ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Trabajo Fin de Grado

Evaluación de protocolos de enrutamiento en redes ad-hoc de vehículos (VANET)

(Evaluation of Routing Protocols on Vehicular Ad-Hoc Network (VANET))

Para acceder al Título de

Graduado en Ingeniería de Tecnologías de Telecomunicación

Autor: Diego Pordomingo Sedano

Junio - 2019

E.T.S. DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACION

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

CALIFICACIÓN DEL TRABAJO FIN DE GRADO

Realizado por: Diego Pordomingo Sedano
Director del TFG: Roberto Sanz Gil
Título: "Evaluación de protocolos de enrutamiento en redes ad-hoc de vehículos (VANET)"
Title: "Evaluation of Routing Protocols on vehicular ad-hoc network (VANET) "

Presentado a examen el día:

para acceder al Título de

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Composición del Tribunal:
Presidente: Sanz Gil, Roberto
Secretario: Fanjul Vélez, Félix
Vocal: Irastorza Teja, José Ángel

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente Fdo.: El Secretario

Fdo.: El Vocal Fdo.: El Director del TFG
(sólo si es distinto del Secretario)

AGRADECIMIENTOS

Quiero agradecer a mi tutor, Roberto, la ayuda que me ha proporcionado a la hora de abordar este trabajo puesto que, era un tema que elegí yo y que no estaba muy claro cómo se iba a enfocar. Además, también quiero agradecerle la atención que me ha prestado todas las semanas para resolverme dudas y orientarme, y la rapidez con la que me ha atendido por correo.

Quiero también agradecérselo a todos mis compañeros de "Teleco", que en temporada de exámenes siempre me han animado para seguir adelante y me han ayudado con cualquier cosa que pudieran.

Me gustaría también hacer mención especial a mis amigos de Erasmus. Para mí, un año especial que nunca olvidaré por todas las experiencias vividas y viajes realizados con ellos. A pesar de que académicamente no fuera para nada sencillo, los momentos vividos con ellos, hicieron que fuera una experiencia inimaginable.

Agradecérselo también a mis amigos más cercanos. Sin sus consejos y su apoyo en muchos momentos en los que esta carrera te puede desbordar, no podría haberlo hecho.

Por último y más importante, darle las gracias a mis padres. Ellos sin duda, son mi motor y han creído en mí desde el principio pese a los altibajos. Siempre me han permitido hacer lo que más me gusta y sin ellos no sería ni habría conseguido nada. No puedo estar más agradecido.

¡Muchas gracias!

ÍNDICE

RESUMEN **6**
ABSTRACT **1**
Índice de figuras **2**
Índice de tablas. **4**
Lista de acrónimos. **5**
Motivación. **7**
Objetivos. **9**

Capítulo 1: Estado del arte. **10**
1.1 DSRC 10
1.1.1. Arquitectura 12
1.1.2. DSRC estándares básicos [3] 14
1.1.3. Capa física 15
1.1.4. Capa MAC 15
1.1.5. Upper layers 16

Capítulo 2: Análisis de los algoritmos de enrutamiento. **20**
2.1 Basados en la topología 20
2.1.1 Proactivos 20
2.1.1.1 OLSR 20
2.1.1.2 DSDV 22
2.1.2 Reactivos 23
2.1.2.1 AODV 23
2.1.2.2 DSR 24
2.2 Basados en la posición. 26
2.2.1 Location services 27
2.2.1.1 GLS 27
2.2.1.2 Quorum-Based Location Service 27
2.3 Forwarding Strategy 28
2.3.1 GPSR 28

Capítulo 3: Descripción del software Ns3 ..... **30**
3.1 Módulos Ns3 31
3.2 Namespace Ns3 31
3.3 Estructura de un script en Ns3 31
3.4 NetAnim 33

3.5 Instalación y configuración de Ns3 33

Capítulo 4: Desarrollo del proyecto 35

4.1 Simulación del script AODV y cambios realizados. 35

4.2 Configuración de los dos entornos para simulación de los algoritmos. 42

4.3 Resultados 48

4.3.1 Escenario urbano 48

4.3.2 Escenario rural 54

Capítulo 5: Conclusiones y líneas futuras. 61

5.1 Conclusiones. 61

5.2 Líneas futuras. 62

Bibliografía. 63

RESUMEN

Durante los últimos años, el número de vehículos ha ido aumentando a un ritmo desorbitado. Esto, entre otras cosas, está produciendo un aumento del tráfico que hace necesario un control sobre el mismo. Además, se han ido incorporando medidas de seguridad que, aun estando en una situación inmadura, sirven como muestra del futuro que nos aguarda en este campo.

Con todo ello, este trabajo se ha centrado en el estudio de los diferentes mecanismos que contribuyen a que un vehículo conozca en todo momento las condiciones de su entorno, avanzando hacia una conducción autónoma y ordenada para así, ganar en eficiencia y seguridad. En este caso, se han visto de forma teórica ciertos elementos pasivos que podría incorporar un vehículo para evitar y comunicar eventos producidos en la vía, así como también el tipo de infraestructura que se tendría que desplegar para ello. Junto con estos mecanismos, se han analizado con la ayuda del software ns3, diferentes algoritmos que podrían utilizarse para que la circulación fuera total o parcialmente autónoma, teniendo en cuenta una serie de parámetros con los que se intenta recrear un escenario lo más real posible. Todos estos argumentos, llevan a la unificación en la tecnología que hemos querido presentar aquí: la tecnología VANET.

