la red por este improbable inconveniente no merece la pena.

Sin embargo, este tiempo se reduce significativamente en las comunicaciones V2X debido a la velocidad de los vehículos, y aquí sí surge un problema pues un automóvil no puede frenar durante un trayecto para que dé tiempo a intercambiar la información que hiciese falta. En esta ocasión sí que fue necesaria la búsqueda de soluciones y la modificación del comportamiento de estas redes para corregir este contratiempo.

Así, se hizo indispensable extender la norma 802.11 para permitir las comunicaciones en estos entornos. Para ello se desarrolló una nueva cláusula, la 802.11p. A ésta también se le conoce como la cláusula de Acceso Inalámbrico en Entornos Vehiculares o WAVE por sus siglas en inglés.

3.1 Introducción

En 1999 la FCC estadounidense reservó 75 MHz en la banda de los 5.9 GHz para ser usados exclusivamente en las comunicaciones V2X, un tipo de comunicación de corto alcance (DSRC). Posteriormente, en 2008, Europa haría lo propio reservando en este caso 50 MHz en la misma banda (y otros 25 MHz para futuras aplicaciones).

El objetivo principal de las comunicaciones V2X es mejorar la seguridad y el flujo de tráfico en las carreteras.

Por ejemplo: un vehículo A podría avisar de que está frenando o acelerando cuando se aproxime a un semáforo, consiguiendo una mejora significativa en la coordinación de los automóviles y mejorando la fluidez del tráfico en las carreteras.

Por otro lado, otro vehículo B que ha sufrido un accidente tendría la capacidad de avisar con varios cientos de metros de antelación a otros coches que se estén aproximando a él, de forma que puedan reducir su velocidad.

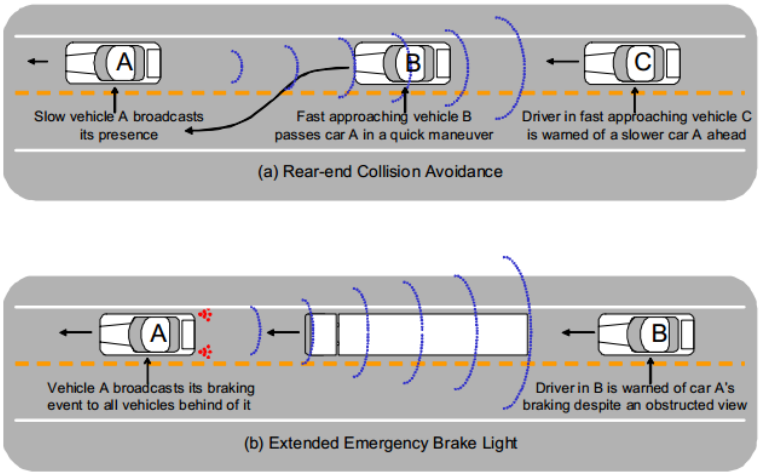


Figura 3-1. Aplicaciones de las comunicaciones V2X

Estas dos aplicaciones y muchas otras podrían desarrollarse a raíz de una implantación de las tecnologías de comunicación entre vehículos. Para conseguir esto, uno de los primeros pasos que se dio fue la asignación de las frecuencias en las que transmitir y la organización de los canales. (6)

En EEUU, por ejemplo, se realizó la siguiente distribución del espectro de frecuencia:

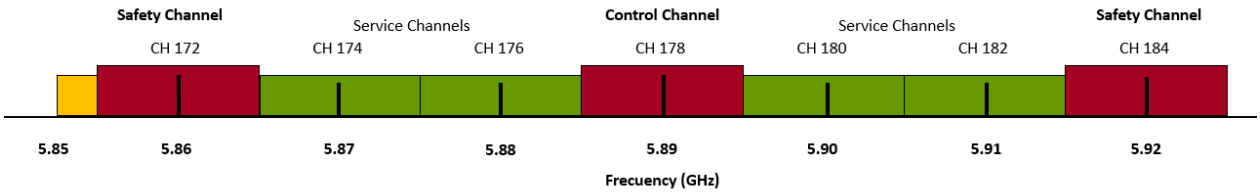


Figura 3-2. Distribución de los canales para 802.11p en EEUU

Reservándose:

- El canal 178 para el canal de control. Éste está restringido a comunicaciones de seguridad.
- Los canales 172 y 184 para casos especiales de seguridad prioritaria.
- Los otros 4 canales están dedicados a propósitos generales.
- Además, existe un octavo canal de 5 MHz al principio de la banda usado como banda de guarda.

En Europa la organización de los canales fue algo diferente, aunque se persiguió el mismo propósito: (7)

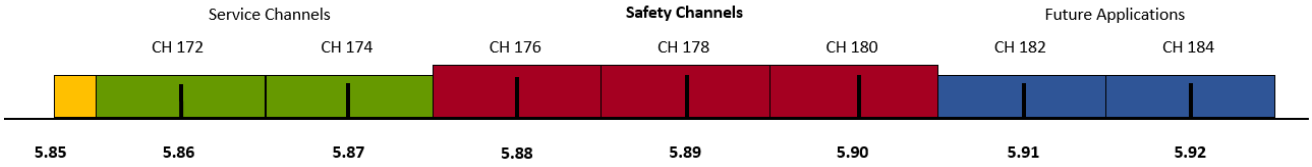


Figura 3-3. Distribución de los canales para 802.11p en Europa

Reservándose:

- Los canales 176, 178 y 180 para comunicaciones relacionadas con la seguridad
- Los canales 172 y 174 para comunicaciones de propósito general

- Los canales 182 y 184 para aplicaciones futuras
- Además de un octavo canal de 5 MHz usado como banda de guarda.

Cabe destacar que la banda en la que se usa 802.11p, la banda de los 5.9 GHz, es una banda libre, pero con licencia. Esto significa que se puede transmitir gratuitamente en ella, pero ha de hacerse siguiendo las pautas de unas determinadas tecnologías.

Por contrastar con otras bandas, la ampliamente usada banda de los 2.4 MHz permite transmitir gratuitamente, y hacerlo sobre la tecnología que el usuario considere (como WiFi o Bluetooth). Los canales de televisión, sin embargo, tienen que pagar por usar las frecuencias que los gobiernos le facilitan para poder transmitir, siguiendo además las pautas que el gobierno le requiera.

En la banda de 5.9 GHz nos encontramos con una situación intermedia. Por un lado, no será necesario pagar para poder transmitir en estas frecuencias. Por el otro, no se podrá hacer de forma libre si no que todo aquél que transmita en estas frecuencias deberá hacerlo siguiendo los estándares definidos.

Cómo último punto, destacar que el objetivo de la IEEE es estandarizar las tecnologías V2X mundialmente, para lo que está desarrollando toda una serie de protocolos, agrupados en lo que se conoce como la torre de protocolos WAVE. (6)

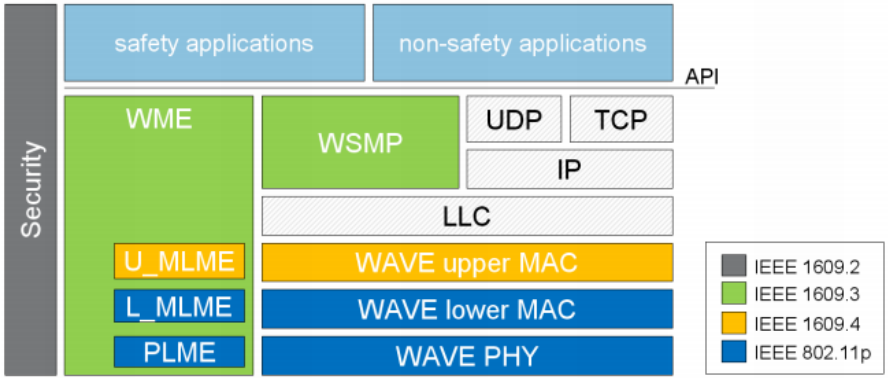


Figura 3-4. Torre de protocolos WAVE

Donde podemos observar que 802.11p es sólo una parte de esta torre. Concretamente la cláusula que nos ocupa define la capa PHY y la capa MAC inferior (que se encarga de las comunicaciones en un solo canal).

Para la coordinación en la comunicación de varios canales a la vez existe el estándar 1609.4. Las capas de red y transporte del modelo OSI serían implementadas por la norma 1609.3. La norma 1609.2 es la encargada de la seguridad entre las aplicaciones y los procesos que se ejecutan en distintas capas. Además, la IEEE aún se encuentra desarrollando otra serie de estándares como la 1609.6 que será el encargado de los servicios de gestión remota.

Ahora que tenemos un poco de perspectiva de en qué situación nos encontramos, vamos a pasar a comentar superficialmente cuáles han sido los cambios más importantes de 802.11p.

3.2 Capa MAC

Como ya hemos visto, la motivación por la cual se desarrolló la norma 802.11p fue la conseguir adaptar las comunicaciones 802.11 existentes hasta la fecha a las comunicaciones V2X. De forma que se consiguiese reducir la latencia al mínimo posible en el intercambio de mensajes.

Para ello veamos primero cuáles son los problemas que se pretenden evitar en estas redes: (6)

3.2.1 BSS

La norma 802.11 define una BSS (Basic Service Set) como una red compuesta por una serie de nodos (STAs) coordinados por otro nodo llamado AP, como se puede apreciar en la figura:

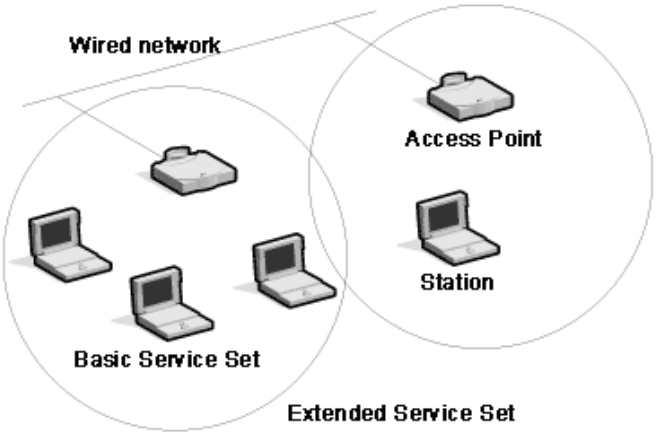


Figura 3-5. Red BSS

En una BSS los STAs esperan a recibir tramas Beacon (tramas lanzadas por los APs con toda la información que un STA necesita para poder iniciar el proceso de conexión), ya sea de forma pasiva o activa. Tras recibirla establecen una conexión con el AP después de una serie de pasos, entre los que se incluye la autenticación y la asociación. Una vez autenticados podrán empezar a lanzar mensajes a la red, siempre con una "autorización del AP".

Toda esta fase de autenticación lleva consigo un retardo inicial que puede llegar a ser crítico en las comunicaciones V2X.

La norma también define otro tipo de red que permite la comunicación fuera de una BSS, las redes "ad-hoc" en la que no existe un AP que coordine el intercambio de mensajes. Sin embargo, estas redes aún requieren de una complejidad y una latencia que no puede ser tolerada en las redes V2X.

3.2.2 BSSID

Otro concepto importante en las comunicaciones inalámbricas es el campo BSSID. Éste es un campo a nivel MAC de 48 bits de longitud que identifica de manera única a cada BSS (usando, por ejemplo, la dirección MAC del AP). Con este campo los nodos de una red 802.11 pueden filtrar fácilmente las tramas que pertenecen a sus BSS y las que no. La utilidad de este campo es obvia, pero al no existir un proceso de asociación con un AP en las redes 802.11p, necesita ser definido previamente para que los dispositivos nivel MAC puedan identificar los mensajes que han sido enviados. (6)

La solución a este problema es el uso de un valor del BSSID especial, el llamado wildcard BSSID, el cual está compuesto por todo "1s", algo así como las direcciones de difusiones IP donde todos los bits se establecen a 1.

3.2.3 Soluciones 802.11p

Para poder solucionar los problemas citados anteriormente, 802.11p introduce los siguientes cambios a nivel MAC: (8)

- Se permite el intercambio de mensajes fuera del contexto de las BSS, en una especie de red "ad-hoc" pero más simplificada. De esta forma se eliminan las latencias de la fase de asociación y autenticación del AP y se permite el intercambio de información útil desde el primer momento. Cabe destacar que es aquí donde nos encontramos el cambio más importante de la norma, y aunque existen otros con el objetivo de reducir latencias, éste es sin duda el destacado. Por supuesto, sin esta fase de conexión previa se eliminan muchas prestaciones importantes del nivel MAC, como por ejemplo la encriptación de los mensajes. Para dar soluciones a estos problemas, habrá que tratarlos en capas superiores.

- Además, se permite el uso de los wildcard BSSID para que, como hemos visto anteriormente, se tenga la seguridad de que la capa MAC receptora va a aceptar el mensaje. También es importante destacar que la norma 802.11 sólo permite el uso del wildcard BSSID en determinados casos, siendo uno de ellos el uso de 802.11p.

Con estos cambios a nivel MAC (vistos de forma superficial) se consigue el ahorro de unos segundos muy preciados en una comunicación donde el tiempo del que se dispone para el intercambio de mensajes puede ser muy reducido.

3.3 Capa PHY

La norma introduce pocos cambios en la capa PHY. Esto es debido a que una de las cláusulas de la norma 802.11 (la 802.11a) ya contempla las comunicaciones en la banda de los 5 GHz y es relativamente fácil adaptarlo a los 5.9 GHz. Además, realizar cambios en la capa PHY supone invertir muchos recursos ya que, mientras que realizar cambios en la capa MAC normalmente sólo requiere modificaciones a nivel software, cambios en la capa PHY suelen implicar variaciones a nivel electrónico (que normalmente requieren desarrollar nuevas tecnologías). Debido a esto, cuando se desarrolló la cláusula 802.11p se intentó modificar lo menos posible lo que ya estaba definido para la capa más baja del estándar OSI.

3.3.1 Canales de 10 MHz

Una de estas modificaciones fue pasar de los canales 20 MHz que nos encontrábamos, a canales de 10 MHz. De hecho 802.11 ya contempla los canales de 10 MHz, por lo que es una modificación sencilla ya que prácticamente solo implica doblar los parámetros temporales de las transmisiones 802.11a. (6)

La razón de este cambio es poder lidiar con el aumento en la dispersión del retardo (D_s) sufrida en las comunicaciones vehiculares. Dicho de otra forma: los cambios sufridos en la amplitud, la fase y los retardos que sufren las frecuencias no son constantes (aunque realmente solo lo son en el caso ideal) y empeoran con respecto a las comunicaciones 802.11a. Esta diferencia entre las distintas frecuencias se produce debido a que la respuesta impulsiva del canal no es constante.

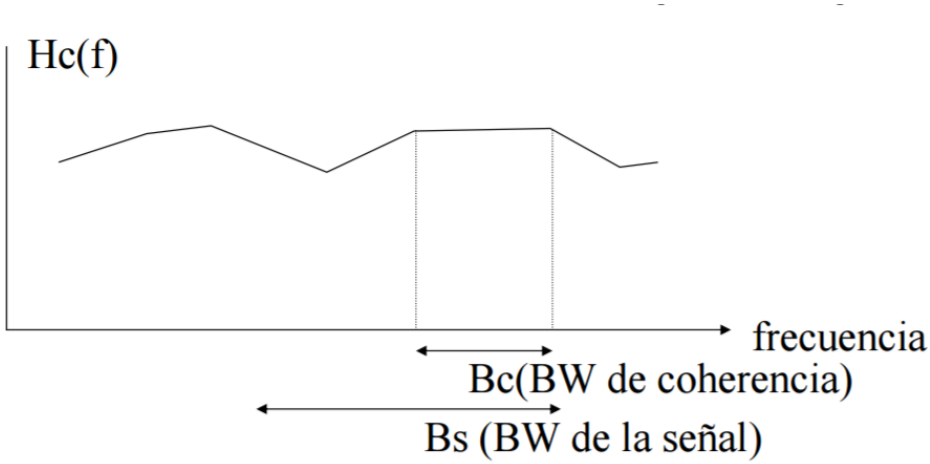


Figura 3-6. Respuesta impulsiva de un canal cualquiera

Lo que quiere decir que debemos reducir el rango de frecuencias (o el ancho de banda) para poder encontrar un subconjunto de ellas cuyos cambios se produzcan de una forma más o menos constante. Este ancho de banda es llamado el ancho de banda coherente (B_c) y está relacionado con la dispersión temporal de la siguiente forma:

$$B_c = \frac{1}{D_s}$$

La reducción del ancho de banda a la mitad hace que de los teóricos 6 a 54 Mbps que permite transmitir 802.11a pasemos a una tasa de transmisión de 3 a 27 Mbps.

3.3.2 Requisitos de rendimiento del receptor mejorados

Debido a la proximidad de los canales en las comunicaciones V2X y a la posible importancia de la información que se puede manejar, se ha de prestar especial atención a la interferencia que se puede producir entre distintos canales. (6)

Los problemas de interferencia entre canales son ampliamente conocidos en las comunicaciones inalámbricas y la solución más efectiva contra ellos es mediante políticas de gestión del canal que se escapan al alcance de la cláusula 802.11p. Aun así, se definen una serie de mejoras en los requisitos de recepción, existiendo de este modo:

- **Una categoría de tipo 1**: donde los niveles de decisión para aceptar un paquete a nivel físico son los ya definidos en 802.11. Esta categoría establece los requisitos mínimos.

  Estos niveles de decisión son: (8)

Modulación	Coding Rate (R)	Rechazo canal adyacente (dB)	Rechazo canal no adyacente (dB)
BPSK	½	16	32
BPSK	¾	15	31
QPSK	½	13	29
QPSK	¾	11	27
16-QAM	½	8	24
16-QAM	¾	4	20
64-QAM	⅔	0	16
64-QAM	¾	-1	15

Tabla 3-1. Niveles de decisión para 802.11

- **Una categoría de tipo 2**: donde se aumentan los niveles de decisión para el receptor. Esta categoría es opcional y por supuesto requiere de más costes económicos que la categoría 1.

  Estos niveles de decisión más restrictivos son: (8)

Modulación	Coding Rate (R)	Rechazo canal adyacente (dB)	Rechazo canal no adyacente (dB)
BPSK	½	28	42
BPSK	¾	27	41
QPSK	½	25	39
QPSK	¾	23	37
16-QAM	½	20	34
16-QAM	¾	16	30
64-QAM	⅔	12	26
64-QAM	¾	11	25

Tabla 3-2. Niveles de decisión más restrictivos para 802.11p

3.3.3 Niveles de potencia

Por último, añadir que existen una serie de limitaciones en potencia a la hora de transmitir que imponen los organismos reguladores de cada región: (8)

- Para EEUU se ha impuesto un límite de 48 dBm
- Para Europa el límite es de 33 dBm

3.4 Implementación en NS-3

Para permitir las simulaciones en entornos vehiculares NS-3 ha realizado una serie de modificaciones y ha añadido nuevos módulos en su código fuente.

Actualmente se encuentran desarrollados los módulos que implementan las capas MAC y PHY de la norma 802.11p. Aún se está trabajando para implementar en el software los protocolos superiores de la torre de protocolos WAVE

3.4.1 Capa MAC

Para implementar la capa MAC se hacen uso de tres clases:

- ns3::OcbWifiMac
- ns3::OrganizationIdentifier
- ns3::VendorSpecificActionHeader

De las cuales merece la pena comentar brevemente la ns3::OcbWifiMac

3.4.1.1 ns3::OcbWifiMac

Esta clase en NS-3 es la que permite las comunicaciones fuera del contexto de una BSS (OCB son las siglas de "outside of the context of a BSS").

En esta clase no existen las asociaciones, desasociaciones, autenticaciones y sincronizaciones necesarias en otros estándares WiFi, con objeto de salvar los problemas que las comunicaciones V2X conllevan, como ya hemos comentado.