Palabras clave

Tráfico, VANET, algoritmo, vehículo, ns3, tecnología, conducción autónoma.

ABSTRACT

In the last years, the number of vehicles has increased at disproportionated rhythm. This fact, among others, is producing a raising on the traffic which brings up a need to control it. Moreover, security measures, which are still immature, have been incorporated to show what is the near future in this field.

That is why, this project has been focused on the study of different mechanisms. These mechanisms contribute to full time vehicle's awareness of its surrounding conditions which lead up to an autonomous and organized driving so as to gain in efficiency and security. In this case, we have studied, in a theoretical way, some passive elements that could be incorporated to a vehicle in order to avoid and report life events on the road. We have also considered the type of infrastructure that should be deployed to carry this out. Together with these mechanisms, we have also analysed, with the Ns3 software, different algorithms that could be used to reach a full or partial autonomous traffic. To achieve that, we have taken into account some parameters that we found suitable to reproduce the most possible and realistic scenario. All these arguments lead to a unified technology which we wanted to present here: the VANET technology.

Keywords

Traffic, VANET, algorithm, vehicles, ns3, technology, autonomous driving.

Índice de figuras

Figura 1: Escenario VANET. 7

Figura 2: Primeros proyectos VANET. 8

Figura 3: Canales DSRC. 10

Figura 4: Sensores ADAS. 11

Figura 5: Platooning. 11

Figura 6: Ejemplo de OBU. 12

Figura 7: Ejemplo de RSU. 13

Figura 8: Multi-hop V2V. 13

Figura 9: One-hop V2I. 14

Figura 10: Estándares WAVE. 14

Figura 11: Channel Coordination. 16

Figura 12: ETSI ITS-G5. 17

Figura 13: Relaxed State. 18

Figura 14: Restricted State. 18

Figura 15: Máquina de estados DCC. 18

Figura 16: Ejemplo CAM. 19

Figura 17: Ejemplo DENM. 19

Figura 18: Ejemplo MPRs (1). 21

Figura 19: Ejemplo MPRs (2). 21

Figura 20: Ejemplo RREQ y RREP. 23

Figura 21: Fallo en el enlace. 24

Figura 22: Mantenimiento de ruta. 25

Figura 23: Ejemplo packet salvaging. 26

Figura 24: Jerarquía en forma de cuadrado de GLS. 27

Figura 25: Ejemplo quorum-based. 28

Figura 26: Greedy Forwarding. 29

Figura 27: Organización Ns3. 30

Figura 28: Simulación en NetAnim. 33

Figura 29: Configuración VM Ubuntu. 35

Figura 30: Líneas para poder ejecutar NetAnim. 36

Figura 31: Archivo wscript. 36
Figura 32: Declaración de la clase AodvExample. 37
Figura 33: Parámetros de simulación. 37
Figura 34: Configuración AodvExample. 38
Figura 35: Posicionamiento grid. 38
Figura 36: Movilidad aplicada. 40
Figura 37: Simulación de la movilidad en NetAnim. 40
Figura 38: Configuración capa MAC y física. 41
Figura 39: Función “InternetStack”. 41
Figura 40: Ejecución del ping. 42
Figura 41: Distancia para los dos entornos. 44
Figura 42: Ejemplo de parámetros que se pueden cambiar en la consola. 44
Figura 43: Movilidad. 45
Figura 44: Modelo de dos rayos. 47
Figura 45: fdp de Nakagami. 47
Figura 46: PDR vs NodeDensity @ 7 m/s. 49
Figura 47: PDR vs NodeDensity @ 12 m/s. 49
Figura 48: PDR vs NodeDensity @ 16 m/s. 50
Figura 49: OverHead vs NodeDensity @ 7 m/s. 51
Figura 50: OverHead vs NodeDensity @ 12 m/s. 51
Figura 51: OverHead vs NodeDensity @ 16 m/s. 52
Figura 52: GoodPut vs NodeDensity @ 7 m/s. 52
Figura 53: GoodPut vs NodeDensity @ 12 m/s. 53
Figura 54: GoodPut vs NodeDensity @ 16 m/s. 53
Figura 55: PDR vs NodeDensity @ 500 metros. 55
Figura 56: PDR vs NodeDensity @ 1000 metros. 55
Figura 57: PDR vs NodeDensity @ 2000 metros. 56
Figura 58: OverHead vs NodeDensity @ 500 metros. 57
Figura 59: OverHead vs NodeDensity @ 1000 metros. 57
Figura 60: OverHead vs NodeDensity @ 2000 metros. 58
Figura 61: GoodPut vs NodeDensity @ 500 metros. 59
Figura 62: GoodPut vs NodeDensity @ 1000 metros. 59
Figura 63: GoodPut vs NodeDensity @ 2000 metros. 60

Índice de tablas.