3.4.2 Capa PHY

Puesto que la capa PHY de 802.11p está basada en la capa PHY de 802.11a y además apenas se realizan cambios, NS-3 no ha realizado ni modificado ningún módulo. En su lugar, cuando se defina un nodo en la aplicación y se le asocie un dispositivo de nivel físico, a éste se le establecerá una modulación OFDM (la usada en 802.11) con 10 MHz de anchura de canal.

Respecto a los niveles de potencia de transmisión no se ha implementado nada tampoco. Si quisiéramos cambiarlos (como en este caso) habría que ajustarlo manualmente como veremos más adelante.

4 CODIFICACIÓN Y VALIDACIÓN DEL SISTEMA

4.1 Consideraciones sobre la codificación

Para poder entender bien las decisiones que se han tomado a la hora de codificar y porqué se han realizado las simplificaciones que se han realizado, es necesario explicar qué clase de entorno queremos reproducir.

4.1.1 El entorno de la simulación

Es importante comentar que en nuestros experimentos queremos estudiar un escenario en el que el entorno se encuentre en el mejor caso posible, es decir, sin interferencias.

Estudiar un entorno en el caso que menos perjudica la comunicación es útil para plantearnos cuál es el valor máximo o mínimo que puede tomar el parámetro que deseamos medir. En nuestro caso nos interesará saber cuál es la tasa efectiva máxima que podemos generar en la aplicación.

Si por ejemplo obtenemos como resultado que (en el mejor de los casos) la tasa eficaz máxima es de 5 Mbps, sabremos con total seguridad que en ningún otro caso seremos capaces de transmitir más rápido a nivel de aplicación. Este límite superior habrá que tenerlo en cuenta a la hora de diseñar la aplicación.

4.2 Planteamiento del escenario

A continuación, vamos a detallar el planteamiento de los escenarios sobre los que vamos a realizar las simulaciones, sus simplificaciones y qué pretendemos estudiar en ellos.

4.2.1 Escenario 1

Como ya vimos en la introducción de esta memoria, en el escenario 1 se pretende observar el comportamiento de una comunicación basada en 802.11p cuando sólo existe un vehículo emitiendo y varias balizas recibiendo.

En este escenario el vehículo se estará desplazando con un velocidad constante v_x a lo largo del eje X, es decir el vector velocidad del vehículo será:

$$\vec{v} = (v_x, 0, 0)$$

Como hemos comentado nos interesa estudiar el escenario en el mejor de lo casos, para ello:

- Supondremos que sólo existe un vehículo en el entorno, o lo que es lo mismo, no se producirán interferencias en la señal transmisora que puedan producir un empeoramiento de las prestaciones del canal.

- La capa física transmitirá a la máxima potencia permitida en Europa para las comunicaciones 802.11p: 33 dBm. De todas formas, se dejará parametrizado en el código por si en un futuro se deseara simular con otro valor.

Regula-tory class	Channel starting frequency (GHz)	Channel spacing (MHz)	Channel set	Transmit power limit (mW)	Transmit power limit (EIRP)	Emissions limits set	Behavior limits set
13[a]	5.0025	5	171–184	—	33 dBm	7	17, 18
14[a, b]	5	10	171–184	—	33 dBm	7	17, 18
15[a, b]	5	20	172–183	—	23 dBm	7	17, 18
16	5	20	100, 104, 108, 112, 116, 120, 124, 128, 132, 136, 140	—	30 dBm	7	1, 3, 4, 17, 18
~~16~~17–255	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved

[a]This regulatory class specifies a list of channels in the 5.9 GHz band. Current regulations may only permit a subset of these channels.

Tabla 4-1. Potencias de transmisión máximas según el ancho de banda del canal

Por otro lado, colocaremos una serie de balizas a lo largo del eje X, equidistantes entre sí y a una altura z de unos 5 metros, para simular que están colocadas en semáforos (posible ubicación real de ellas). Estas balizas mantendrán una posición constante.

A la hora de establecer la distancia entre dos balizas consecutivas es importante tener en cuenta el rango máximo al que llega la antena transmisora (y). Si la distancia entre dos balizas es menor a $2y$ habrá un momento en el que el vehículo estará en una posición en la que sea capaz de transmitir correctamente a dos balizas a la vez. Esto es algo ineficiente en este tipo de comunicaciones ya que se prevé que las balizas serán capaces de compartir la información entre ellas, por lo que hacer que dos balizas reciban el mismo paquete de datos es innecesario. Además, mientras más juntas unas de las otras, más balizas necesitaremos para cubrir una misma distancia, lo que se traduce en más costes.

y depende de muchos factores, como pueden ser el tamaño del paquete, la modulación utilizada para transmitir, el modelo de pérdidas por propagación (del que hablaremos más adelante) o la potencia a la que transmita la capa física. Es por tanto muy difícil de calcular analíticamente, sobre todo porque estos parámetros pueden cambiar en el transcurso de una comunicación. En el código no habrá límites a la hora de establecer la distancia entre balizas, aunque sería recomendable hacer unas estimaciones de esta distancia y separar la balizas entre sí a un mínimo de $2y$.

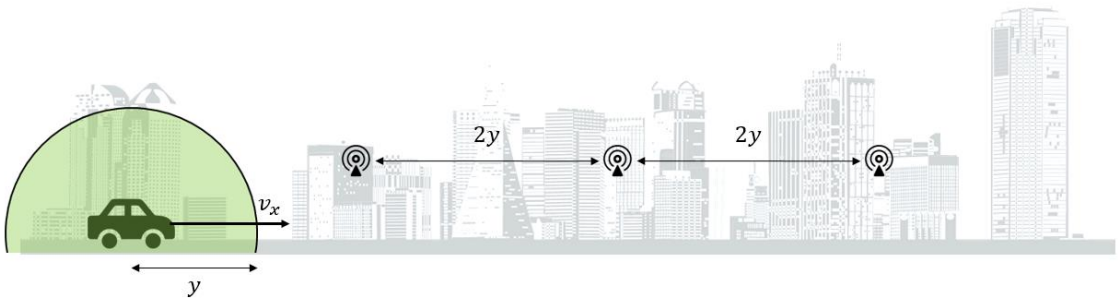


Figura 4-1. Esquema del escenario 1

4.2.2 Escenario 2

En el segundo escenario lo que vamos a plantear son dos coches aproximándose entre ellos cada uno con una velocidad propia, pero en sentidos contrarios.

Ambos vehículos transmitirán y recibirán información.

Nuevamente:

- Consideraremos que en el escenario sólo existen estos dos automóviles para que no se produzcan interferencias indeseadas.
- La potencia de transmisión de la capa física será de 33 dBm.

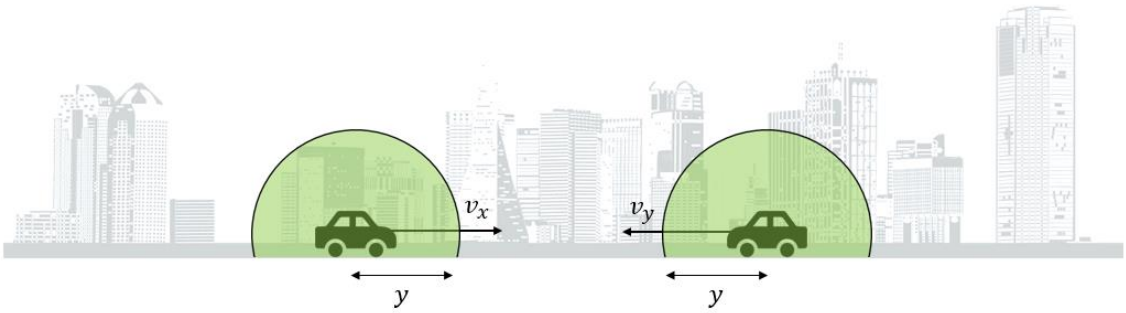


Figura 4-2. Esquema del escenario 2

El objetivo principal de este escenario es estudiar la comunicación bidireccional en los entornos V2X. Aquí también mediremos la tasa efectiva a nivel aplicación.

4.3 Capa PHY

En la capa PHY comentaremos los tres aspectos más importantes que van a influir en nuestra simulación:

1. El ancho de banda del canal elegido.
2. El modelo de error de propagación.
3. El modelo de pérdidas por propagación

4.3.1 Ancho de banda de 10 MHz

Sobre esto no hay mucho que comentar. Como explicamos en el capítulo 3, la capa PHY de 802.11p está basada en la capa PHY de 802.11a. Esta última permite las comunicaciones a 5, 10 y 20 MHz. Con la primera estaríamos renunciando a dos y cuatro veces la tasa de transmisión que se puede obtener usando 10 y 20 MHz respectivamente. Mientras que, como ya vimos, con 20 MHz la probabilidad de error de símbolo es demasiado alta en los entornos de comunicación V2X. Por lo tanto, el ancho de banda será de 10 MHz.

Otra razón añadida es que NS-3 no permite usar la clase ns3::OcbWifiMac con otro ancho de banda que no sea el de 10 MHz.

4.3.2 El modelo de error de propagación

Cuando en NS-3 la capa física recibe un paquete, el simulador recurre a una serie de estadísticos para determinar si el paquete ha llegado correctamente o con errores. Para ello tiene en cuenta las interferencias, el ruido del canal y las pérdidas. La primera se controla con la clase ns3::InterferenceHelper mientras que para las otras dos se recurre a un modelo.

Ya que nuestro escenario no tendrá interferencias, no nos pararemos en explicar cómo trata NS-3 las interferencias. Sí que es más interesante comentar brevemente el modelo elegido para los errores y las pérdidas de propagación.

4.3.2.1 NistErrorRateModel

Lo primero que hay que tener en cuenta en los modelos de errores debidos al ruido, es que NS-3 de momento solo añade a la recepción ruido gaussiano blanco (AWGN); cualquier posible efecto de desvanecimiento en la señal no se tiene en cuenta.

En NS-3 se ofrecen tres modelos para determinar la probabilidad de éxito de una recepción, estos son:

- ns3::DsssErrorRateModel
- ns3::NistErrorRateModel
- ns3::YansErrorRateModel

El primero sólo es usado cuando estamos usando la norma 802.11b por lo que no es interesante comentarlo. Los otros dos sí merece la pena pararse en ellos.

En NS-2 el modelo que se usaba por defecto era el YansErrorRateModel. Sin embargo, un estudio mostró claras diferencias entre este modelo de error y los resultados obtenidos sobre un canal sin interferencias. Es por esto que se desarrolló un nuevo modelo, el NistErrorRateModel. (9)

Los resultados con este nuevo modelo mejoraron significativamente y se convirtió por tanto en el usado por defecto en NS-3. Es además el de uso recomendado, por lo menos en el caso en el que no haya interferencias, y será por tanto el modelo que usemos.

A continuación, mostraremos algunas de las gráficas que presentó NS-3 para justificar la adición de este nuevo modelo, y que comparan los resultados obtenidos con el modelo Yans (en azul), el modelo Nist (en rojo) y las mediciones en entornos reales (en verde):

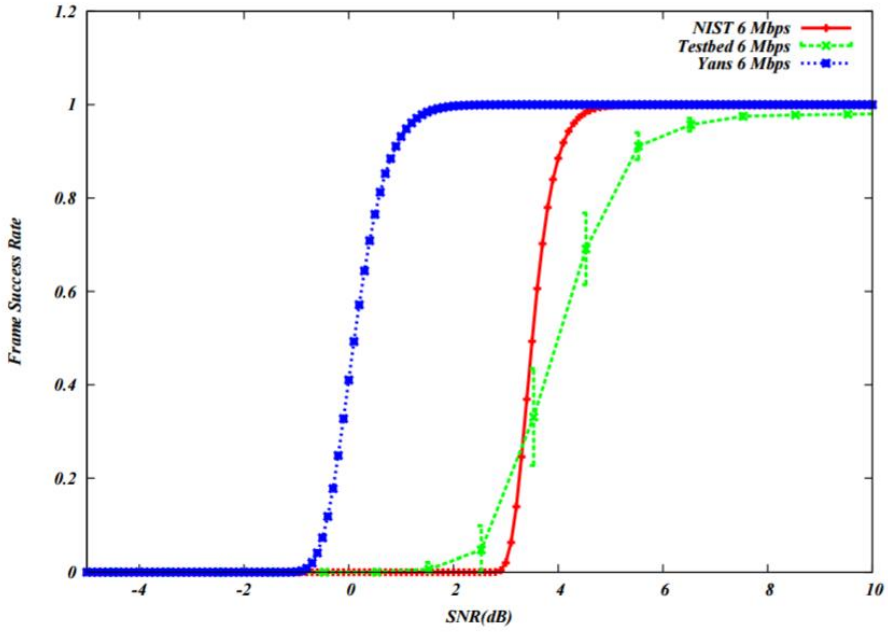


Figura 4-3. Comparación Nist-Yans para 6Mbps

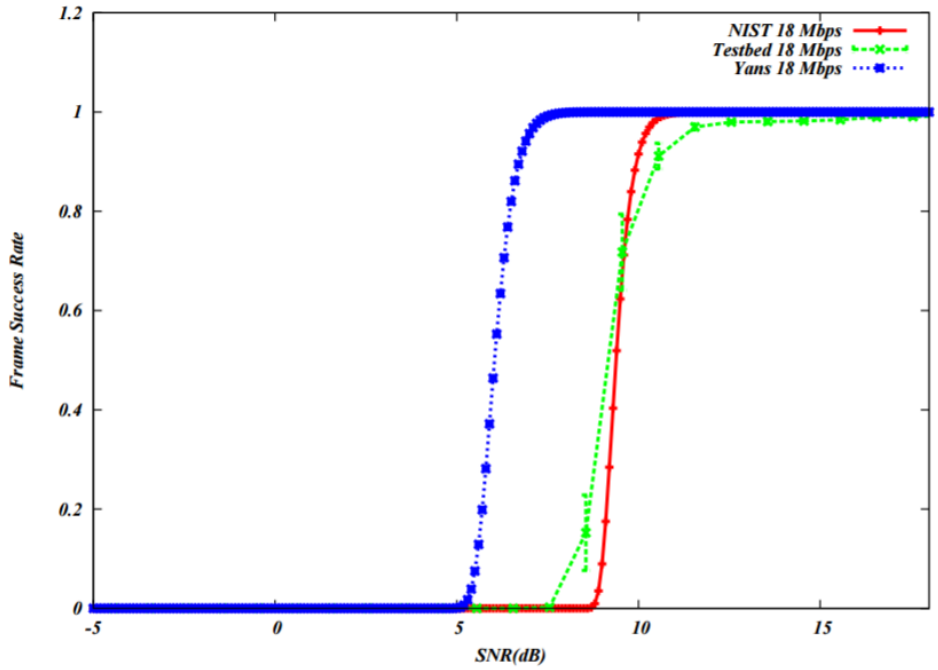


Figura 4-4. Comparación Nist-Yans para 18Mbps

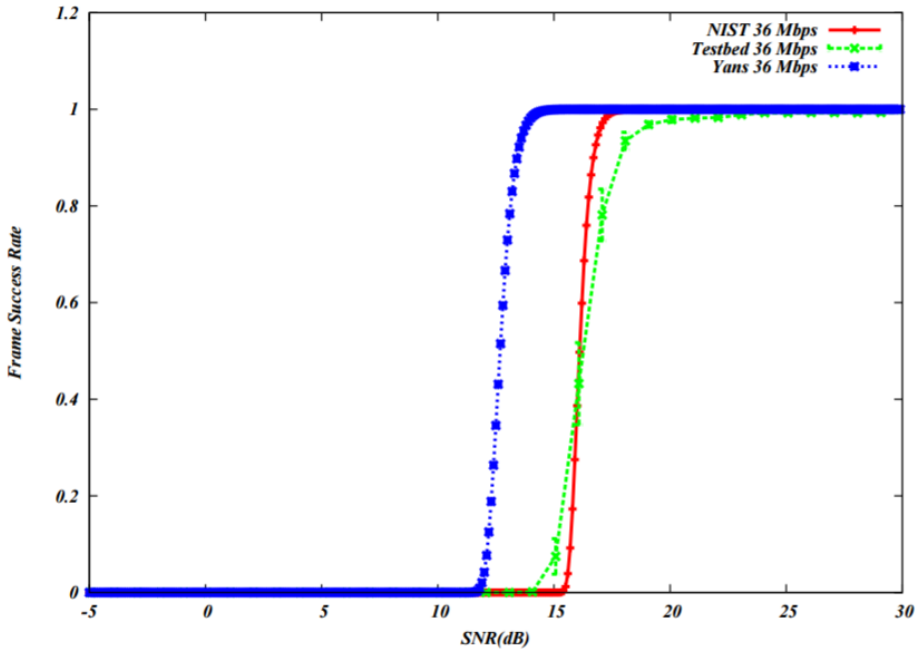


Figura 4-5. Comparación Nist-Yans para 36Mbps

4.3.2.2 El modelo de pérdidas por propagación

Elegir un modelo preciso para las pérdidas por propagación en los entornos de comunicaciones V2X no es una tarea sencilla. De hecho, no hay un gran consenso respecto a cuál ofrece los resultados más precisos. Por lo general cada estudio realiza sus propias mediciones y crea a partir de éstas un modelo distinto. Hay que tener en cuenta por tanto que éste puede ser un punto crítico a la hora de obtener precisión en los resultados, tanto en este trabajo como en futuros estudios.

En lo que sí hay un consenso en lo relativo al modelo de pérdidas por propagación es en que, el modelo de pérdidas logarítmicas usado normalmente, que asume perdidas exponenciales con respecto a la distancia, no es suficiente en estos entornos. Esto es debido principalmente a los desvanecimientos que puede sufrir la señal por el fenómeno conocido como multipath o multitrayecto, que ocurre por la existencia de obstáculos, como pueden ser lo edificios o los árboles.

El multitrayecto hace que una misma señal nos llegue por diferentes caminos (debido a las reflexiones, refracciones, etc. que sufren las señales) y que las contribuciones de cada una de ellas puedan sumarse o cancelarse, haciendo muy difícil estimar la potencia con la que nos va a llegar la señal (recordemos que el nivel físico en NS-3 calcula, siguiendo un modelo, la potencia del paquete recibido para determinar si lo acepta o no).

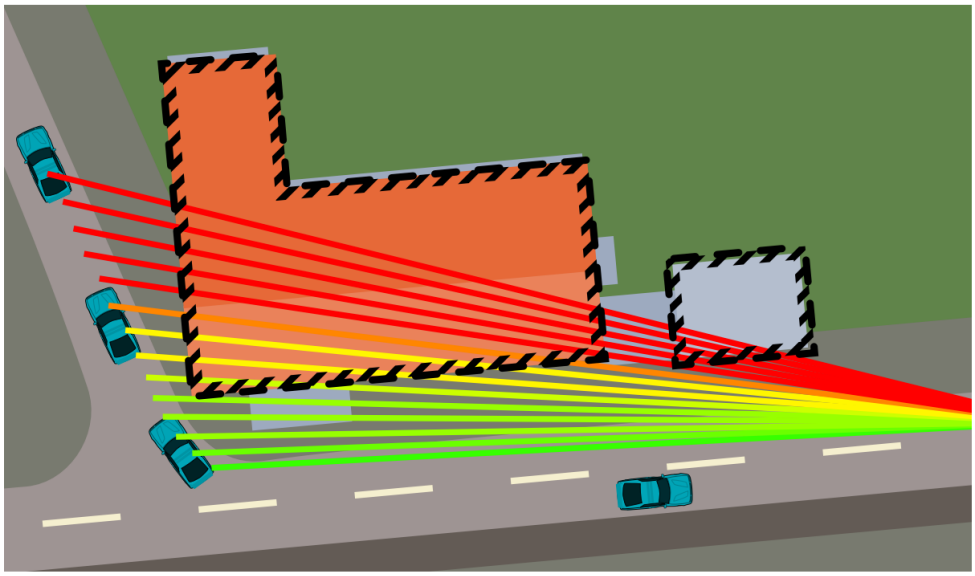


Figura 4-6. Representación del desvanecimiento de las señales en entornos V2X

Por tanto, para elegir nuestros modelos de pérdidas de propagación nos vamos a basar en las recomendaciones de NS-3 y en un estudio presentado por la Universidad Técnica de Delft, una de las universidades técnicas con más prestigio del mundo.

Por un lado, el estudio sugiere, para modelar estos desvanecimientos de la señal, usar la función de distribución de Nakagami (10), cuya fórmula es:

$$f_R(R) = \frac{2m^m R^{2m-1}}{\Gamma(m)\Omega^m} e^{-(m/\Omega)R^2}$$

Donde R es la amplitud del canal, Ω es la potencia media de desvanecimiento y Γ es la función Gamma.

m es el parámetro de desvanecimiento, que mide la gravedad de éste; a menor m mayor es el desvanecimiento. Para modelar m en función de la distancia el estudio utiliza la siguiente ecuación, basada en datos experimentales:

$$m = -0.69 Ln(distancia[m]) + 4.929$$

Si calculamos esta m desde 1 hasta 1000 metros (las distancias más grandes de nuestros experimentos serán de

unos 500 metros), obtenemos la siguiente gráfica:

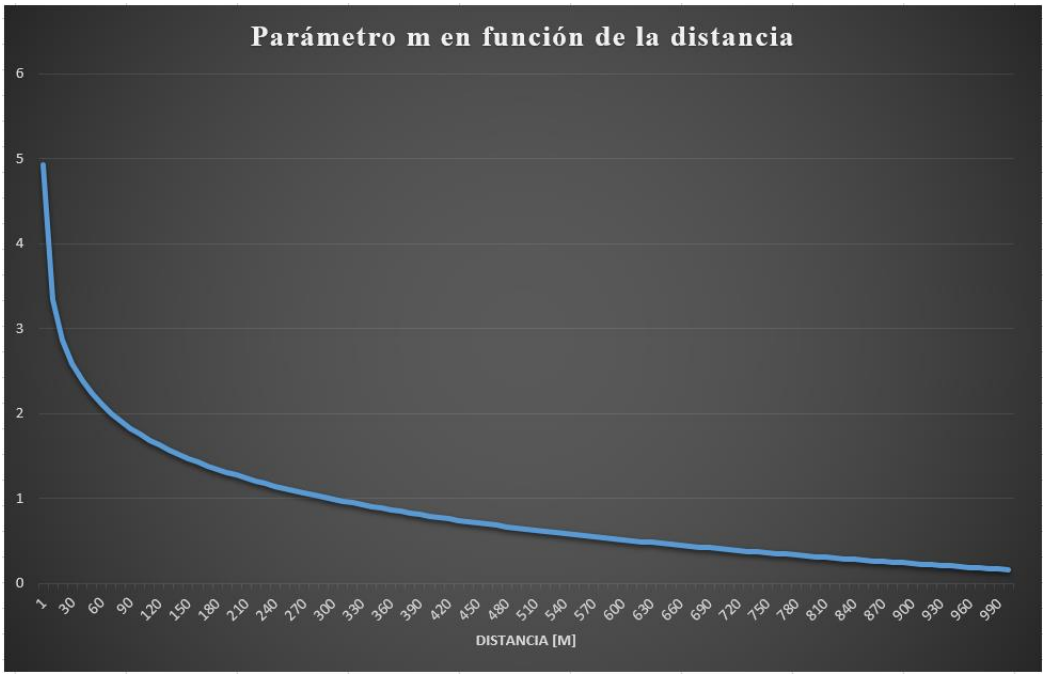


Figura 4-7. Parámetro m de Nakagami en función de la distancia

Además, si miramos la documentación de NS-3, ésta nos advierte que no cuentan con un modelo de pérdidas de propagación para las comunicaciones vehiculares, y una de las alternativas que nos sugiere es usar el modelo de Nakagami (que modela los desvanecimientos de las señales) y que sí está implementado.

En NS-3 este modelo toma unos valores fijos de m en función de la distancia, pudiendo definir hasta tres intervalos. (11)

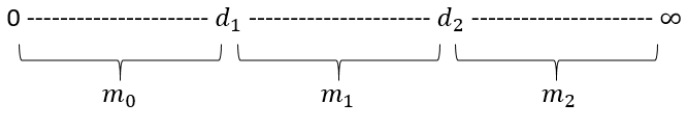


Figura 4-8. Valores del parámetro m de Nakagami en NS-3

Por tanto, para intentar replicar los valores de m que se presentan en el estudio anterior, se han definido los siguientes intervalos:

- De 0 a 70 metros una m de valor 2.92
- De 70 a 300 metros una m de valor 1.38
- De 300 en adelante una m de valor 0.5

Además, NS-3 aclara que este modelo de pérdidas de propagación no atenúa la señal con respecto a la distancia, por lo que recomienda usarlo en conjunto con otros modelos que sí implementen esta característica. Para esto, tomaremos el modelo por defecto en NS-3, que es además uno de los más usado, el LogDistancePropagationLossModel que considera perdidas exponenciales con respecto a la distancia. (12)

Recapitulando, en un intento de conseguir el modelo más preciso posible de pérdidas por propagación, usaremos el LogDistancePropagationLossModel para modelar las perdidas asociadas a la distancia, y el NakagamiPropagationLossModel para modelar los desvanecimientos producidos por los obstáculos.

4.4 Capa MAC

Un factor importante a decidir en la capa MAC es el algoritmo a usar para decidir la tasa de transmisión. NS-3 ofrece muchos de ellos, algunos basados en implementaciones reales, y otros sólo encontrados en la literatura. Mencionaremos tres de ellos: (13)

- **MinstrelWifiManager**. Éste es el algoritmo implementado actualmente por defecto en Linux. Minstrel mantiene un seguimiento de la probabilidad de enviar con éxito paquetes en cada tasa disponible. Tras esto multiplica esta probabilidad por la cantidad de datos que espera (para evitar la elección de tasa menores que suelen tener mayores probabilidades de éxito), y con esta información selecciona la tasa a la que transmitir.
- **ArfWifiManager**. El algoritmo por defecto de NS-3. Este algoritmo aumenta la tasa de transmisión cada vez que n paquetes son enviados correctamente. Cuando se pierden dos paquetes consecutivos, se reduce la tasa y se inicia un temporizador que tendrá que vencer antes de poder volver a incrementarla.
- **ConstantRateWifiManager**. En este caso se utilizará siempre la misma tasa de transmisión. Sin embargo, la norma 802.11 especifica claramente una serie de tasas para enviar las respuestas. Esto puede hacer que la tasa con la que se envíe un paquete de respuesta sea distinta a la tasa con la que se envíe un paquete de datos. Sin embargo, esto no es algo a evitar ya que se especifica en la norma.

Resultará interesante comparar estos tres algoritmos, pues mientras algunos consumen un tiempo extra en buscar la manera más rápida de transmitir, otros se ahorran ese tiempo a costa de renunciar a transmitir a la mejor tasa posible. Como este tiempo puede llegar a ser crítico en las comunicaciones V2X, realizaremos experimentos para comprobar cuál de ellos es más idóneo.

4.5 Capa de red y superiores

En la capa de red sólo hemos tenido en cuenta una consideración. Las direcciones IP destinos deben ser de difusión. Esto es debido a que en un escenario real no vamos a saber de antemano a qué dirección de red dirigirnos. Además, casi con toda seguridad necesitaremos transmitir a más de un nodo a lo largo del recorrido de un vehículo en un trayecto habitual. Es por esto que saber la dirección IP de los nodos a los que vamos a transmitir se hace del todo imposible, por lo que será necesario enviar a una dirección de difusión y que en capas superiores se decida qué hacer con cada paquete.

Sin embargo, debido a que nos hemos encontrado con algunos problemas al usar los protocolos de tasa adaptativa con direcciones de difusión (se verá más detalladamente en el capítulo 5), para el escenario 1 (cuando solo haya una baliza) y el escenario 2 utilizaremos direcciones IP unicast.

Para la capa de transporte hemos elegido el protocolo UDP. Las razones son las mismas que hay detrás de cualquier aplicación que usa UDP, evitar posibles retardos en las comunicaciones y aprovechar la poca carga adicional que incluye. En comparación con otros protocolos como TCP (que es el más conocido, aunque también existen otros como DCCP), UDP ofrece muchas menos prestaciones y por lo tanto el tiempo adicional que incorpora a la comunicación es muchísimo menor que el de otros protocolos.

Por último, hemos definido las aplicaciones que van a usar tantos los nodos transmisores como los receptores:

- Para los nodos transmisores hemos usado la aplicación OnOff definida en NS-3. Estas aplicaciones mandan paquetes de un tamaño configurable a tasas también configurables durante un tiempo “tOn” para después pasar a un estado en el que no transmiten nada durante un tiempo “tOff”. Sin embargo, a nosotros nos interesa exprimir las capacidades del canal al máximo, por lo que hemos configurado este tiempo “tOff” a 0 de forma que la aplicación nunca esté apagada. Entonces ¿Por qué no hemos usado otra aplicación que este constantemente enviando? Porque bajo las condiciones que se necesitaban en nuestros escenarios (protocolo UDP, envío constante y una tasa de transmisión regulable) la aplicación OnOff era la única que genera trazas. Es decir, esta aplicación es la única que te avisaba de cuándo envía un paquete, lo que simplifica bastante la tarea de medir parámetros a posteriori, y salvo por unas configuraciones extras, al final se comporta como una aplicación que

siempre está transmitiendo. Además, con la única intención de minimizar el tiempo de simulación real, se ha establecido que la aplicación siempre transmita a una tasa de 1Mbps mayor que la capa física. Con ello aseguraremos que la capa física siempre tiene algo que transmitir y no llenaremos la cola de eventos de forma ineficiente.

- Para los nodos que reciban hemos configurado aplicaciones UdpServer definidas en NS-3. Estas aplicaciones son muy simples. Lo único que hacen es escuchar y esperar para recibir paquetes UDP. Permiten también trazar estas llegadas de paquetes.

4.6 Resultados

Como hemos comentado anteriormente, lo que nos interesa medir en nuestras simulaciones es la tasa real de transmisión (Mbps) a nivel aplicación que podemos obtener. Es decir, cuántos bytes útiles hemos sido capaces de mandar satisfactoriamente en todo el tiempo que hemos estado dentro de un trayecto definido.

Para ello ha sido muy importante el uso de las trazas de NS-3. Sin éstas hubiese sido mucho más complejo averiguar cuándo se estaban produciendo eventos de relevancia en la simulación (de ahí que sea una de las características más importantes del simulador). En concreto las trazas a las que hemos prestado atención son las siguientes:

- El envío de un paquete a nivel aplicación.
- La recepción correcta de un paquete a nivel físico.
- El descarte de un paquete a nivel físico.
- La recepción de un paquete a nivel aplicación.

Con la primera y la última controlamos qué paquetes se están enviando, de qué tamaño y cuántas veces se están recibiendo.

Con el segundo y el tercero somos capaces de controlar cuándo el vehículo está suficientemente cerca de la antena receptora para poder transmitir y cuando está demasiado lejos para poder volver a transmitir un paquete correctamente. Es decir, cuándo se entra y se sale de la zona de cobertura.

Además, se han dejado codificadas las acciones ante otra serie de trazas por si interesase comprobar en otros estudios qué está pasando entre estas trazas que sí estamos controlando. Estas acciones de momento son simplemente imprimir por logs cuándo se producen estas llamadas.

Veamos ahora cómo hemos recogido los datos necesarios en los dos escenarios.

El escenario 1 se puede plantear de varias maneras:

1. Con una sola baliza. Interesante para medir cuáles son los valores que alcanza la tasa eficaz como mínimo.

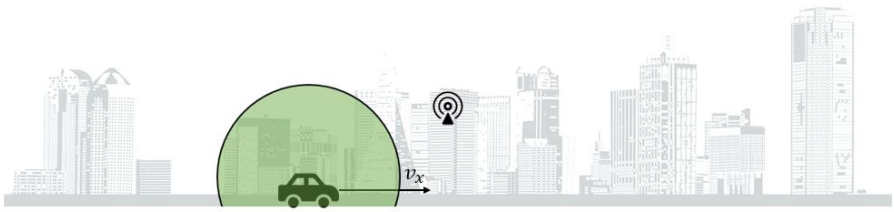


Figura 4-9. Escenario 1, Planteamiento 1

2. Con varias balizas separadas entre sí una distancia x de manera que dos balizas nunca van a recibir una misma trama a la vez. Interesante para comprobar cuánto podemos alejar las balizas en un trayecto sin llegar a bajar la tasa eficaz de unos umbrales mínimos.

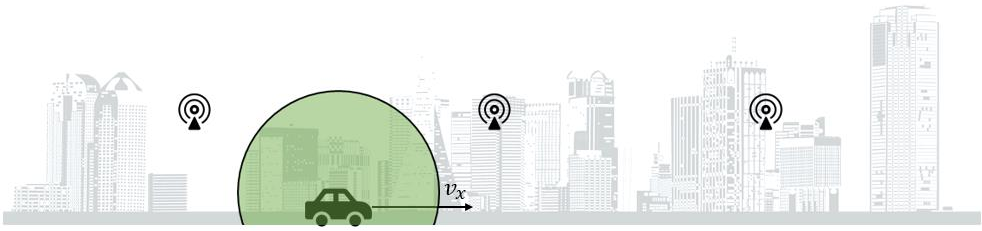


Figura 4-10. Escenario 1, Planteamiento 2

3. Con varias balizas separadas entre sí a una distancia x de manera que dos balizas sí puedan llegar a recibir la misma trama prácticamente a la vez. En principio no es interesante hacer experimentos sobre esta condición, pero hay que controlarlo para que no nos falsee los datos en caso de que ocurra de una forma imprevista, ya sea porque se cambie el tamaño del paquete, la tasa de transmisión o cualquier otro parámetro que influya en el alcance.

Figura 4-11. Escenario 1, Planteamiento 3

Para garantizar la solidez de los datos en cualquiera de estas tres condiciones estamos controlando los dos parámetros de interés de la siguiente manera:

- **Bytes útiles:** En el momento en el que la aplicación transmita un paquete, se guarda en una estructura bidimensional que almacena un identificador único por paquete y el tamaño de paquete. Cuando llegue a la capa de aplicación del receptor, se mirará el identificador del paquete, se buscará en la estructura y, si se encuentra, se sumará el tamaño del paquete a los bytes útiles transmitidos. Por último, se eliminará de la estructura donde fue guardado. Esto se hace para que, si un paquete se recibe en dos aplicaciones, la primera de ellas lo cuente como datos útiles, no así la segunda, lo que tiene sentido pues estaríamos transmitiendo la misma información dos veces.
- **Tiempo de transmisión:** Para contar el tiempo que estamos transmitiendo medimos el tiempo en el que entramos en la zona de cobertura (tiempo inicial) y el tiempo en el que salimos (tiempo final) y calculamos la diferencia.

  Se considera el tiempo inicial aquél en el que seamos capaces de transmitir correctamente por primera vez.

  El tiempo final se considera aquél en el que se cumplan estas dos condiciones:

  1. Se ha recorrido una distancia igual a: el número de balizas menos uno por la distancia de separación entre ellas, más el rango de cobertura que tengan las antenas en esta simulación. Este rango se mide la primera vez que transmitimos correctamente, comprobando la distancia entre el vehículo y la baliza. Es decir:

     $$d * (nBalizas - 1) + rango$$

     Donde d es la distancia entre las balizas, nBalizas es el número de balizas de la simulación y rango es la separación máxima que puede haber entre una baliza y el vehículo para que las tramas se reciban correctamente.

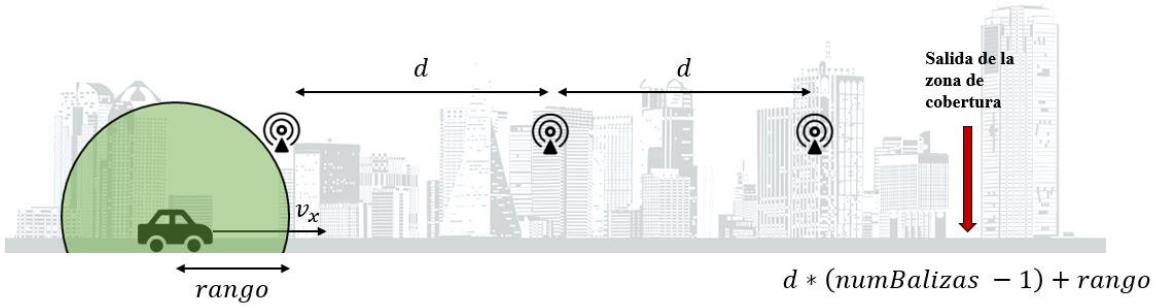


Figura 4-12. Distancia hasta salir de la zona de cobertura

2. Se lleva más de medio segundo sin recibir paquetes correctamente. Esto se hace porque en el caso de que se comience la simulación dentro de la zona de cobertura de la antena el rango no se va a medir bien (va a ser más pequeño) y la primera condición se cumplirá antes de lo previsto. Es por esto por lo que se esperará medio segundo sin recibir nada (que es mucho tiempo en comparación con las velocidades a las que se transmite) para deducir que hemos salido de la zona de cobertura. La razón de elegir medio segundo es porque es tiempo suficiente para entender que ya no somos capaces de transmitir correctamente pero no es representativo en comparación al tiempo total simulado, y por lo tanto apenas influirá a la hora de presentar los resultados.

Para el segundo escenario los parámetros se han medido de la siguiente forma:

- **Bytes útiles:** Igual que para el escenario 1. En este caso hemos llevado la cuenta de los bytes de los dos vehículos por separado para presentar el resultado final como la media de los dos.
- **Tiempo de simulación:** Volvemos a establecer dos condiciones:
  1. Que el primer vehículo este en la posición en el eje de las X como mínimo de:

$$x_0 + \frac{x_0 - y_0 - rango}{v_x + v_y} * v_x$$

Donde x_0 es la posición del vehículo 1 cuando se entra en la zona de cobertura, y_0 es la posición del vehículo 2 cuando se entra en la zona de cobertura, $rango$ es la distancia entre los dos vehículos al entrar en la zona de cobertura, v_x es la velocidad del vehículo 1 y v_y es la velocidad del vehículo 2.

Esta ecuación viene de calcular en el siguiente escenario (cuando los vehículos entran en la zona de cobertura):

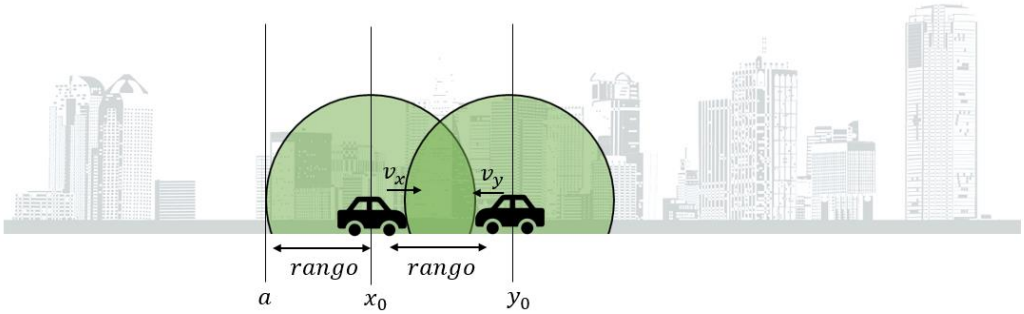


Figura 4-13. Posición inicial en el escenario 2

En qué posición deberán estar los vehículos para que ambos no puedan comunicarse más entre ellos (a es la posición más alejada a la que el vehículo 1 es capaz de transmitir como máximo).