Tabla 1: Escenario Urbano. 48

Tabla 2: Escenario Rural. 54

Tabla 3: Conclusiones. 62

Lista de acrónimos

ABS	Antilock Brake System
ADAS	Advanced Driver Assistance System
AODV	Ad-Hoc On-Demand Distance Vector
BPSK	Binary Phase Shift Keying
BSM	Basic Safety Message
CAM	Cooperative Awareness Messages
CCH	Control Channel
CDMA	Code Division Multiple Access
CSMA	Carrier Sense Multiple Access
DSDV	Destination-Sequenced Distance Vector
DSR	Dynamic Source Routing
DSRC	Dedicated Short Range Communication
EDCA	Enhanced Distributed Channel
ESP	Electronic Stability Program
FDMA	Frequency Division Multiple Access
GPS	Global Positioning System
GPSR	Greedy Perimeter Stateless Routing
GLS	Grid Location Service
ISI	InterSymbol Interference
LTE	Long Term Evolution
MIB	Management Information Base
OBU	On-Board Unit
OFDM	Orthogonal Frequency Division Multiplexing
OLSR	Optimized Link State Routing
PDR	Packet Delivery Ratio
QAM	Quadrature Amplitude Modulation
QOS	Quality Of Service
RSU	RoadSide Unit
SDN	Software Defined Network
SCH	Service Control Channel
TDMA	Time Division Multiple Access
V2I	Vehicle To Infrastructure

V2V	Vehicle To Vehicle
VANET	Vehicular Ad-hoc Network
WAVE	Wireless Access for Vehicular Environment
WiFi	Wireless Fidelity
WSA	WAVE Service Advertisement

Motivación

VANET es una tecnología que está en constante crecimiento y que está convirtiéndose en un campo de gran interés para los investigadores. La comunicación entre vehículos (V2V) y entre vehículos e infraestructura (V2I) está ganando gran popularidad a la hora de buscar más seguridad en las carreteras utilizando la comunicación entre sistemas. A pesar de todos los elementos pasivos y activos de seguridad con los que cuenta un vehículo hoy en día, como pueden ser el ABS o el ESP, el número de accidentes en las carreteras no desciende. Esto es debido, entre otras cosas, a que existe un factor humano que hace que, por error de él, se produzcan dichos accidentes. Algunos estudios han demostrado que el 60% o más de accidentes, se podrían haber evitado si se hubieran implementado sistemas de aviso mediante mensajes entre vehículos para avisar del suceso. Estos sistemas, combinados con sistemas electrónicos digitales como la asistencia de frenada u otros, hacen que este campo sea de gran interés para intentar alcanzar una casi total seguridad en la circulación. Además, con el aumento de vehículos en las carreteras, se necesitan una serie de mejoras en el aprovechamiento de las mismas y también en el ahorro de combustible. Por ello se quieren implementar técnicas como el platooning.

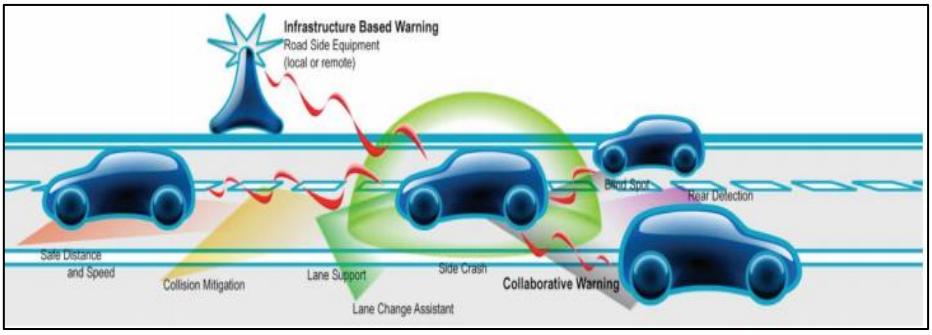


Figura 1: Escenario VANET [1].

El uso de tecnologías inalámbricas aplicadas a vehículos ha sido objeto de estudio desde los años 70. Uno de los proyectos pioneros fue el Sistema Integral Automovilístico de Control de Tráfico desarrollado por el MITI (Ministerio de Industria y Comercio Internacional) de Japón en el 1973. Su objetivo era reducir la congestión de tráfico y disminuir el número de accidentes que se producían en su tránsito dando a los conductores información de las rutas y asistencia de emergencia en caso de necesidad. En 1986 se llevó a cabo el proyecto PROMETHEUS (PROgraMme for European Traffic with Highest Efficiency and Unprecedented Safety) constituido por 19 países de Europa. Este proyecto hizo que se impulsara la investigación de la tecnología de comunicación radio de corto alcance que luego se convertiría en uno de los elementos más importantes para la implementación de estos sistemas y que hoy conocemos como DSRC (Dedicated Short Range Communication). En 1990, el concepto de VANET empezó a tomar importancia en el ámbito de las telecomunicaciones, así como los sistemas de posicionamiento como el GPS y dispositivos inalámbricos de bajo coste, que favorecieron el avance de las redes vehiculares.