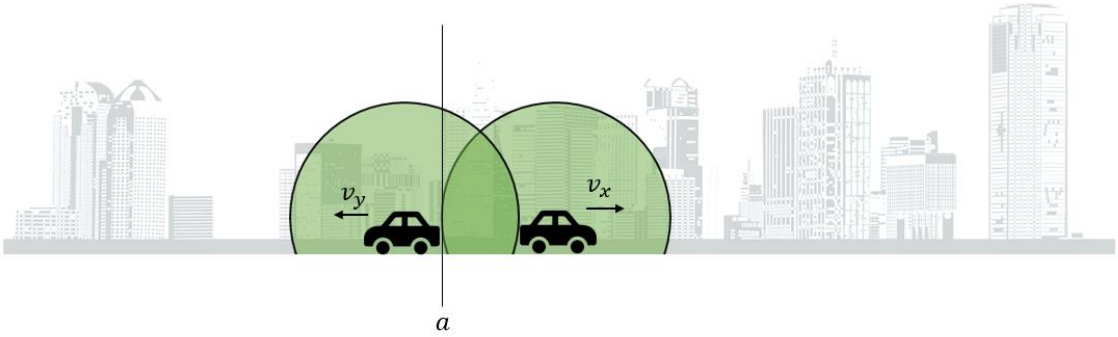


Figura 4-14. Posición final en el escenario 2

Para ello nos planteamos cuándo a y la posición del vehículo 2 serán iguales:

$$a = x_0 - rango + t * v_x$$

$$y = y_0 - t * v_y$$

Donde t es el tiempo e y es la posición del vehículo 2

Y si despejamos t y lo sustituimos en la ecuación que define la posición del vehículo 1:

$$x = x_0 + t * v_x$$

Llegamos a la fórmula inicial que nos la distancia que tiene que se tiene que recorrer.

2. Damos un margen de medio segundo por los mismos motivos comentados en el escenario 1.

5 ANÁLISIS DE RESULTADOS

Es débil porque no ha dudado bastante y ha querido llegar a conclusiones.

- Miguel de Unamuno -

5.1 Diseño de experimentos

Para poder estudiar el comportamiento de las comunicaciones V2X hemos diseñado los siguientes experimentos de interés, cuyos resultados estudiaremos para poder entender cómo se comportan este tipo de redes:

1. **Experimento 1.** *Análisis de la influencia de la tasa de transmisión y de la velocidad del vehículo.*

   Con el que pretendemos medir cuál es la tasa óptima para transmitir y si la velocidad del vehículo juega un papel importante. Para ello realizaremos simulaciones sobre el escenario 1 con una sola baliza, transmitiendo con todas las tasas posible y con los protocolos Minstrel y Arf, con velocidades desde los 20 km/h hasta los 120 km/h y con un tamaño de paquete a nivel de aplicación de 1500 Bytes.

2. **Experimento 2.** *Análisis del rendimiento de las comunicaciones en función de la distancia de separación de las balizas.*

   Donde se observará hasta cuánto se pueden separar las balizas entre ellas para poder mantener una tasa eficaz mínima necesaria para que una aplicación pueda funcionar satisfactoriamente. Para ello realizaremos simulaciones en el escenario 1 con cinco balizas separadas entre ellas una distancia desde los 800 metros hasta los 2 kilómetros, transmitiendo con todas las tasas posible y con los protocolos Minstrel y Arf, con una velocidad de 50 km/h para el vehículo y un tamaño de paquete a nivel de aplicación de 1500 Bytes.

3. **Experimento 3.** *Análisis de las comunicaciones V2X bidireccionales.*

   En el que se medirá el rendimiento de una comunicación vehicular en la que transmiten dos nodos. Para ello se harán simulaciones sobre el escenario 2, transmitiendo con todas las tasas posible y con los protocolos Minstrel y Arf, con velocidades desde los 20 km/h hasta los 120 km/h y con un tamaño de paquete a nivel de aplicación de 1500 Bytes.

4. **Experimento 4.** *Análisis de la influencia del tamaño del paquete.*

   Con el que se observará cómo influye el tamaño del paquete en los parámetros de la red que nos son de interés. Para ellos se simulará sobre el escenario 1 con una sola baliza, transmitiendo con todas las tasas posible y con los protocolos Minstrel y Arf, con una velocidad de 50 km/h para el vehículo y un tamaño de paquete que variará de 100 a 1500 Bytes.

5. **Experimento 5.** *Análisis del protocolo Minstrel.*

   En este experimento cambiaremos los parámetros del protocolo Minstrel para averiguar con cuáles podemos obtener el máximo rendimiento en redes vehiculares. Para ello cambiaremos la frecuencia con él que el protocolo escanea la red para averiguar cuál es la mejor tasa posible. Simularemos el

escenario 1 con una sola baliza, una velocidad de 50 km/h para el vehículo y un tamaño de paquete a nivel de aplicación de 1500 Bytes.

6. **Experimento 6.** *Análisis del protocolo Arf.*

   Donde se cambiarán los parámetros del protocolo Arf para averiguar con cuáles podemos obtener el máximo rendimiento en redes vehiculares. Para ello cambiaremos el número de paquetes correctos necesarios para que se aumente de tasa. Simularemos en el escenario 1 con una sola baliza, una velocidad de 50 km/h para el vehículo y un tamaño de paquete a nivel de aplicación de 1500 Bytes.

7. **Experimento 7.** *Análisis de la influencia de la altura de la antena*

   Con el que se observará cuál es la influencia de la altura de la antena en las redes vehiculares. Para ello variaremos la altura de la antena de 1 a 10 metros. Se simulará sobre el escenario 1 con una sola baliza, transmitiendo con todas las tasas posible y con los protocolos Minstrel y Arf, con una velocidad de 50 km/h para el vehículo y un tamaño a nivel de aplicación de 1500 Bytes.

5.2 Resultados y conclusiones

5.2.1 Experimento 1. Análisis de la influencia de la tasa de transmisión y de la velocidad del vehículo.

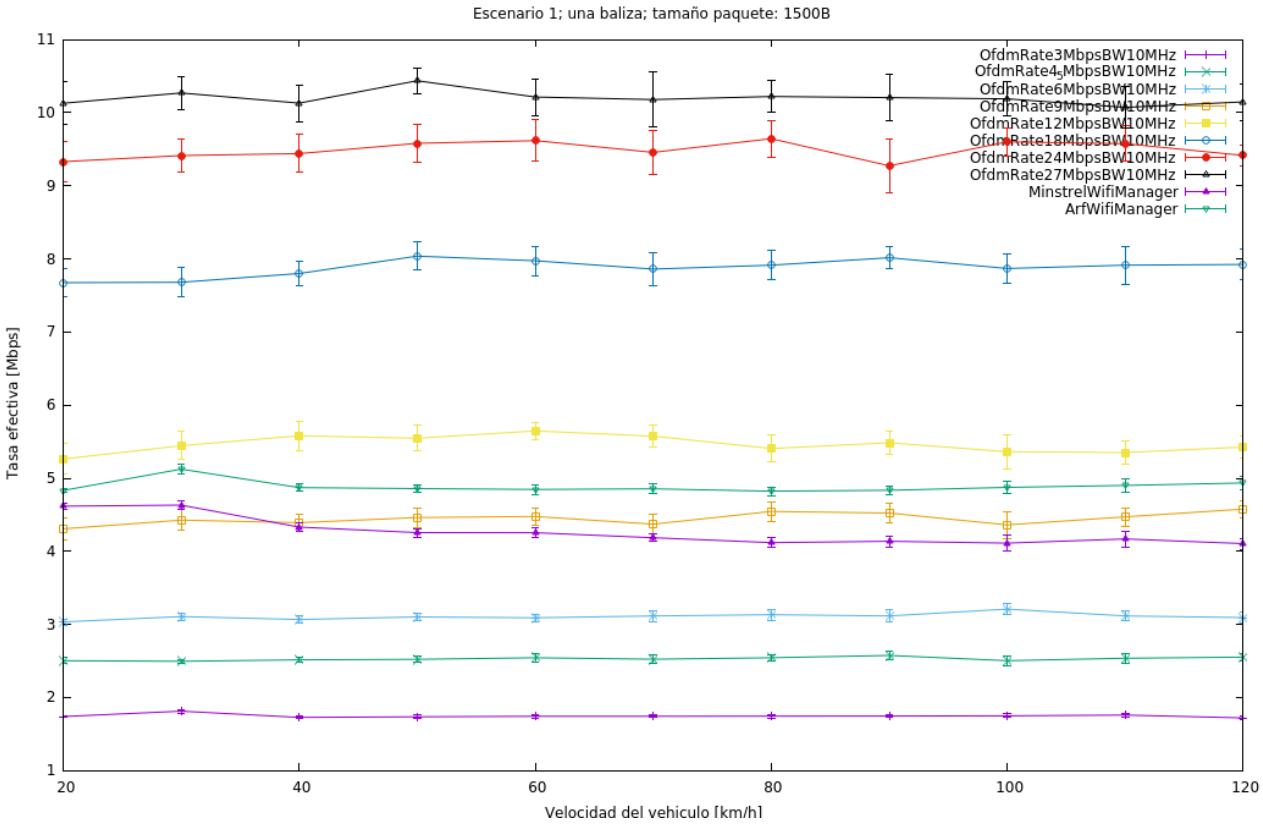


Figura 5-1. Experimento 1. Tasa efectiva

Velocidad (km/h)	20	30	40	50	60	70	80	90	100	110	120
Tasa de transmisión	Tasa Efectiva (Mbps)										
OfdmRate3MbpsBW10MHz	1,74	1,81	1,73	1,73	1,74	1,74	1,74	1,74	1,75	1,76	1,72
OfdmRate4_5MbpsBW10MHz	2,50	2,49	2,51	2,52	2,54	2,52	2,54	2,57	2,50	2,53	2,55
OfdmRate6MbpsBW10MHz	3,03	3,10	3,06	3,10	3,08	3,11	3,13	3,11	3,20	3,11	3,09
OfdmRate9MbpsBW10MHz	4,30	4,42	4,39	4,46	4,47	4,37	4,54	4,52	4,36	4,47	4,57
OfdmRate12MbpsBW10MHz	5,26	5,44	5,57	5,54	5,64	5,57	5,40	5,48	5,36	5,35	5,42
OfdmRate18MbpsBW10MHz	7,67	7,68	7,79	8,03	7,97	7,85	7,91	8,01	7,86	7,91	7,92
OfdmRate24MbpsBW10MHz	9,33	9,41	9,44	9,58	9,61	9,45	9,64	9,27	9,60	9,57	9,41
OfdmRate27MbpsBW10MHz	10,12	10,27	10,13	10,43	10,21	10,17	10,22	10,20	10,18	10,07	10,14
MinstrelWifiManager	4,61	4,63	4,33	4,25	4,25	4,18	4,11	4,13	4,11	4,16	4,10
ArfWifiManager	4,83	5,12	4,87	4,85	4,84	4,85	4,82	4,83	4,87	4,90	4,93

Tabla 5-1. Experimento 1. Tasa efectiva

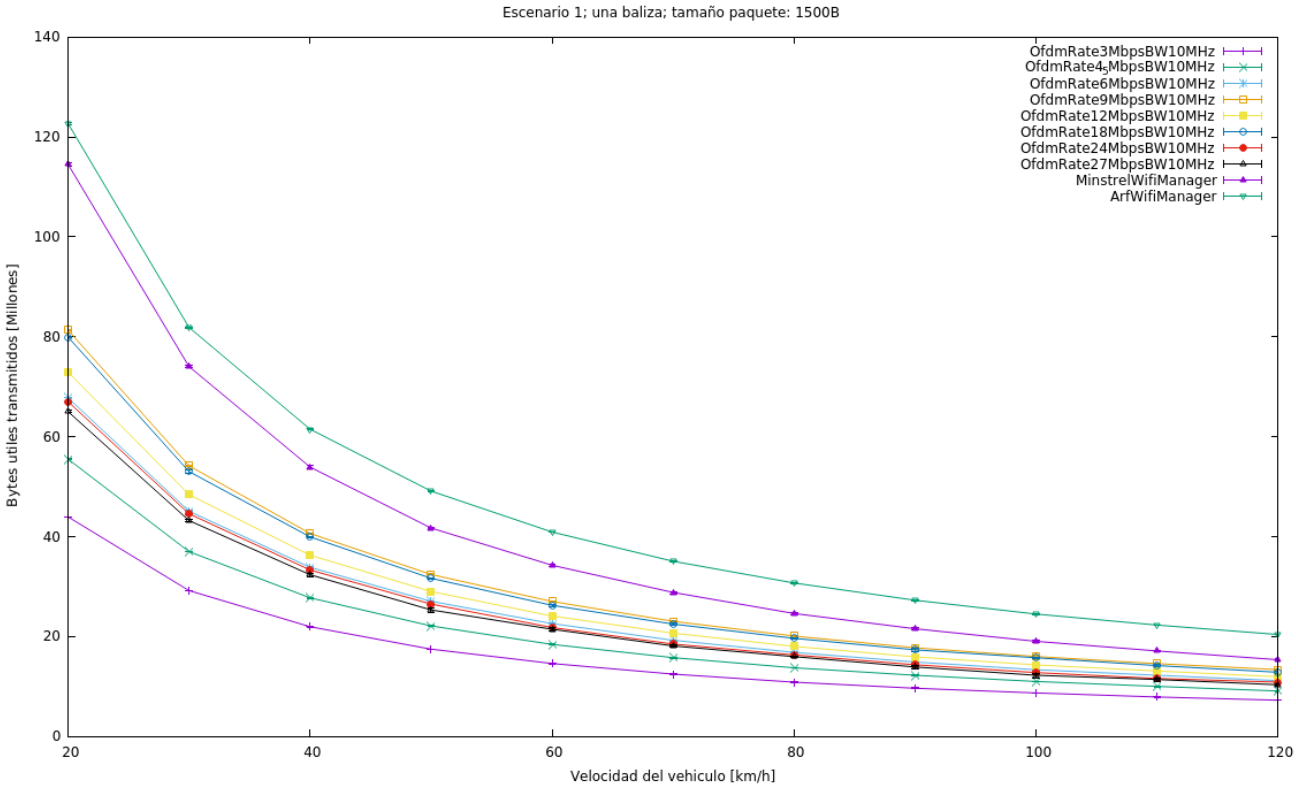


Figura 5-2. Experimento 1. Bytes útiles

Velocidad (km/h)	20	30	40	50	60	70	80	90	100	110	120
Tasa de transmisión	Bytes útiles (Millones)										
OfdmRate3MbpsBW10MHz	43,94	29,19	21,95	17,47	14,58	12,47	10,87	9,65	8,70	7,89	7,25
OfdmRate4_5MbpsBW10MHz	55,50	37,00	27,73	22,13	18,42	15,76	13,75	12,24	11,00	10,00	9,09
OfdmRate6MbpsBW10MHz	67,77	45,10	33,84	27,04	22,58	19,24	16,83	14,89	13,35	12,24	11,15
OfdmRate9MbpsBW10MHz	81,34	54,17	40,63	32,42	27,01	23,04	20,10	17,74	15,99	14,57	13,38
OfdmRate12MbpsBW10MHz	72,91	48,42	36,27	29,00	24,09	20,66	18,02	15,92	14,31	13,09	11,94
OfdmRate18MbpsBW10MHz	79,92	53,01	39,94	31,65	26,23	22,50	19,62	17,33	15,74	14,18	12,84
OfdmRate24MbpsBW10MHz	67,01	44,57	33,32	26,50	21,80	18,49	16,31	14,34	12,77	11,59	10,87
OfdmRate27MbpsBW10MHz	65,02	43,20	32,35	25,33	21,44	18,11	15,98	13,90	12,24	11,39	10,33
MinstrelWifiManager	114,52	74,03	53,93	41,71	34,25	28,81	24,61	21,55	19,02	17,10	15,35
ArfWifiManager	122,69	81,85	61,48	49,11	40,90	35,07	30,70	27,25	24,49	22,28	20,39

Tabla 5-2. Experimento 1. Bytes útiles

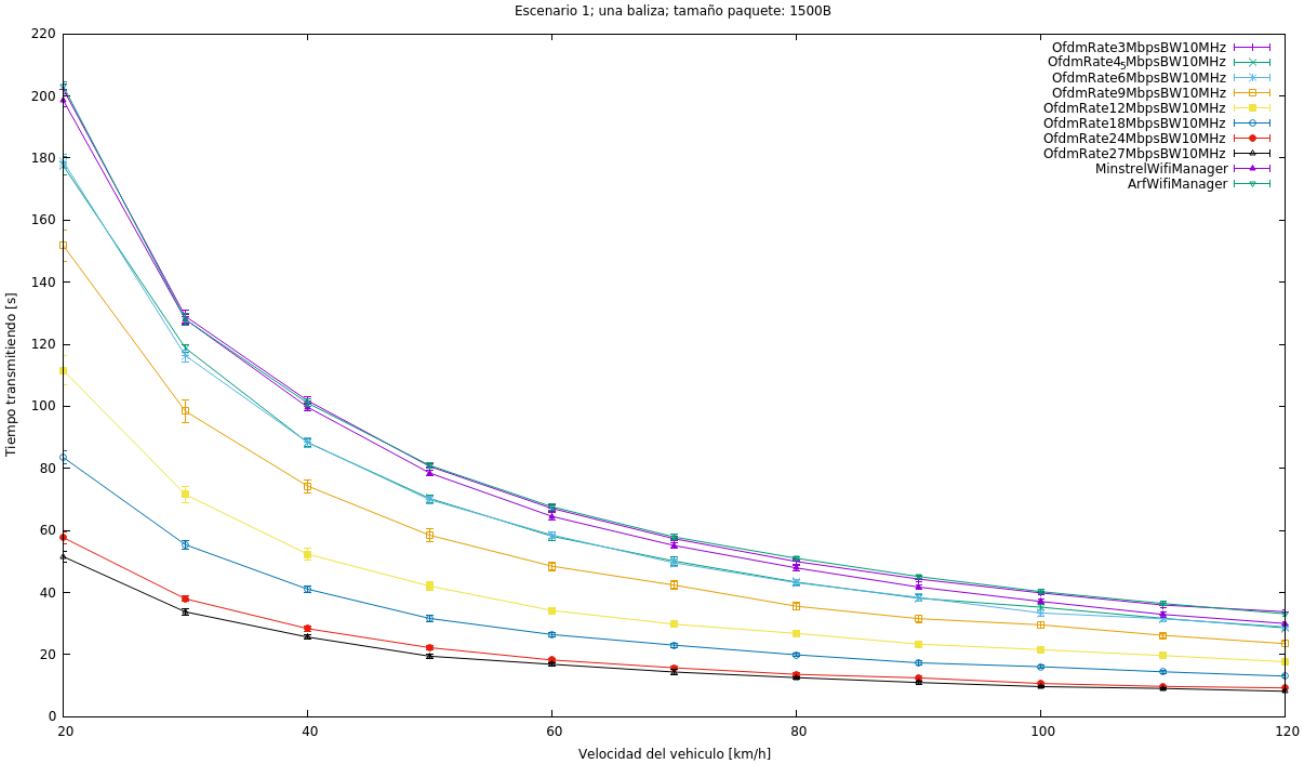


Figura 5-3. Experimento 1. Tiempo transmitiendo

Velocidad (km/h)	20	30	40	50	60	70	80	90	100	110	120
Tasa de transmisión	Tiempo transmitiendo (s)										
OfdmRate3MbpsBW10MHz	202,22	128,98	101,79	80,60	67,05	57,30	49,94	44,30	39,85	35,93	33,78
OfdmRate4_5MbpsBW10MHz	177,56	118,75	88,29	70,29	58,11	50,09	43,32	38,14	35,24	31,64	28,58
OfdmRate6MbpsBW10MHz	178,91	116,31	88,38	69,84	58,54	49,51	43,13	38,38	33,39	31,51	28,94
OfdmRate9MbpsBW10MHz	151,80	98,41	74,27	58,42	48,44	42,36	35,54	31,53	29,54	26,16	23,48
OfdmRate12MbpsBW10MHz	111,62	71,56	52,32	42,05	34,21	29,76	26,80	23,34	21,56	19,66	17,67
OfdmRate18MbpsBW10MHz	83,53	55,39	41,05	31,60	26,40	22,99	19,90	17,33	46,05	14,42	13,02
OfdmRate24MbpsBW10MHz	57,70	37,97	28,34	22,2	18,21	15,72	13,59	12,46	10,68	9,72	9,25
OfdmRate27MbpsBW10MHz	51,56	33,73	25,63	19,45	16,85	14,35	12,54	10,96	9,64	9,07	8,16
MinstrelWifiManager	198,57	127,99	99,69	78,47	64,50	55,12	47,89	41,73	37,09	32,90	29,98
ArfWifiManager	203,18	127,95	101,04	80,91	67,58	57,85	50,98	45,12	40,25	36,42	33,13

Tabla 5-3. Experimento 1. Tiempo transmitiendo

Si analizamos las gráficas presentadas podemos darnos cuenta de tres aspectos importantes:

1. Los protocolos de tasa adaptativa consiguen transmitir más bytes que los protocolos de tasa fija.
2. Arf obtiene mejores resultados que Minstrel.
3. Las tasas fijas más altas no son siempre mejores a pesar de transmitir más rápido.

Para poder explicar estos tres puntos ha sido necesario analizar los registros del programa y observar en la red qué iba ocurriendo paso por paso. Tras ello hemos comprendido el porqué de estos tres aspectos.

Por un lado, el primer punto es un resultado lógico ya que con los protocolos de tasa adaptativa combinamos las ventajas de usar tasas bajas (podemos transmitir durante más tiempo), y las de usar tasas altas (podemos transmitir más rápido), con lo que la suma de ambas características da un resultado superior al de las tasas fijas. Eso sí, la capa de aplicación tendrá que generar los datos más lentamente que si transmitiésemos siempre a tasas altas.

Además, si nos fijamos en los dos protocolos de tasa adaptativa con los que hemos trabajado, observamos como Arf obtiene mejores resultados. La razón es la agilidad de cada protocolo; mientras que Minstrel actualiza las estadísticas cada segundo (por defecto), Arf aumenta la tasa cada vez que recibe diez paquetes consecutivos correctos (que tardan en llegar mucho menos de un segundo). Por lo tanto Arf es capaz de detectar mucho antes cuándo se puede empezar a transmitir con tasas más altas, lo que le permite obtener mejores resultados en entornos vehiculares.

Por último, explicaremos por qué algunas tasas contantes funcionan mejor que otras que en teoría son más rápidas. El motivo se debe al tiempo medio que pasa entre dos transmisiones consecutivas. Por ejemplo, teniendo en cuenta que la tasa de 27 Mbps, que teóricamente permite transmitir tres veces más rápido que la de 9 Mbps, lo hace en una tercera parte de tiempo, podríamos pensar que al final la dos deberían transmitir más o menos lo mismo. Sin embargo, si obtenemos las estadísticas de cuánto tiempo pasa (en media) entre dos transmisiones consecutivas a nivel físico, nos encontramos con que la tasa más alta no transmite paquetes tres veces más rápido, sino un poco menos.

El porqué de esto se debe a los paquetes ACK que envía la baliza receptora. Según se especifica en la norma 802.11, las respuestas ACK no se transmiten con la misma tasa que los paquetes de datos, sino con unas de las tasas que cumpla una serie de condiciones impuestas por la norma. Esto hace que cuando usamos la tasa de 27 Mbps los ACKs se transmitan a 12 Mbps y si usamos la tasa de 9 Mbps los ACKs se transmites a 6Mbps.

Luego, aunque con 27 Mbps se generen paquetes tres veces más rápido y se tarden en transmitir tres veces menos, la espera a la recepción del ACK no es el triple de rápida, lo que explica que con 27 Mbps no seamos capaces de llegar a transmitir tres veces más rápido.

5.2.2 Experimento 2. Análisis del rendimiento de las comunicaciones en función de la distancia de separación de las balizas.

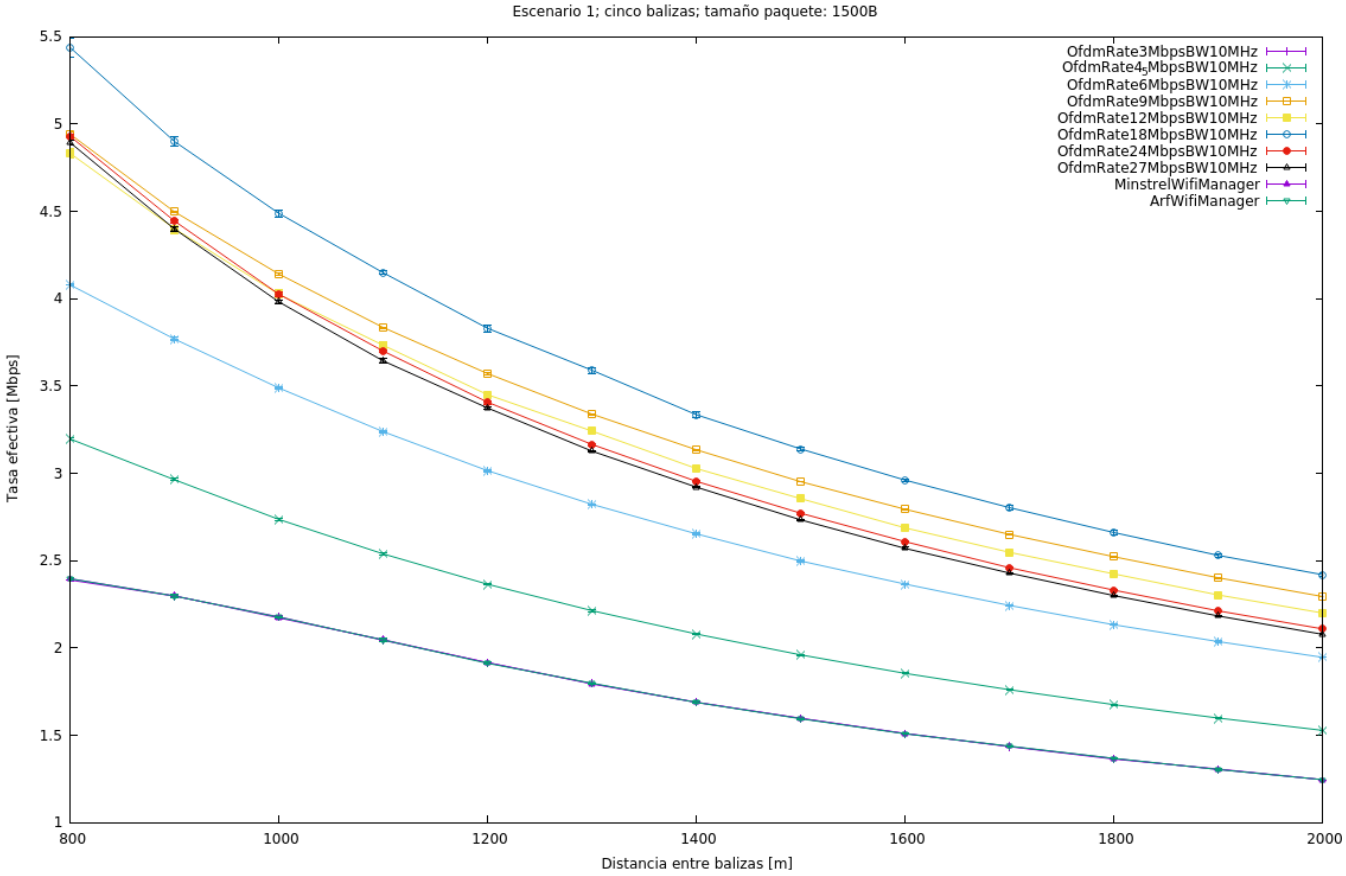


Figura 5-4. Experimento 2. Tasa Efectiva

Distancia entre balizas (m)	800	900	1000	1100	1200	1300	1400	1500	1600	1700	1800	1900	2000
Tasa de transmisión	Tasa Efectiva (Mbps)												
OfdmRate3MbpsBW10MHz	2,38	2,29	2,17	2,04	1,91	1,79	1,68	1,59	1,50	1,43	1,36	1,30	1,24
OfdmRate4_5MbpsBW10MHz	3,19	2,96	2,73	2,53	2,36	2,21	2,07	1,96	1,85	1,76	1,67	1,59	1,52
OfdmRate6MbpsBW10MHz	4,07	3,76	3,48	3,23	3,01	3,82	2,65	2,49	2,36	2,24	2,13	2,03	1,94
OfdmRate9MbpsBW10MHz	4,94	4,49	4,14	3,83	3,57	3,33	3,13	2,95	2,79	2,64	2,52	2,40	2,29
OfdmRate12MbpsBW10MHz	4,83	4,40	4,02	3,73	3,44	3,24	3,02	2,85	2,68	2,54	2,42	2,30	2,19
OfdmRate18MbpsBW10MHz	5,43	4,89	4,48	4,14	3,82	3,58	3,33	3,13	2,96	2,80	2,66	2,52	2,41
OfdmRate24MbpsBW10MHz	4,92	4,44	4,02	3,69	3,40	3,16	2,95	2,77	2,60	2,45	2,33	2,21	2,10
OfdmRate27MbpsBW10MHz	4,89	4,39	3,98	3,64	3,37	3,12	2,92	2,73	2,56	2,42	2,29	2,18	2,07
MinstrelWifiManager	2,39	2,29	2,17	2,04	1,91	1,79	1,68	1,59	1,51	1,43	1,36	1,30	1,24
ArfWifiManager	2,39	2,29	2,17	2,04	1,91	1,79	1,68	1,59	1,50	1,43	1,36	1,30	1,24

Tabla 5-4. Experimento 2. Tasa Efectiva

Uno de los aspectos más llamativos de esta gráfica es la repentina caída que sufren los protocolos de tasa adaptativa. El problema surge en las direcciones multicast o de difusión. En el escenario 1, al sólo tener que transmitir a una baliza podíamos transmitir a una dirección IP en particular (esto se ha hecho así sabiendo que si no los protocolos de tasa adaptativa no iban a funcionar). Sin embargo en este escenario, como no podemos saber a qué baliza tenemos que transmitir en cada momento, es necesario utilizar direcciones multicast.

El problema con las direcciones multicast surge debido a la dificultad que supone intentar estimar la correlación de los paquetes recibidos. Esto es algo muy estudiado en las transmisiones unicast, no así en las multicast. Por lo que la mayoría de los protocolos de tasa adaptativa (y todos los que están implementados en NS-3) toman las tasas más bajas disponibles, lo que explica la caída que vemos en las gráficas

Otro de los aspectos que puede llamar la atención en esta gráfica es el hecho de que para este escenario transmitir a 18 Mbps es mejor que hacerlo a 9 Mbps, en contraposición al escenario 1.

El motivo es el fenómeno del desvanecimiento comentado en el capítulo 4. Este fenómeno hace que la potencia con la que llega la señal al receptor varíe de una forma que se suele modelarse como un proceso aleatorio.

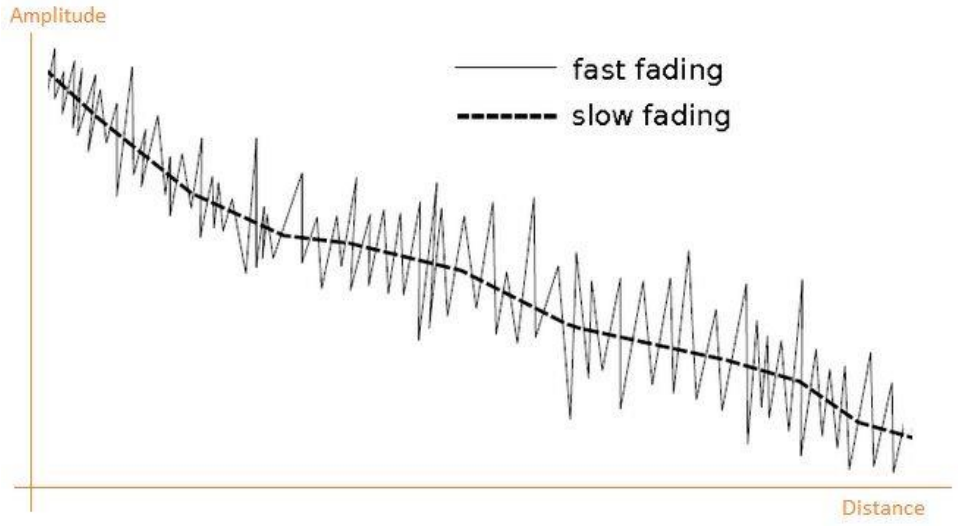


Figura 5-5. Fenómenos de fast fading y slow fading.

Esto puede hacer que, por cuestiones de probabilidad, nos llegue un paquete con una potencia suficiente para aceptarlo, que de otra forma daríamos por erróneo. El motivo por el que esto ocurre con direcciones multicast y no con las unicast es la diferencia de frecuencia con la que transmitimos cuando no estamos en zona de cobertura. Cuando usamos direcciones unicast se emplea el protocolo ARP. Tal y como hemos configurado el escenario, el protocolo ARP solo transmite un paquete cada segundo en caso de que no se le responda. Esto hace que sea muy improbable que fuera del rango de cobertura nos llegue un paquete con la potencia suficiente como para aceptarlo.

Sin embargo, cuando usamos direcciones multicast no hay un límite de paquetes a transmitir, y el vehículo que emite satura el canal con todos los paquetes que puede (esta saturación tendremos por supuesto que controlarla en un escenario real). Lo que hace mucho más probable que nos llegue un paquete fuera del rango de cobertura con la potencia suficiente para aceptarlo. Luego la tasa de 18 Mbps que es el doble de rápida que la de 9 Mbps, ahora permite transmitir más de la mitad de tiempo que con la de 9 Mbps, con lo que al final conseguimos mejores resultados.

5.2.3 Experimento 3. Análisis de las comunicaciones V2X bidireccionales.

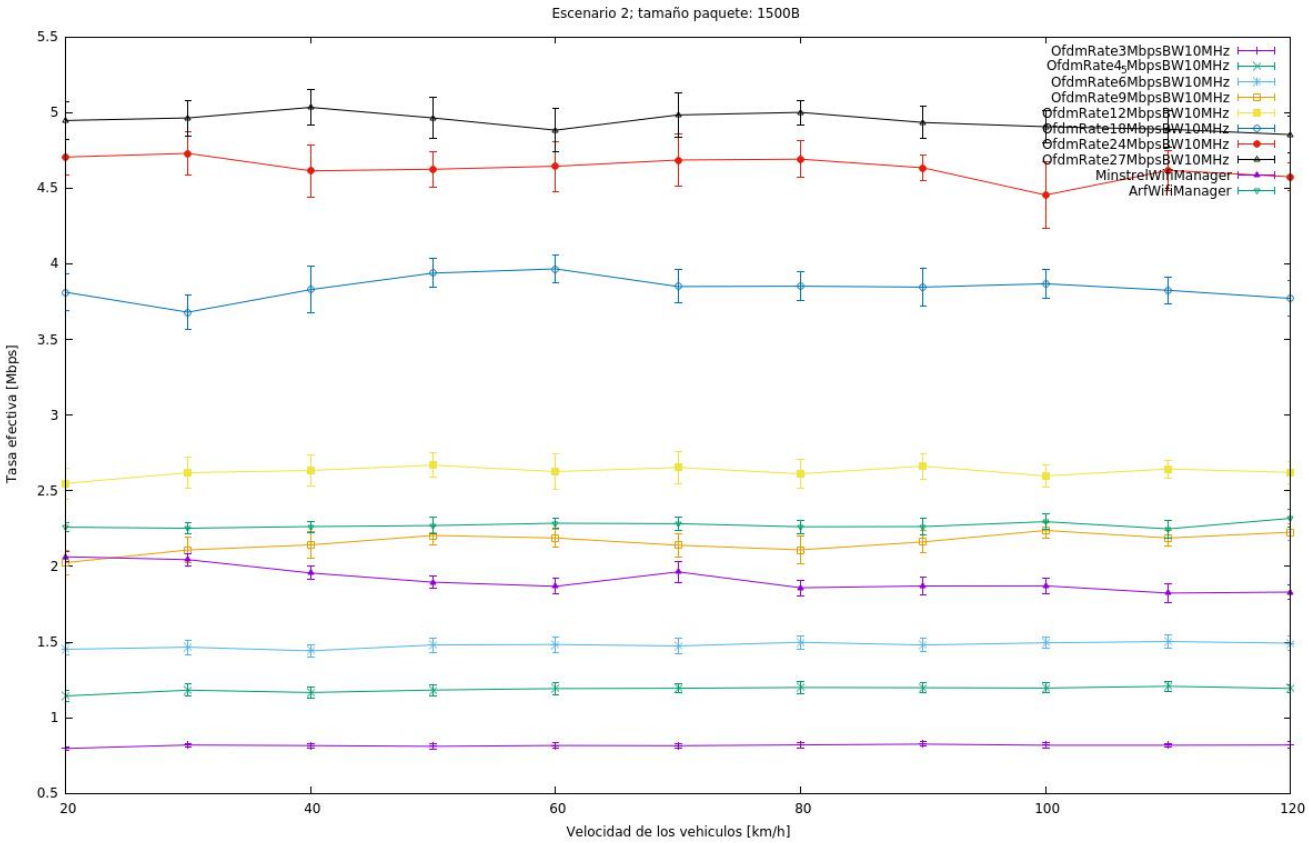


Figura 5-6. Experimento 3. Tasa Efectiva

Velocidad (km/h)	20	30	40	50	60	70	80	90	100	110	120
Tasa de transmisión	Tasa Efectiva (Mbps)										
OfdmRate3MbpsBW10MHz	0,79	0,81	0,81	0,81	0,81	0,81	0,82	0,82	0,81	0,81	0,81
OfdmRate4_5MbpsBW10MHz	1,14	1,18	1,16	1,18	1,19	1,19	1,19	1,19	1,19	1,20	1,19
OfdmRate6MbpsBW10MHz	1,45	1,46	1,44	1,48	1,48	1,47	1,49	1,48	1,49	1,50	1,49
OfdmRate9MbpsBW10MHz	2,02	2,10	2,14	2,20	2,18	2,14	2,10	2,16	2,23	2,18	2,22
OfdmRate12MbpsBW10MHz	2,54	2,61	2,63	2,66	2,62	2,65	2,61	2,66	2,59	2,64	2,62
OfdmRate18MbpsBW10MHz	3,81	3,68	3,82	3,93	3,96	3,84	3,85	3,84	3,86	3,82	3,77
OfdmRate24MbpsBW10MHz	4,70	4,72	4,61	4,62	4,64	4,68	4,69	4,63	4,45	4,61	4,57
OfdmRate27MbpsBW10MHz	4,94	4,96	5,03	4,96	4,88	4,98	5,00	4,93	4,90	4,88	4,85
MinstrelWifiManager	2,06	2,04	1,95	1,89	1,86	1,96	1,85	1,87	1,87	1,82	1,83
ArfWifiManager	2,25	2,25	2,26	2,26	2,28	2,28	2,26	2,26	2,29	2,24	2,31