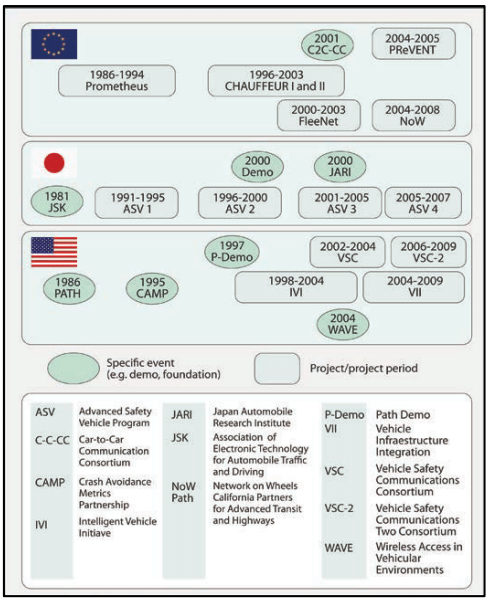


Figura 2: Primeros proyectos VANET [2].

El concepto de red de vehículos fue propuesto, por primera vez, por Delphi Delco Electronics Systems e IBM Corporation en el año 1998. Posteriormente surgieron proyectos en Europa como por ejemplo CHAUFFEUR que conformaron un punto de partida para definir estándares para las comunicaciones y diseño de redes vehiculares. Como vemos en la Figura 2, en 2004 empezaría el proyecto más actual llamado WAVE o habitualmente referido como DSRC. Dicha tecnología, que explicaremos más adelante, usa frecuencias en torno a 5.9 GHz con el objetivo de tener una latencia muy pequeña (alrededor del milisegundo), usar poca potencia y tener alta fiabilidad ya que no puede haber error en el envío de las alertas u otros mensajes que se produzcan entre vehículos.[2]

Está claro que la mejora que se puede hacer actualmente es muy amplia. Desde hace pocos años, se empiezan a fabricar coches autónomos y se van puliendo ciertos mecanismos que todavía pueden ser algo inseguros. También, van saliendo más vehículos al mercado que son híbridos o totalmente eléctricos y que preparan el cambio, que seguramente no tarde en producirse, del vehículo de combustión al eléctrico. Por ello, en este trabajo vamos a abordar ciertos aspectos de la tecnología VANET, centrándonos en los algoritmos de enrutamiento que pueden llevar a cabo los mensajes enviados por los vehículos, buscando los más eficientes y fiables.

Objetivos

El objetivo principal de este proyecto es estudiar el funcionamiento de diferentes algoritmos de enrutamiento aplicados a las redes de vehículos, así como otros parámetros y estructura de la tecnología WAVE.

De forma más específica vamos a hablar de los siguientes puntos:

- Conceptos básicos de la tecnología VANET, historia y estado del arte.
- Análisis teórico de algunos tipos de algoritmos de enrutamiento entre los cuales se encuentran proactivos y reactivos.
- Conocer el funcionamiento del software ns3 y Netanim como principales herramientas de este proyecto analizando las librerías y los módulos utilizados para su realización.
- Análisis práctico, mediante simulación, con el software ns3 de algoritmos proactivos y reactivos.

Capítulo 1: Estado del arte

La conectividad entre vehículos puede ser considerada como una aplicación con futuro muy cercano ya que, como hemos dicho anteriormente, ya existen coches con conducción autónoma y este desarrollo va a tender a una automatización de este sector. Esto a su vez, añade más valor a la industria del mercado del automóvil. Como hemos comentado antes, uno de los primeros hitos en este campo, fue la estandarización de DSRC.

1.1 DSRC

Dedicated Short Range Communication (DSRC) es un protocolo diseñado para comunicaciones a alta velocidad para VANET y, por lo tanto, para redes con baja latencia, y está considerado actualmente como la tecnología más prometedora. En este sentido, la única problemática que podría causar esta tecnología es la necesaria mejora del vehículo en tema de hardware para adaptarlo. Opera en la banda de 5.9 GHz y con un ancho de banda de 75 MHz. El espectro está dividido en canales de 10 MHz con una banda de guarda de 5 MHz. [4]

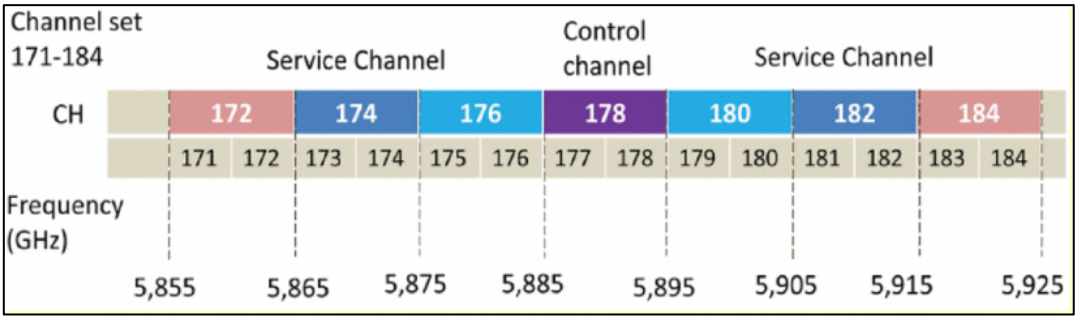


Figura 3: Canales DSRC. [5]