Tabla 5-5. Experimento 3. Tasa Efectiva

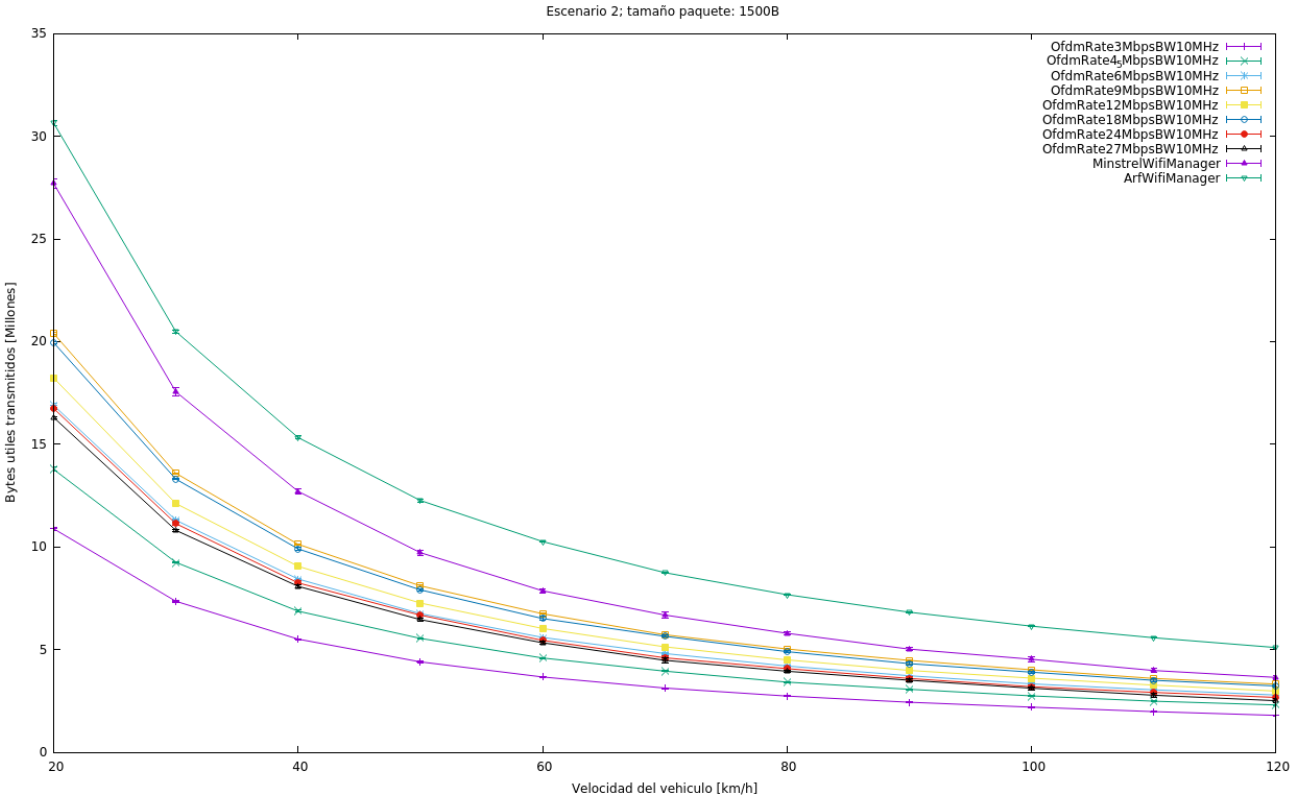


Figura 5-7. Experimento 3. Bytes útiles

Velocidad (km/h)	20	30	40	50	60	70	80	90	100	110	120
Tasa de transmisión	Bytes útiles (Millones)										
OfdmRate3MbpsBW10MHz	10,88	7,35	5,51	4,40	3,67	3,13	2,74	2,44	2,20	1,98	1,80
OfdmRate4_5MbpsBW10MHz	13,78	9,25	6,87	5,55	4,59	3,95	3,42	3,07	2,75	2,49	2,30
OfdmRate6MbpsBW10MHz	16,89	11,30	8,43	6,75	5,59	4,81	4,20	3,73	3,33	3,04	2,79
OfdmRate9MbpsBW10MHz	20,39	13,59	10,13	8,11	6,75	5,72	5,03	4,48	4,02	3,60	3,32
OfdmRate12MbpsBW10MHz	18,22	12,11	9,06	7,27	6,03	5,14	4,50	3,99	3,61	3,27	2,98
OfdmRate18MbpsBW10MHz	19,95	13,30	9,90	7,91	6,51	5,65	4,90	4,31	3,90	3,51	3,23
OfdmRate24MbpsBW10MHz	16,74	11,13	8,25	6,68	5,45	4,60	4,07	3,60	3,19	2,91	2,67
OfdmRate27MbpsBW10MHz	16,30	10,81	8,08	6,46	5,33	4,48	3,95	3,51	3,12	2,78	2,51
MinstrelWifiManager	27,69	17,55	12,70	9,72	7,86	6,68	5,80	5,02	4,53	3,98	3,65
ArfWifiManager	30,64	20,48	15,32	12,26	10,26	8,75	7,68	6,82	6,14	5,58	5,10

Tabla 5-6. Experimento 3. Bytes útiles

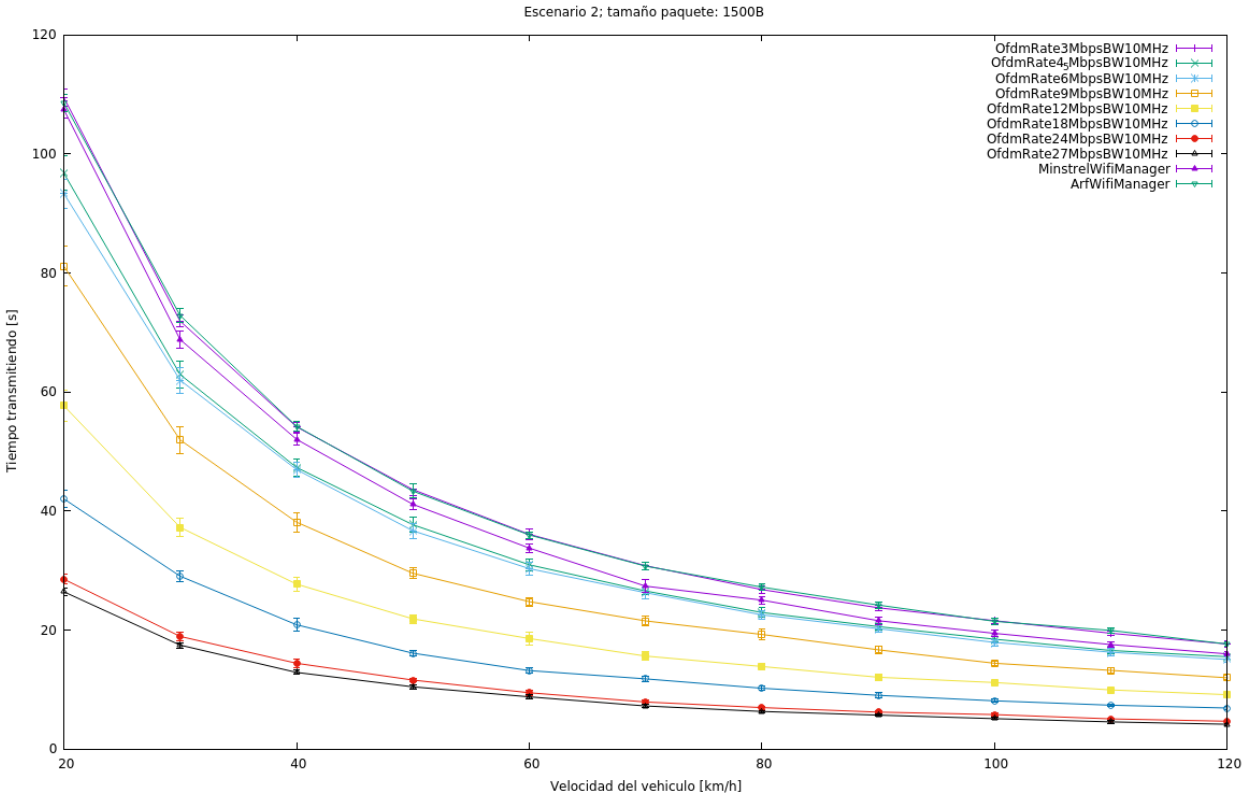


Figura 5-8. Experimento 3. Tiempo transmitiendo

Velocidad (km/h)	20	30	40	50	60	70	80	90	100	110	120
Tasa de transmisión	Tiempo transmitiendo (s)										
OfdmRate3MbpsBW10MHz	109,47	71,89	54,14	43,59	36,08	30,81	26,79	23,73	21,58	19,45	17,64
OfdmRate4_5MbpsBW10MHz	96,76	62,93	47,31	37,69	30,97	26,56	22,94	20,58	18,46	16,57	15,51
OfdmRate6MbpsBW10MHz	93,34	61,97	46,96	36,67	30,32	26,28	22,55	20,24	17,91	16,28	15,04
OfdmRate9MbpsBW10MHz	81,13	51,95	38,12	29,54	24,76	21,53	19,26	16,67	14,41	13,22	11,98
OfdmRate12MbpsBW10MHz	57,68	37,26	27,72	21,87	18,57	15,62	13,87	12,06	11,17	9,94	9,15
OfdmRate18MbpsBW10MHz	42,05	29,06	20,88	16,12	13,18	11,79	10,22	9,03	8,10	7,36	6,89
OfdmRate24MbpsBW10MHz	28,55	18,93	14,41	11,60	9,45	7,91	6,97	6,22	5,80	5,07	4,68
OfdmRate27MbpsBW10MHz	26,44	17,46	12,88	10,45	8,78	7,23	6,33	5,70	5,11	4,56	4,16
MinstrelWifiManager	107,46	68,80	52,03	41,12	33,76	27,37	25,03	21,57	19,44	17,55	15,98
ArfWifiManager	108,60	72,85	54,23	43,34	35,97	30,73	27,21	24,19	21,45	19,92	17,65

Tabla 5-7. Experimento 3. Tiempo transmitiendo

A la vista de los resultados podemos observar como las capacidades del canal se reducen significativamente cuando dos nodos están transmitiendo a la vez. Somos capaces de enviar sólo la cuarta parte de información, aunque solo hayamos reducido a la mitad el tiempo de cobertura. Esto es debido al número de colisiones del canal, ya que pasamos de no tener ninguna a que se produzcan frecuentemente.

Otro fenómeno observado en este experimento (que no se refleja en las gráficas) es el posible apoderamiento del canal por parte de uno de los nodos. Es decir, cabe la posibilidad de que uno de los nodos siempre este transmitiendo ya que no hay un punto de acceso que sincronice las estaciones para que todos tengan su oportunidad para transmitir, con lo que el otro nodo puede quedarse sin mandar información.

5.2.4 Experimento 4. Análisis de la influencia del tamaño del paquete.

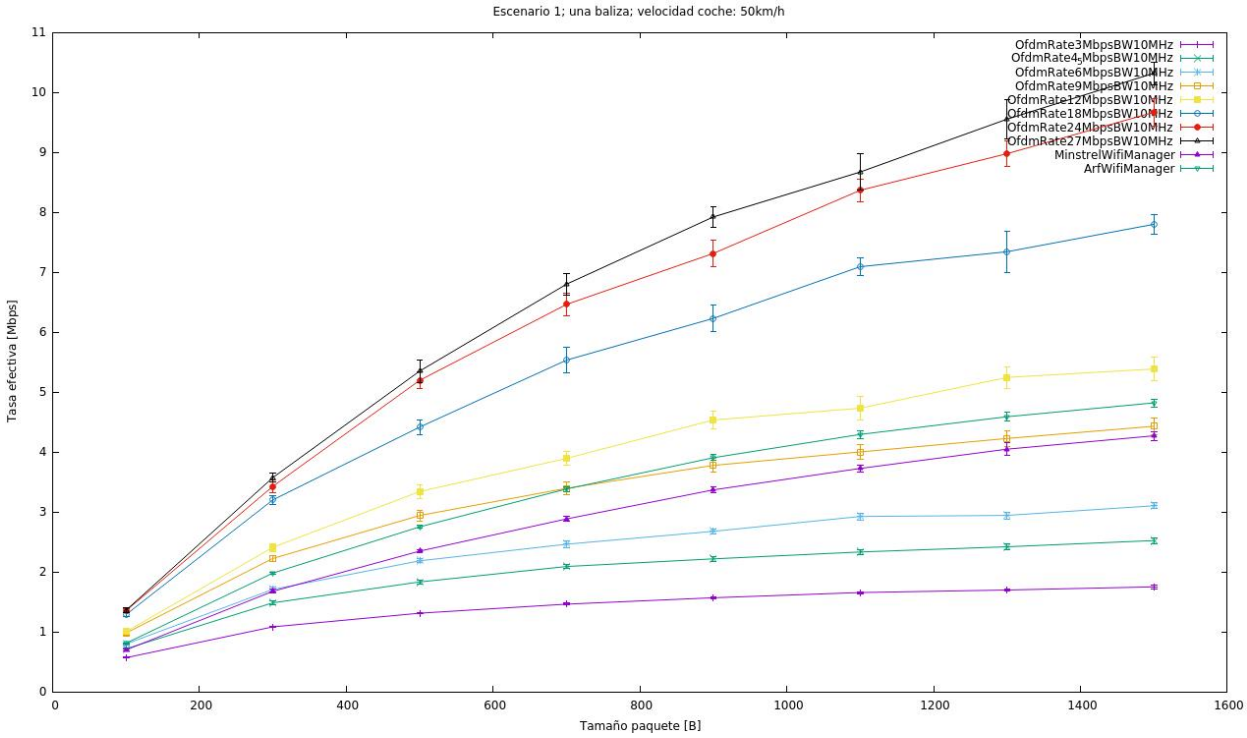


Figura 5-9. Experimento 4. Tasa Efectiva

Velocidad (km/h)	100	300	500	700	900	1000	1300	1500
Tasa de transmisión	Tasa Efectiva (Mbps)							
OfdmRate3MbpsBW10MHz	057	1,08	1,31	1,46	1,57	1,65	1,69	1,75
OfdmRate4_5MbpsBW10MHz	0,72	1,48	1,83	2,09	2,21	2,33	2,42	2,52
OfdmRate6MbpsBW10MHz	0,80	1,70	2,18	2,46	2,67	2,92	2,94	3,10
OfdmRate9MbpsBW10MHz	0,98	2,22	2,94	3,39	3,77	4,00	4,22	4,43
OfdmRate12MbpsBW10MHz	2,41	3,34	3,89	4,53	4,73	5,24	5,38	29,04
OfdmRate18MbpsBW10MHz	1,28	3,20	4,41	5,53	6,23	7,09	7,34	7,79
OfdmRate24MbpsBW10MHz	1,35	3,42	5,19	6,46	7,31	8,36	8,97	9,66
OfdmRate27MbpsBW10MHz	1,36	3,57	5,35	6,80	7,92	8,67	9,55	10,31
MinstrelWifiManager	0,69	1,67	2,34	2,88	3,37	3,72	4,04	4,27
ArfWifiManager	0,81	1,98	2,75	3,38	3,90	4,29	4,58	4,82

Tabla 5-8. Experimento 4. Tasa Efectiva

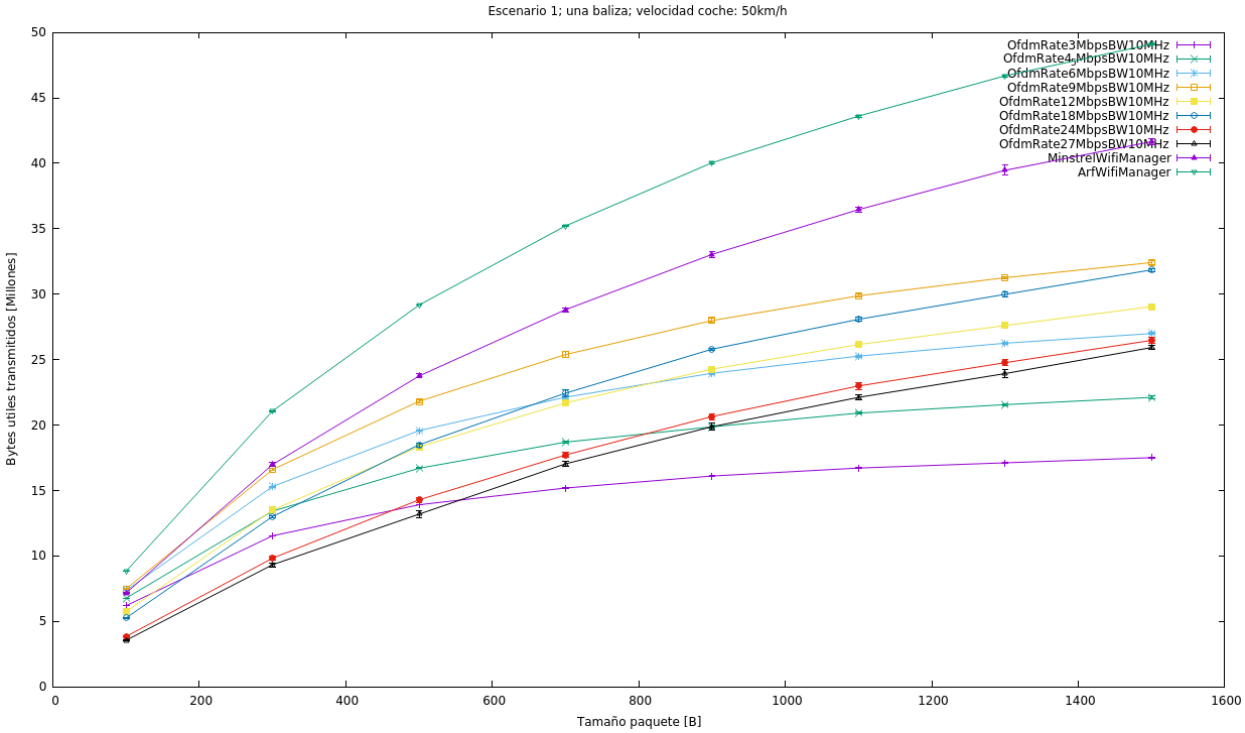


Figura 5-10. Experimento 4. Bytes útiles

Velocidad (km/h)	100	300	500	700	900	1100	1300	1500
Tasa de transmisión	Bytes útiles (Millones)							
OfdmRate3MbpsBW10MHz	6,19	11,54	13,91	15,19	16,10	16,71	17,11	17,50
OfdmRate4_5MbpsBW10MHz	6,77	13,42	16,69	18,69	19,86	20,91	21,55	22,11
OfdmRate6MbpsBW10MHz	7,36	15,30	19,57	22,13	23,95	25,26	26,25	26,99
OfdmRate9MbpsBW10MHz	7,48	16,60	21,81	25,39	27,98	29,87	31,25	32,41
OfdmRate12MbpsBW10MHz	5,75	13,50	18,33	21,69	24,27	26,14	27,60	29,04
OfdmRate18MbpsBW10MHz	5,26	13,01	18,49	22,44	25,79	28,08	29,99	31,84
OfdmRate24MbpsBW10MHz	3,84	9,83	14,28	17,71	20,65	22,98	24,76	26,46
OfdmRate27MbpsBW10MHz	3,55	9,33	13,19	17,03	19,87	22,11	23,93	25,92
MinstrelWifiManager	7,16	16,98	23,75	28,81	33,03	36,45	39,46	41,65
ArfWifiManager	8,83	21,06	29,17	35,21	40,03	43,60	46,68	49,09

Tabla 5-9. Experimento 4. Bytes útiles

Como podemos ver en las siguientes gráficas, mientras mayor sea el tamaño del paquete que transmitamos mayor será la cantidad de información que somos capaces de hacer llegar al otro extremo de la comunicación. Sin embargo, esto no quiere decir que transmitir con el tamaño de paquete más grande posible sea lo mejor, ya que este es más probable que contenga errores. En un entorno real con varios vehículos transmitiendo a la vez, probablemente haya que buscar un equilibrio entre la cantidad de tiempo que queremos ocupar el canal y la tolerancia a los errores que se puede soportar.

5.2.5 Experimento 5. Análisis del protocolo Minstrel.

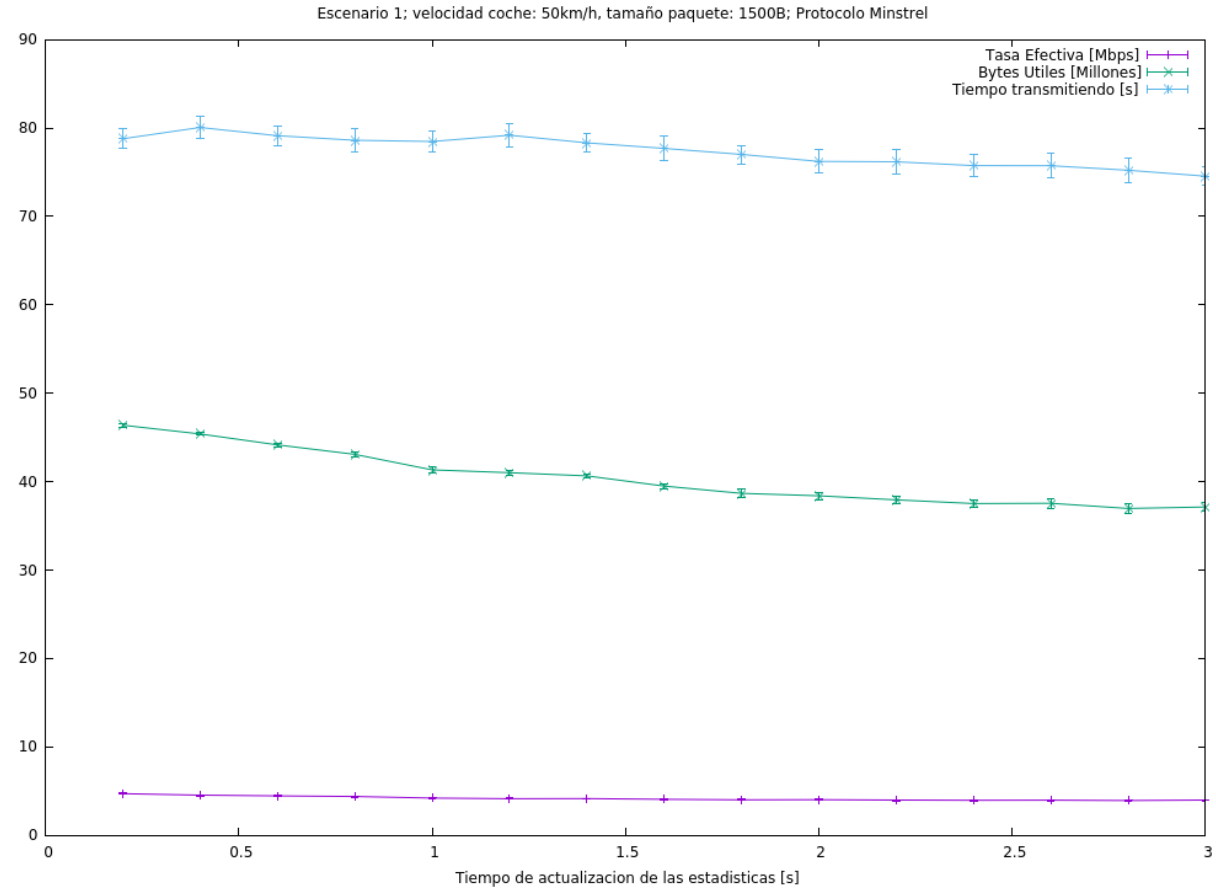


Figura 5-11. Experimento 5

Velocidad (km/h)	0,2	0,4	0,6	0,8	1,0	1,2	1,4	1,6	1,8	2,0	2,2	2,4	2,6	2,8	3,0
Tasa Efectiva (Mbps)	4,71	4,54	4,46	4,39	4,21	4,14	4,15	4,07	4,01	4,03	3,98	3,96	3,97	3,93	3,98
Bytes útiles (Millones)	46,37	45,39	44,16	43,08	41,34	41,01	40,65	39,49	38,67	38,40	37,93	37,51	37,54	36,96	37,13
Tiempo transmitiendo (s)	78,77	80,05	79,12	78,60	78,47	79,17	78,31	77,69	77,00	76,22	76,17	75,74	75,72	75,21	74,54

Tabla 5-10. Experimento 5

Con este experimento hemos podido determinar bajo qué condiciones el protocolo Minstrel opera mejor en estas redes. Ya que son entornos muy cambiantes es más óptimo forzar al protocolo a actualizar sus estadísticas con la mayor frecuencia posible. Aunque con ello sobrecarguemos un poco más la red, los resultados muestran que se aprovecha mejor el canal y se consigue transmitir más información

5.2.6 Experimento 6. Análisis del protocolo Arf.

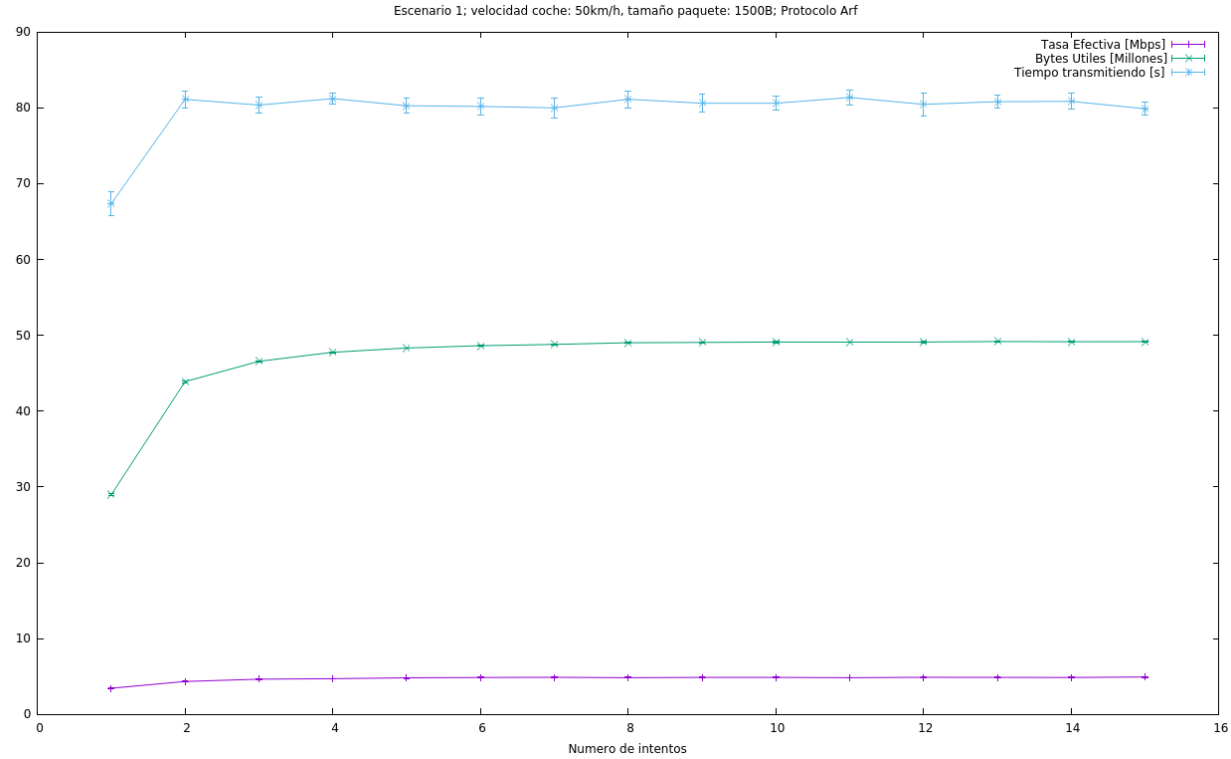


Figura 5-12. Experimento 6

Número de intentos	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Tasa Efectiva (Mbps)	3,44	4,33	4,63	4,70	4,81	4,85	4,88	4,83	4,87	4,87	4,83	4,89	4,87	4,86	4,92
Bytes útiles (Millones)	28,94	43,87	46,55	47,75	48,31	48,62	48,79	49,01	49,05	49,10	49,09	49,09	49,17	49,13	49,16
Tiempo transmitiendo (s)	67,31	81,10	80,36	81,20	80,27	80,17	79,99	81,11	80,59	80,59	81,35	80,44	80,80	80,85	79,85

Tabla 5-11. Experimento 6

Como podemos observar no hay grandes diferencias en los resultados cuando variamos el número de paquetes necesarios para que Arf aumente la tasa de transmisión. Esto se debe a que la diferencia de tiempo que existe entre recibir 10 o 15 paquetes es mínima, por lo que el protocolo mantiene su agilidad para adaptar la tasa.

Todo esto no ocurre si ese número de paquetes es muy pequeño, ya que Arf aumenta la tasa cada vez que un paquete llega correctamente, equivocándose muchas veces y desperdiciando mucho tiempo en ello.

5.2.7 Experimento 7. Análisis de la influencia de la altura de la antena

Hemos realizado experimento variando la altura de la antena, pero estos no han arrojado información de interés.

Variando la altura de la antena receptora de 1 a 10 metros se han conseguido prácticamente los mismos resultados (salvo por los pocos metros de diferencia que existen en el rango de cobertura). Esto es debido a la ausencia de obstáculos en el escenario de nuestra simulación. En un entorno con obstáculos convendría volver a realizar este experimento y comprobar como varían los parámetros de la red.

6 CONCLUSIONES

6.1 Resumen del análisis

Una vez entendido como se comportan los escenarios que hemos simulado, cabe preguntarnos cuál es la tasa óptima con la que transmitir. Sin embargo, esta es una pregunta que aún no se puede contestar, pues dependerá de los requisitos de la aplicación a usar. Si por un lado lo que nos interesa es transmitir la mayor cantidad de bytes posibles entonces la mejor opción será usar los protocolos de tasa adaptativa. Si por el contrario, nos interesa tener el canal libre el mayor tiempo posible nuestra mejor opción será usar las tasas más altas para que transmitan en el menor tiempo posible.

Además, tras analizar los resultados hemos obtenido las siguientes conclusiones:

1. **Los protocolos de tasa adaptativa más extendidos no funcionan con multicast.** Afortunadamente se está trabajando en este problema ahora que es crítico en algunas redes y ya se han propuesto algunas soluciones. Codificar y probar estas soluciones podría unos de los primeros pasos a tomar en futuros trabajos.
2. Habrá que estudiar cuál será la cantidad de Bytes a transmitir para poder determinar a qué velocidad tendrá que ir el vehículo para que la aplicación funcione
3. Habrá que estimar cuántas y cómo de separadas van a estar las balizas en un recorrido medio, para poder así diseñar con ciertas garantías la aplicación final.
4. **Sólo con tener un nodo más trasmitiendo enviamos la cuarta parte de bytes totales**. Es decir, las interferencias en estas redes juegan un papel fundamental, y habrá que estudiar el canal en un caso más real con más vehículos transmitiendo a la vez, para comprobar cuánta información somos capaces de enviar y hasta dónde se reducen las capacidades del canal.
5. El hecho de que un nodo se apodere del canal puede suponer un problema si existe otro con información más prioritaria que transmitir.
6. Si estamos transmitiendo paquetes de pocos bytes no es recomendable usar tasas de transmisiones muy altas ya que estamos aumentado la probabilidad de error debido al uso de constelaciones más restrictivas.
7. La mejor opción para los protocolos adaptativos es intentar aumentar su tasa de transmisión lo más rápido posible. Ya que las redes vehiculares son muy dinámicas y los tiempos de transmisión pueden llegar a ser muy cortos, es necesario detectar cuando podemos transmitir con tasas más altas lo más rápido posible.

6.2 Líneas futuras

Tras la realización del trabajo se han detectado varias líneas a seguir en futuras ampliaciones del mismo:

- Con el objetivo de obtener la mayor precisión posible, sería interesante obtener un modelo lo más real posible del modelo de pérdidas por propagación
- Estudiar cómo implementar protocolos de tasa adaptativa para transmisiones multicast, pues estas van a ser necesarias previsiblemente en entornos vehiculares.
- Tras esto el siguiente paso natural sería programar un escenario más real en el que existan obstáculos y otros coches transmitiendo para poder obtener cómo se comporta la red en su caso más perjudicial para las comunicaciones.
- Por último, considero que a la vista de los resultados (teniendo en cuenta que el canal ofrece un estado óptimo para la comunicación) es interesante realizar experimentos con TCP y estudiar si es viable conseguir las prestaciones que ofrece este protocolo.

REFERENCIAS

1. **Evans, Dave.** *Internet de las cosas. Como la próxima evolución de Internet lo cambia todo.* 2011. Available: http://www.cisco.com/c/dam/global/es_mx/solutions/executive/assets/pdf/internet-of-things-iot-ibsg.pdf.

2. **Shannon, Robert E.** *Introduction to the art and science of simulation.* 1998. Available: http://www.informs-sim.org/wsc98papers/001.PDF.

3. **Prof. Kavicka, Antonín.** *Basic methods synchronizing the course of simulation experiments.*

4. **Prof. Madinabeitia Luque, Germán.** Introducción a NS-3.

5. **Documentación Helpers NS-3. [En línea] https://www.nsnam.org/docs/manual/html/helpers.html.**

6. ***IEEE 802.11p: Towards an International Standard.*** **Delgrossi, Luca. 2008.**

7. ***Standardization of Wireless Vehicular Communications within IEEE and ETSI.*** **Sjöberg, Katrin. 2011.**

8. IEEE. ***Wireless Access in Vehicular Enviroments.*** **2010.**

9. Henderson, Guangyu Pei and Thomas R. ***Validation of OFDM error rate model in ns-3.***

10. Tarikul Islam, Yongchang Hu, Dr. Ertan Onur. ***Realistic Simulation of IEEE 802.11p Channel in Mobile Vehicle to Vehicle Communication.***

11. NS-3. ns3::NakagamiPropagationLossModel Class Reference. [En línea] https://www.nsnam.org/doxygen/classns3_1_1_nakagami_propagation_loss_model.html#details.

12. —. WAVE models. [En línea] https://www.nsnam.org/docs/models/html/wave.html.

13. —. Wi-Fi Module Design Documentation. [En línea] https://www.nsnam.org/docs/models/html/wifi-design.html.

14. Documentación Movilidad NS-3. [En línea] https://www.nsnam.org/docs/models/html/mobility.html.

ÍNDICE DE CONCEPTOS

GLOSARIO

AP
Access Point 19
AWGN
Additive white Gaussian noise 27
BSS
Basic Service Set 19
DSRC
Dedicated Short Range Communication 17
IBSG
Grupo de soluciones empresariales basadas en Internet 1
IEEE
Institute of Electrical and Electronics Engineers 19
IoT
Internet de las Cosas 1
OCB
Outside of the Context of a BSS 23
OFDM
Orthogonal Frequency-Division Multiple Access 23
OSI
Open System Interconnection 19
POO
Programación Orientada a Objetos 10
QoS
Calidad de Servicio 14
STA
Station 19
V2I
Vehículo a Infraestructura 3
V2V
Vehículo a vehículo 3
V2X
Vehículo a Vehículo/Infraestructura 3
WAVE
Wireless Access in Vehicular Environments 17

ANEXO A: CÓDIGO NS-3

En este anexo vamos a mostrar el código de NS-3 para que el lector pueda comprobar cómo se ha programado todo lo comentado en el capítulo 4 de la memoria.

A.1 Canal WiFi

```
/*
 * ---------- CANAL WIFI (CAPAS PHY Y MAC) ----------
 */
Wifi80211pHelper wifiHelper;
    // Ya que el estandar 802.11p solo permite las comunicaciones a 5, 10 o 20MHZ
    // y NS-3 solo permite 10MHZ para 802.11p
    wifiHelper.SetStandard (WIFI_PHY_STANDARD_80211_10MHZ);
    if (strcmp(tasaPHY, "MinstrelWifiManager") != 0 &&
                strcmp(tasaPHY, "ArfWifiManager") != 0) {
        wifiHelper.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
                "DataMode", StringValue (tasaPHY),
                "ControlMode", StringValue (tasaPHY),
                "NonUnicastMode", StringValue (tasaPHY));
        NS_LOG_INFO("Tasa establecida a: " << tasaPHY);
    } else if (strcmp(tasaPHY, "MinstrelWifiManager") == 0) {
        Config::SetDefault("ns3::MinstrelWifiManager::UpdateStatistics",
                TimeValue(Time(minstrelTime)));
        Config::SetDefault("ns3::MinstrelWifiManager::PacketLength",
                UintegerValue(minstrelPaq));
        wifiHelper.SetRemoteStationManager ("ns3::MinstrelWifiManager");
        NS_LOG_INFO("Utilizando algoritmo MinstrelWifiManager");
    } else {
        Config::SetDefault("ns3::ArfWifiManager::SuccessThreshold",
                UintegerValue(intentos));
        wifiHelper.SetRemoteStationManager ("ns3::ArfWifiManager");
        NS_LOG_INFO("Utilizando algoritmo ArfWifiManager");
    }

// Configuramos el modelos NIST de errores que es el recomendado por NS-3
// para canales sin interferencias
YansWifiPhyHelper wifiPHY;
    wifiPHY.SetErrorRateModel ("ns3::NistErrorRateModel");
// Configuramos el modelo de perdidas de propagacion de Nakagami para
// modelar los desvanecimientos de la señal
Config::SetDefault("ns3::NakagamiPropagationLossModel::Distance1", DoubleValue(300));
Config::SetDefault("ns3::NakagamiPropagationLossModel::Distance2", DoubleValue(1000));

Config::SetDefault("ns3::NakagamiPropagationLossModel::m0", DoubleValue(M0));
Config::SetDefault("ns3::NakagamiPropagationLossModel::m1", DoubleValue(M1));
Config::SetDefault("ns3::NakagamiPropagationLossModel::m2", DoubleValue(M2));
YansWifiChannelHelper wifiChannel;
    wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");
    // Añadimos el modelo de perdidas de propagacion LogDistance para modelar
    // la perdida de portencia de una señal cuando recorre una distancia
    wifiChannel.AddPropagationLoss ("ns3::LogDistancePropagationLossModel");
    wifiChannel.AddPropagationLoss ("ns3::NakagamiPropagationLossModel");
Ptr<YansWifiChannel> channel = wifiChannel.Create ();
wifiPHY.SetChannel (channel);

wifiPHY.SetPcapDataLinkType (YansWifiPhyHelper::DLT_IEEE802_11);
// Configuramos la capa MAC para que siga la norma 80211p con calidad de servicio
QosWaveMacHelper wifiMAC;
    wifiMAC.SetType ("ns3::OcbWifiMac", "QosSupported", BooleanValue (true));
```

```
53.
54. NetDeviceContainer devices = wifiHelper.Install(wifiPHY, wifiMAC, nodes);
```