Permite un rango de conectividad de hasta 1000 metros. La principal motivación para su desarrollo fue la de evitar colisiones mediante el envío de datos y advertencias entre vehículos. Cada vehículo equipado con esta tecnología realiza un broadcasting de cierta información básica como, por ejemplo: Posición mediante GPS, velocidad y aceleración. Por otra parte, en cuanto al envío y recepción de mensajes de alerta (llamados BSM) estos se utilizan para medir la posición de los "vecinos" y ver si pueden ser una amenaza o no de colisión. Si un vehículo determina que hay posibilidad de colisión o de otro peligro, el sistema OBU puede avisar al conductor e incluso tomar control del coche si fuera necesario. Este tipo de aviso al conductor puede ser del tipo visual mediante avisos en la pantalla de control de visualización, sonoros o táctiles. Estos avisos se llevan a cabo mediante sensores emplazados en el coche que se comunican con él y con la OBU. Este tipo de seguridad también lleva una tecnología que se ocupa de la asistencia al conductor llamada ADAS (Advanced Driver Assistance System). Otra forma técnica de llamar a este mecanismo es: Cooperative perception.

Figura 4: Sensores ADAS.[3]

Por otra parte, también podemos recibir mensajes de seguridad por parte de la RSU como por ejemplo el estado de las señales en una intersección o la existencia de algún peligro puntual como por ejemplo hielo en la calzada o un vehículo averiado, que pueda forzar un desvío en la ruta del coche. Esta "relación" entre vehículo e infraestructura se le llama: cooperative maneuvering. Sin embargo, no solo usamos esta tecnología para propósitos de seguridad, sino que también hay otros propósitos como pueden ser la asistencia de navegación, realización pagos de peajes o gasolina y sobre todo mejorar la eficiencia de las carreteras y ahorrar combustible con ello. En cuanto al ahorro del combustible y de la eficiencia, existe una tecnología bastante eficaz y con mucha relación con la comunicación entre vehículos llamada "platooning".[3]

Platooning (pelotón) es un grupo de vehículos que circulan de manera unida, como si fuera un tren con vagones, que se comunican entre sí parámetros de velocidad, distancia de seguridad y destino de cada coche. Con esto reducimos la distancia entre vehículos, consumo de combustible, seguridad en la carretera y aprovechamiento de ella. Como se puede apreciar, hay cierta similitud con el esquema de una carrera ciclista.[3]

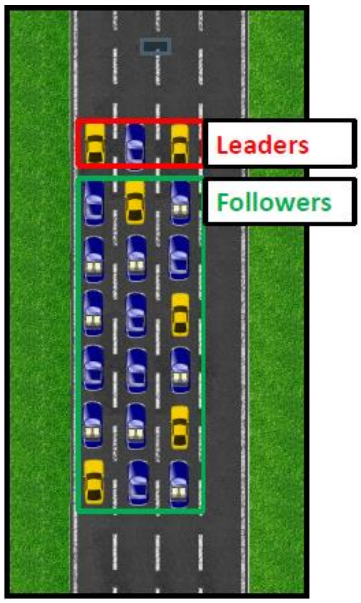


Figura 5:Platooning.[3]

Como hemos dichos, los vehículos se informan entre otras cosas de la dirección de destino o de advertencias. Los tipos de mensajes enviados son los siguientes:[3]

- Join/Leave: Muestra intención de formar parte de un pelotón o de abandonarlo y si desea ser leader o follower.
- Announcement/warning: Un vehículo que no tenga constancia de la existencia de ese pelotón tiene que saber que existe.
- Group communication: Mensajes enviados con objeto de mantener cierta coordinación entre todos los miembros del pelotón.

En el ejemplo de la Figura 4, si tomamos como referencia que estamos en una carretera a unos 100-120 km/h, necesitaremos una latencia máxima entre vehículos de 10 ms y de 500 ms entre vehículo e infraestructura.

1.1.1. Arquitectura

En cuanto a la arquitectura, está formado por dos partes: La OBU (On-Board Unit) y la RSU (Road Site Unit). [6]

La OBU es un sistema integrado en el vehículo que está conectado tanto a la red DSRC como a la red interna del vehículo. Su principal función es la de dar conexión vehículo a vehículo (V2V) y recoger mensajes del resto de elementos de la infraestructura. En la OBU podemos distinguir distintos elementos:

- Mobile Router: Encargado de las comunicaciones de vehículo tanto interna como externa. El sistema está formado por módulos en Java que interactúan entre sí en tiempo real.
- Vehicle Host: Se puede definir como el ordenador de abordo que controla todos los elementos del coche.
- Vehicle Gateway: Es la pasarela entre el host del vehículo y los sensores, y elementos de control del coche.
- Human Media Interface: Es una entidad opcional que se conecta al sistema de forma inalámbrica mediante el Mobile Router y descarga datos del vehículo para que el usuario pueda tener un tracking de él.
- GPS: Lleva incorporado un sistema de localización del vehículo.