A.2 Movilidad

```
1.  /*
2.       * ---------- MOVILIDAD ----------
3.       */
4.
5.      // Dependiendo del escenario definiremos una baliza con posicion constante
6.      // a la que se le aproxima un coche con velocidad constante,
7.      // o dos vehiculos aprosimandose entre ellos.
8.      if (escenario == 1)
9.      {
10.         // Creamos las balizas separadas uniformemente una distancia "distanciaBalizas"
    ,
11.         // empezando por la posicion 0
12.         // º <---x(m)----> º <-----x(m----> º <-----x(m)----> º <----x(m)----> º
13.
14.         for (int i = 1; i <= numBalizas; i++) {
15.             MobilityHelper movilidadBaliza;
16.
17.             Ptr<ListPositionAllocator> positionBalizas = CreateObject<ListPositionAlloc
    ator> ();
18.             positionBalizas->Add (Vector (distanciaBalizas * (i-
    1), 0.0, alturaAntena));
19.
20.             movilidadBaliza.SetPositionAllocator (positionBalizas);
21.             movilidadBaliza.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
22.             movilidadBaliza.Install (nodes.Get(i));
23.
24.             NS_LOG_INFO("Posicion para la baliza " << i << ": "
25.                 << nodes.Get(i)->GetObject<ConstantPositionMobilityModel>()-
    >GetPosition());
26.         }
27.
28.         // Creamos el vehiculo en una posicion inicial a la izquierda de la primera
29.         // baliza e igual a la mitad de distancia de separacion entre balizas
30.         // V <--x/2(m)--> º <--x(m)--> º <--x(m)--> º <--x(m)--> º <--x(m)--> º
31.
32.         MobilityHelper movilidadVehiculo;
33.
34.         Ptr<ListPositionAllocator> positionVehiculo
35.                 = CreateObject<ListPositionAllocator> ();
36.         //positionVehiculo->Add (Vector (-distanciaBalizas/2, 0.0, 0.1));
37.         positionVehiculo->Add (Vector (posicIni1, 0.0, 1.0));
38.
39.         movilidadVehiculo.SetPositionAllocator (positionVehiculo);
40.         movilidadVehiculo.SetMobilityModel ("ns3::ConstantVelocityMobilityModel");
41.         movilidadVehiculo.Install (nodes.Get(0));
42.         // Definimos la velocidad del vehiculo en m/s
43.         nodes.Get(0)->GetObject<ConstantVelocityMobilityModel>()
44.                 ->SetVelocity(Vector(velocidad1, 0.0, 0.0));
45.
46.         NS_LOG_INFO("Posicion para el vehiculo: "
47.                 << nodes.Get(0)->GetObject<ConstantVelocityMobilityModel>()
48.                         ->GetPosition());
49.     }
50.     else if (escenario == 2)
51.     {
52.         MobilityHelper movilidadVehiculo1;
53.         MobilityHelper movilidadVehiculo2;
54.
```

```
        // Creamos los vehiculos con las posiciones y las velocidades pasadas por param
    etros

        Ptr<ListPositionAllocator> positionVehiculo1 = CreateObject<ListPositionAllocat
    or> ();
        positionVehiculo1->Add (Vector (posicIni1, 0.0, 1.0));
        movilidadVehiculo1.SetPositionAllocator (positionVehiculo1);
        movilidadVehiculo1.SetMobilityModel ("ns3::ConstantVelocityMobilityModel");
        movilidadVehiculo1.Install (nodes.Get(0));
        // Definimos la velocidad en m/s
        nodes.Get(0)->GetObject<ConstantVelocityMobilityModel>()
            ->SetVelocity(Vector(velocidad1, 0.0, 0.0));
        NS_LOG_INFO("Posicion para el vehiculo 1: "
                << nodes.Get(0)->GetObject<ConstantVelocityMobilityModel>()
                 ->GetPosition());
        NS_LOG_INFO("Velocidad para el vehiculo 1: "
                << nodes.Get(0)->GetObject<ConstantVelocityMobilityModel>()
                 ->GetVelocity());

        Ptr<ListPositionAllocator> positionVehiculo2
               = CreateObject<ListPositionAllocator> ();
        positionVehiculo2->Add (Vector (posicIni2, 0.0, 1.0));
        movilidadVehiculo2.SetPositionAllocator (positionVehiculo2);
        movilidadVehiculo2.SetMobilityModel ("ns3::ConstantVelocityMobilityModel");
        movilidadVehiculo2.Install (nodes.Get(1));
        // Definimos la velocidad en m/s
        nodes.Get(1)->GetObject<ConstantVelocityMobilityModel>()
            ->SetVelocity(Vector(velocidad2, 0.0, 0.0));
        NS_LOG_INFO("Posicion para el vehiculo 2: "
                << nodes.Get(1)->GetObject<ConstantVelocityMobilityModel>()
                ->GetPosition());
        NS_LOG_INFO("Velocidad para el vehiculo 2: "
                << nodes.Get(1)->GetObject<ConstantVelocityMobilityModel>()
                ->GetVelocity());
    }
```

A.3 Aplicaciones

```
/*
     * ---------- APLICACIONES ----------
     */

    // Instalamos una primera aplicacion sumidero de prueba
    uint16_t port = 9;

    ApplicationContainer BalizasApp;
    ApplicationContainer VehiculosApp;

    // Preparamos el servidor
    PacketSinkHelper server ("ns3::UdpSocketFactory",
                             Address (InetSocketAddress (Ipv4Address::GetAny (), port))
);

    // Variables aleatorias exponenciales de tiempos de ON y OFF (segundos)
    Ptr<ConstantRandomVariable> tOn = CreateObject<ConstantRandomVariable> ();
        tOn->SetAttribute("Constant", DoubleValue(1));
    Ptr<ConstantRandomVariable> tOff = CreateObject<ConstantRandomVariable> ();
        tOff->SetAttribute("Constant", DoubleValue(0));

    // Preparamos el cliente
    // Clientes OnOff en los equipos de la LAN, que envian los paquetes al sumidero
    if (escenario == 1 && numBalizas > 1) {
        OnOffHelper client ("ns3::UdpSocketFactory",
            Address (InetSocketAddress (ns3::Ipv4Address("192.168.0.255"), port)));
        // Asignamos los atributos de los clientes OnOff
```

```
27.         client.SetAttribute("OnTime", PointerValue(tOn));
28.         client.SetAttribute("OffTime", PointerValue(tOff));
29.         client.SetAttribute("PacketSize", UintegerValue(tamPaq));
30.         client.SetAttribute("DataRate", DataRateValue(DataRate(tasaAPP)));
31.
32.         VehiculosApp.Add(client.Install (nodes.Get(0)));
33.     } else if (escenario == 1 && numBalizas == 1) {
34.         OnOffHelper client ("ns3::UdpSocketFactory",
35.             Address (InetSocketAddress (ns3::Ipv4Address("192.168.0.2"), port)));
36.         client.SetAttribute("OnTime", PointerValue(tOn));
37.         client.SetAttribute("OffTime", PointerValue(tOff));
38.         client.SetAttribute("PacketSize", UintegerValue(tamPaq));
39.         client.SetAttribute("DataRate", DataRateValue(DataRate(tasaAPP)));
40.
41.         VehiculosApp.Add(client.Install (nodes.Get(0)));
42.     } else if (escenario == 2) {
43.         OnOffHelper client1 ("ns3::UdpSocketFactory",
44.             Address (InetSocketAddress (ns3::Ipv4Address("192.168.0.2"), port)));
45.         client1.SetAttribute("OnTime", PointerValue(tOn));
46.         client1.SetAttribute("OffTime", PointerValue(tOff));
47.         client1.SetAttribute("PacketSize", UintegerValue(tamPaq));
48.         client1.SetAttribute("DataRate", DataRateValue(DataRate(tasaAPP)));
49.
50.         OnOffHelper client2 ("ns3::UdpSocketFactory",
51.             Address (InetSocketAddress (ns3::Ipv4Address("192.168.0.1"), port)));
52.         client2.SetAttribute("OnTime", PointerValue(tOn));
53.         client2.SetAttribute("OffTime", PointerValue(tOff));
54.         client2.SetAttribute("PacketSize", UintegerValue(tamPaq));
55.         client2.SetAttribute("DataRate", DataRateValue(DataRate(tasaAPP)));
56.
57.         VehiculosApp.Add(client1.Install (nodes.Get(0)));
58.         VehiculosApp.Add(client2.Install (nodes.Get(1)));
59.     }
60.
61.     if (escenario == 1) {
62.         for (int i = 1; i <= numBalizas; i++) {
63.             BalizasApp.Add(server.Install (nodes.Get(i)));
64.             BalizasApp.Start(Time("0s"));
65.         }
66.         VehiculosApp.Start (Time ("1s"));
67.     } else if (escenario == 2) {
68.         BalizasApp.Add(server.Install (nodes));
69.         BalizasApp.Start(Time("0s"));
70.
71.         // Para evitar que las dos aplicaciones empiezen a la vez y no se
72.         // permitan la comunicacion entre ellas
73.         // establecemos un tiempo de inicio aleatorio
74.         Ptr<NormalRandomVariable> tStart = CreateObject<NormalRandomVariable> ();
75.             tStart->SetAttribute("Mean", DoubleValue(1.0));
76.             tStart->SetAttribute("Variance", DoubleValue(0.3));
77.             tStart->SetAttribute("Bound", DoubleValue(0.5));
78.
79.         std::ostringstream tiempoInicio;
80.
81.         // Establecemos los tiempos de inicios de las aplicaciones
82.         // en tiempos aleatorios para evitar
83.         // que siempre transmitan a la vez
84.         tiempoInicio << tStart->GetValue() << "s";
85.             VehiculosApp.Get(0)->SetStartTime (Time (tiempoInicio.str()));
86.         NS_LOG_INFO("Tiempo de inicio establecido para la aplicacion del primer vehicul
    o: "
87.             << tiempoInicio.str());
88.
89.         tiempoInicio.str("");
90.         tiempoInicio.clear();
91.         tiempoInicio << tStart->GetValue() << "s";
```

```
92.             VehiculosApp.Get(1)->SetStartTime (Time (tiempoInicio.str()));
93.         NS_LOG_INFO("Tiempo de inicio establecido para la aplicacion del segundo vehicu
    lo: "
94.             << tiempoInicio.str());
95.     }
```

A.4 Observador

```
1.   /**
2.    * Metodo que se ejecuta cuando se transmite un paquete
3.    * a nivel de aplicacion
4.    */
5.  void
6.  Observador::InicioTransmAPP ( Ptr<const Packet> paquete )
7.  {
8.      NS_LOG_FUNCTION(paquete);
9.      NS_LOG_INFO("APP manda paquete a los: " << Simulator::Now().GetSeconds()
10.         << "s. De tamaño: " << paquete->GetSize()
11.             << "B. Con UID: " << paquete->GetUid());
12.
13.     // Utilizamos un mapa para evitar que
14.         // un paquete que llegue a dos antenas se cuente por dos
15.     uint64_t uid = paquete->GetUid();
16.     paquetesAPP[ uid ] = paquete->GetSize();
17.
18.     NS_LOG_INFO("\n");
19.
20. }
21.
22. /**
23.   * Metodo que se ejecuta cuando se recibe un paquete
24.   * a nivel de aplicacion
25.   */
26. void
27. Observador::FinRecepAPP ( Ptr<const Packet> paquete, Address const& direccion )
28. {
29.     NS_LOG_FUNCTION(paquete << direccion);
30.     NS_LOG_INFO("APP recibe paquete a los: " << Simulator::Now().GetSeconds()
31.             << "s. De tamaño: " << paquete->GetSize()
32.             << "B. Con UID: " << paquete->GetUid());
33.
34.     // Buscamos el paquete en el mapa
35.     if (enCobertura == true) {
36.         uint64_t uid = paquete->GetUid();
37.         std::map<uint64_t, uint32_t>::iterator iteradorMapa = paquetesAPP.find(uid);
38.
39.         // Comprobamos que el paquete no se ha contado aun
40.         if( iteradorMapa == paquetesAPP.end() )
41.             NS_LOG_WARN("Paquete recibido con UID " << uid << " no encontrado.");
42.         else
43.         {
44.             bUtiles += paquetesAPP[ uid ];
45.             // Borramos el paquete del mapa para asegurar que no se cuente en otra bali
    za
46.             paquetesAPP.erase(iteradorMapa);
47.         }
48.     }
49.
50.     NS_LOG_INFO("\n");
51. }
52.
53. /**
54.   * Metodo que se ejecuta cuando se transmite un paquete
55.   * a nivel de aplicacion (para el vehiculo 2)
56.   */
```

```
57. void
58. Observador::FinRecepAPPEsc2 ( Ptr<const Packet> paquete, Address const& direccion )
59. {
60.     NS_LOG_FUNCTION(paquete << direccion);
61.     NS_LOG_INFO("APP recibe paquete a los: " << Simulator::Now().GetSeconds()
62.             << "s. De tamaño: " << paquete->GetSize()
63.             << "B. Con UID: " << paquete->GetUid());
64.
65.     // Buscamos el paquete en el mapa
66.     if (enCobertura == true) {
67.         uint64_t uid = paquete->GetUid();
68.         std::map<uint64_t, uint32_t>::iterator iteradorMapa = paquetesAPP.find(uid);
69.
70.         // Comprobamos que el paquete no se ha contado aun
71.         if( iteradorMapa == paquetesAPP.end() )
72.             NS_LOG_WARN("Paquete recibido con UID " << uid << " no encontrado.");
73.         else
74.         {
75.             bUtilesEsc2 += paquetesAPP[ uid ];
76.             // Borramos el paquete del mapa para asegurar
77.                         // que no se cuente en otra baliza
78.             paquetesAPP.erase(iteradorMapa);
79.         }
80.     }
81.
82.     NS_LOG_INFO("\n");
83. }
84.
85. /**
86.   * Metodo que se ejecuta cuando se transmite un paquete
87.   * a nivel fisico
88.   */
89. void
90. Observador::InicioTransmPHY ( Ptr<const Packet> paquete )
91. {
92.     NS_LOG_FUNCTION(paquete);
93.     NS_LOG_INFO("PHY manda paquete a los: " << Simulator::Now().GetSeconds()
94.                 << "s. De tamaño: " << paquete->GetSize()
95.                 << "B. Con UID: " << paquete->GetUid());
96.     NS_LOG_INFO("Posicion del vehiculo 1: " << movilidades[0]->GetPosition());
97.
98.     NS_LOG_INFO("\n");
99.
100.}
101.
102./**
103.   * Metodo que se ejecuta cuando se recibe un paquete
104.   * a nivel de fisico
105.   */
106.void
107.Observador::FinRecepPHY ( Ptr<const Packet> paquete )
108.{
109.     NS_LOG_FUNCTION(paquete);
110.     NS_LOG_INFO("PHY recibe paquete a los: " << Simulator::Now().GetSeconds()
111.                 << "s. De tamaño: " << paquete->GetSize()
112.                 << "B. Con UID: " << paquete->GetUid());
113.     NS_LOG_INFO("Tiempo: " << Simulator::Now().GetSeconds());
114.     NS_LOG_INFO("Posicion: " << movilidades[0]->GetPosition().x);
115.
116.     if (enCobertura == false) {
117.         simulacion = true;
118.         enCobertura = true;
119.         rangoX = abs(movilidades[0]->GetPosition().x
120.                         - movilidades[1]->GetPosition().x);
121.         NS_LOG_ERROR("Rango de la antena: " << rangoX);
122.         // Si es la primera vez que entramos en una zona de cobertura
```

```
123.                // empezamos a contar los parametros de simulacion
124.        if (escenarioSimulacion == 1) {
125.            // Almacenamos el momento en el que empezamos a contar
126.            tIniCobertura = Simulator::Now();
127.            NS_LOG_DEBUG("Entrando en zona de cobertura a los: "
128.                                << tIniCobertura.GetSeconds() << "s");
129.        } else if (escenarioSimulacion == 2) {
130.            tIniCobertura = Simulator::Now();
131.            posIniCoberturaV1 = movilidades[0]->GetPosition();
132.            posIniCoberturaV2 = movilidades[1]->GetPosition();
133.            NS_LOG_DEBUG("Posicion del vehiculo 1: " << posIniCoberturaV1);
134.            NS_LOG_DEBUG("Posicion del vehiculo 2: " << posIniCoberturaV2);
135.            // Calculamos la posicion a la que tiene que llegar el vehiculo 1
136.            // para que no sea capaz de enviar al vehiculo 2
137.            double a = posIniCoberturaV1.x;
138.            double b = abs(posIniCoberturaV1.x - rangoX
139.                                - posIniCoberturaV2.x);
140.            double c = abs(movilidades[0]->GetVelocity().x
141.                                - movilidades[1]->GetVelocity().x);
142.            double d = movilidades[0]->GetVelocity().x;
143.            posicionFin = a + ( b/c ) * d;
144.            NS_LOG_DEBUG("Posicion en la que el vehiculo 2 estara fuera de alcance: "
145.                                << posicionFin << "m");
146.        }
147.    } else {
148.        // Almacenamos el tiempo del ultimo paquete recibido correctamente
149.        ultimoRecibido = Simulator::Now();
150.        NS_LOG_INFO("Ultimo paquete recibido a los: "
151.                    << ultimoRecibido.GetSeconds() << "s");
152.    }
153.
154.    NS_LOG_INFO("\n");
155.}
156.
157./**
158.  * Metodo que se ejecuta cuando se tira un paquete
159.  * a nivel fisico
160.  */
161.void
162.Observador::TiraPHY ( Ptr<const Packet> paquete )
163.{
164.    NS_LOG_FUNCTION(paquete);
165.    NS_LOG_INFO("Tirado paquete en PHY: " << Simulator::Now().GetSeconds()
166.                << "s. De tamaño: " << paquete->GetSize()
167.                << "B. Con UID: " << paquete->GetUid());
168.    NS_LOG_INFO("Tiempo: " << Simulator::Now().GetSeconds());
169.    NS_LOG_INFO("PosicIon: " << movilidades[0]->GetPosition().x);
170.
171.    // Solo en el caso de que estemos en la zona de cobertura
172.        // comenzaremos a contar los paquetes que se pierden
173.    if (enCobertura) {
174.        if (escenarioSimulacion == 1) {
175.            // Damos unos 0.5s de seguridad por si no el rango calculado no fue exacto
176.            if (movilidades[0]->GetPosition().x >= distanciaBalizas*(balizas -
    1) + rangoX
177.                    && Simulator::Now().GetSeconds() -
    ultimoRecibido.GetSeconds() > 0.5) {
178.                enCobertura = false;
179.                NS_LOG_DEBUG("Saliendo de zona de cobertura a los: "
180.                                        << Simulator::Now().GetSeconds() << "s");
181.                NS_LOG_DEBUG("Posicion del vehiculo 1 al salir de cobertura: "
182.                                        << movilidades[0]->GetPosition());
183.            }
184.        } else if (escenarioSimulacion == 2) {
185.            if (movilidades[0]->GetPosition().x > posicionFin
```

```
186.                     && Simulator::Now().GetSeconds() -
    ultimoRecibido.GetSeconds() > 0.5) {
187.                 enCobertura = false;
188.                 NS_LOG_DEBUG("Saliendo de zona de cobertura a los: "
189.                                         << Simulator::Now().GetSeconds() << "s");
190.                 NS_LOG_DEBUG("Posicion del vehiculo 1 salir de cobertura: "
191.                                         << movilidades[0]->GetPosition());
192.             }
193.         }
194.     }
195.     if (enCobertura == false && simulacion == true) {
196.         // Si hemos salido de la zona de cobertura
197.         tFinCobertura = Simulator::Now();
198.         simulacion = false;
199.         NS_LOG_DEBUG("Terminando simulacion a los: "
200.                             << tFinCobertura.GetSeconds() << "s");
201.         NS_LOG_DEBUG("\n");
202.         Simulator::Stop();
203.     }
204.     NS_LOG_INFO("\n");
205. }
```