Figura 6: Ejemplo de OBU.

El otro sistema de la arquitectura es la RSU (RoadSide Unit). Como hemos dicho provee de una conexión inalámbrica de tipo DSRC para comunicarse con las OBUs. También lleva insertada una unidad de seguridad de datos, una unidad GPS para posicionamiento y una conexión LTE para carga y descarga de datos, así como para comunicación con otras RSUs.

Las características más importantes para recalcar de esta unidad son:[7]

- Por supuesto, alta velocidad de transmisión y baja latencia.
- Ethernet para transmisión de datos en red cableada.
- Hot spot WiFi para comunicación con dispositivos como portátiles o Smartphones que estén en las proximidades.
- Backhaul LTE.
- Software opcional para formar un sistema central de control de varias RSUs.

Figura 7:Ejemplo de RSU. [7]

Dentro de la arquitectura también podemos diferenciar dos formas de comunicación: Comunicación vehículo a vehículo (V2V) o comunicación vehículo a infraestructura (V2I).

- Comunicación V2V: Son comunicaciones de bajo coste y que no tienen necesidad de infraestructura. Apropiadas para aplicaciones de seguridad donde los vehículos realizan broadcasting de advertencias. Son, en resumen, redes ad-hoc multisalto.[3]

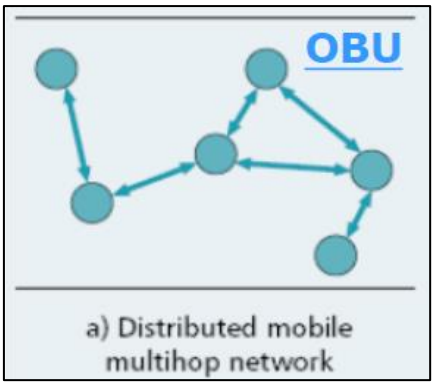


Figura 8:Multi-hop V2V. [3]

- Comunicación V2I: Conexión de corta duración provocada por la velocidad de las OBUs y que necesita alta velocidad de transferencia para que haya envío de información. Estos sistemas se pueden asemejar a una red centralizada one-hop y se establece un mecanismo de handover entre RSUs. [3]

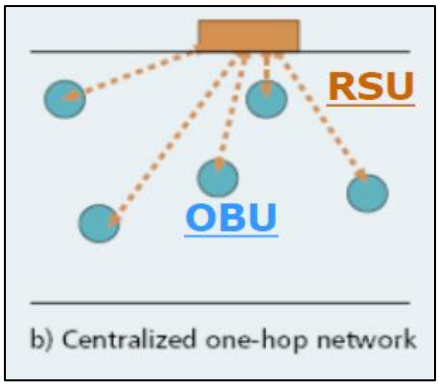


Figura 9: One-hop V2I. [3]

1.1.2. DSRC estándares básicos [3]

En la tecnología DSRC se creó un estándar que agrupaba las especificaciones de capa física, capa MAC y capas más altas como aplicación o sesión. Este estándar se llama WAVE (Wireless Access for Vehicular Environment) o también llamado IEEE p1609. Este estándar acoge las especificaciones para entornos donde la conectividad dura poco tiempo, y que, por lo tanto, la información se transmite en pequeños intervalos de tiempo.

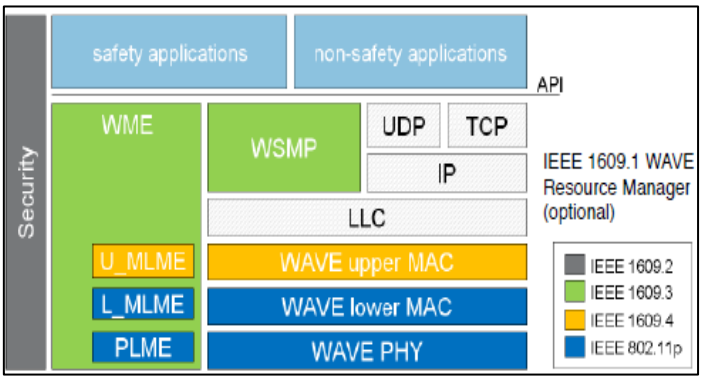


Figura 10: Estándares WAVE.

Como podemos ver en la Figura 10, este grupo de estándares fundamentalmente se divide en dos partes: Plano de gestión y plano de datos. Recorriendo un poco la pila de estándares vemos que tanto la capa física como la capa MAC tienen un protocolo WiFi orientado a vehículos 802.11p. Si subimos de capa encontramos el protocolo 1609.4 que se ocupa de la coordinación, sincronismo y enrutamiento de canales, así como de la prioridad del usuario. El 1609.3 define las operaciones a nivel de red y transporte (capa 3 y 4 OSI). Esto servicios están diseñados para soportar comunicaciones de tipo WAVE y se ocupa de solicitud de servicio, entrega de datos de gestión, configuración IPV6 y mantenimiento de la MIB. El estándar 1609.2 se ocupa de la seguridad de las comunicaciones en el entorno WAVE para aplicaciones y procesos ejecutados en las distintas capas. Estos servicios se ocupan del procesamiento y gestión seguros. Por supuesto, como todo servicio de seguridad, genera datos cifrados y firmados y hace un gran uso de la criptografía. Esta parte del estándar es de las más importantes y una en las que más hincapié se está haciendo actualmente, ya que una brecha de seguridad en cualquier equipo puede ocasionar un accidente.

1.1.3. Capa física

Como vemos en la Figura 10, el protocolo que se ocupa de la capa física es el IEEE 802.11p. Está basado en la capa física de 802.11a, es decir, implementa OFDM (Orthogonal Frequency Division Multiplexing), sin embargo, el ancho de banda es de 10 MHz para que tenga más robustez y más resistencia al jamming. La no utilización del ancho de banda de 5 MHz se debe a que, aunque pueda combatir los efectos del multipath, el tamaño de los paquetes es demasiado largo empeorando el tiempo de coherencia. Trabaja en el rango de frecuencias de 5.85-5.925 GHz y tiene unas velocidades que pueden alcanzar los 3 Mb/s en modulación BPSK y 27 Mb/s con modulación 64 QAM. Los cambios que tiene la norma 802.11p respecto de la 802.11a son para evitar la ISI provocado por el delay multicamino y adecuar el efecto Doppler a las comunicaciones vehiculares. En cuanto a las potencias de transmisión, tenemos valores desde los 15 hasta los 25 dBm dependiendo del entorno al que nos enfrentemos.

Existen dos escenarios: El escenario rural en el que hay poca densidad de vehículos y no existe el fading provocado por los edificios, y el entorno urbano, en el que la densidad de coches puede ser alta y por lo tanto el número de interferencias también lo es. Además, hay que sumar la interferencia causada por las reflexiones y difracciones de los edificios. En el caso de un entorno rural, el rango de transmisión es mayor que en el del entorno urbano puesto que en el urbano no hay mucha distancia entre dos coches y además sería perjudicial por un exceso de interferencias. Las distancias típicas a las que se transmiten en los dos escenarios es de aproximadamente 400 metros en zona rural y en torno a 140 metros en zona urbana, respectivamente. Sin embargo, la tecnología está prepara para transmitir a casi 1km.

El estándar WAVE usa el concepto de multi-channel que puede ser usado tanto para seguridad como para mensajes de información. Usa diferentes canales según la prioridad del mensaje enviado. Estos son 7 canales de 10 MHz que permiten tanto modo de un canal como multicanal. Hay dos tipos de canales que van alternados. El CCH (Control Channel) y el SCH (Service Control Channel). Puesto que se necesita poco delay, el intervalo en el que se usan un canal CCH y otro SCH no debe superar 100 ms.

1.1.4. Capa MAC

Las redes de vehículos son redes con estructura cambiante debido a la movilidad de sus nodos. Por ello, se hace complicado de desarrollar un esquema MAC que recaiga en un control central. En los casos donde hay un control central, se puede aplicar TDMA, FDMA o CDMA ya que hay un nodo responsable de compartir los recursos entre los usuarios y, por lo tanto, garantizando QoS para tráfico importante. Esto en VANET no es implementable debido a la alta movilidad de los nodos y que el nodo "central" no quedaría fijo y, por tanto, requeriría una constante negociación entre nodos. Esto acarrearía un aumento en el delay y una vez que la negociación está completada, seguramente esa información ya sería obsoleta. Por ello lo mejor sería implementar un control CSMA que no requiere un control central y que se dedica a escuchar el canal, ver si está libre y si puede transmitir o, por el contrario, tiene que esperar un tiempo aleatorio para evitar que se produzca una colisión. Sin embargo, esto tiene un gran problema y es que un nodo puede experimentar constante delay a causa de que el canal esté siempre ocupado. Por eso la capa MAC en WAVE es equivalente a la extensión del estándar 802.11e Enhanced Distributed Channel Access (EDCA) Quality of Service (QoS). Siguiendo con este esquema, los mensajes están categorizados con diferentes niveles desde AC0 con el menor nivel hasta AC3 que tiene la prioridad más alta. Cuando se transmite un paquete, primero hay una contienda interna entre ACs para ver quien transmite primero. Luego ese paquete seleccionado, hace otra contienda externa en el canal usando los parámetros seleccionados. Estos parámetros son la Contention Window mínima (CWmin), la CWmáx, los Arbitration Inter-Frame Space (AIFS) y el tiempo de espera (Tw).

Siguiendo con eso tenemos que el channel coordination entre CCH y SCH se produce de esta forma:

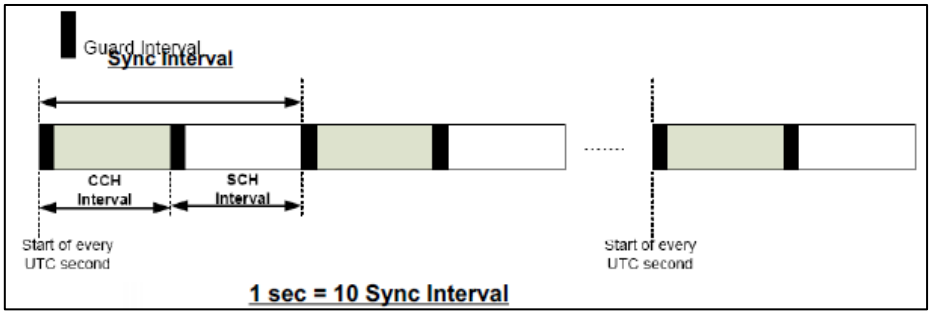


Figura 11:Channel Coordination.

Los nodos se sincronizan temporalmente con el UTC cada 1 segundo a través de GPS con una precisión de nanosegundos. Sin embargo, durante pequeños intervalos cada nodo tiene que recurrir a su reloj de cuarzo interno. A continuación, vamos a nombrar y explicar algunos problemas que tiene este tipo de MAC:

- CSMA/CA implica tener retardos impredecibles en la recepción de mensajes, en especial cuando las condiciones son de alta carga de tráfico.
- Cuantos más contendientes haya en el canal, mayor es la probabilidad de colisión. En este caso podemos referirnos a la generación de mensajes de seguridad que varios nodos pueden tener al mismo tiempo.
- Como hemos comentado TDMA, FDMA o CDMA son implementable a causa del dinamismo de los escenarios, sin embargo, debido a la obtención de sincronismo mediante GPS, podemos decir que se trata de un esquema MAC basado en TDMA. Esto último todavía no ha sido incluido en el estándar.

Posibles soluciones a todo esto:

- Evitar el reenvío de un paquete si su ID ya ha sido visto por ese nodo.
- Solo los nodos más alejados del transmisor realizan retransmisiones de los paquetes con objeto de reducir la latencia.
- Los nodos se pueden agrupar en clústeres y solo el cluster-head puede retransmitir. El problema de esto es el gran overhead que implica la creación de un cluster.
- Retardo aleatorio de transmisiones para evitar colisiones.

1.1.5. Upper layers

Siguiendo con la pila de protocolos WAVE tenemos el estándar IEEE P1609.4. Este estándar describe operaciones multicanal en un entorno inalámbrico para 802.11p. Define entre otras cosas:

- Tiempo de sincronización de los canales CCH y SCH.
- Enrutamiento de canales, es decir, selección del canal y de la prioridad AC adecuados.
- Servicios de gestión para el switching de canales.
- Operación multicanal.

Este estándar es capaz de funcionar en 3 modos:

1. Acceso continuo al canal.
2. Acceso alterno entre los dos canales.
3. Acceso inmediato al canal.

Por encima tenemos el estándar IEEE P1609.3 que provee de direcciones y se ocupa del transporte de datos. Define los servicios que operan en la red y las capas de transporte que dan conectividad a la red entre vehículos y entre infraestructura y vehículos. Hay distintos tipos de tráfico:

- Tramas de gestión que envían WAVE Service Advertisement (WSA). Estos mensajes son enviados mediante el protocolo WSMP. Estos mensajes solo se envían por el canal CCH.
- Tramas de datos como datagramas IP y solo por el canal SCH.

WSMP permite controlar de forma directa las características físicas por ejemplo el número de canal o la potencia transmitida para el envío de los mensajes. También provee con el ID de la aplicación del servicio que se trata llamado PSID.

Como última capa tenemos dentro de la MAC la P1609.2 que se usa para dar seguridad y se ocupa del procesado de los mensajes. Desde mayo 2018, se sigue la regulación General Data Protection Regulation (GDPR), la cual insta que todos los mensajes tienen que ir encriptados. Para ello hay 5 propiedades que se tienen que cumplir, para "asegurar" una ausencia de brechas de seguridad.

- Autenticidad: Asegurarse de que el emisor es quien dice ser.
- Autorización: Prueba de que el emisor tiene los privilegios para poder enviar al receptor.
- Integridad: Cerciorarse de que cualquier cambio en el mensaje pueda ser detectado.
- No rechazo: Terceras partes puedan demostrar las 3 reglas anteriores.
- Confidencialidad: Asegurar que la única persona que puede leer el contenido es a la persona a la que se le envía dicho contenido.

El estándar 1609.2 usa tanto Criptografía de curva elíptica de 256 bits de clave, la cual es más eficiente que el RSA, como también AES-CCM para criptografía simétrica e integridad.

Por último, tendríamos la capa de aplicación la cual tiene 16 tipos de mensajes. El más importante de ellos es el Basic Safety Message (BSM), que es transmitido con una frecuencia de 10 Hz. Se usa tanto como una especie de "latido" de cada vehículo, en el que envía datos como posición, velocidad o dirección, como también mensajes de advertencia de eventos cuando es necesario.

La IEEE no es el único organismo que propone protocolos como solución para las redes de vehículos. El estándar ETSI también los propone para cada capa de WAVE, aunque en este caso, solo cambian las capas más altas de ella en comparación con las del IEEE. Propone ITS-G5 que ahora detallaremos.

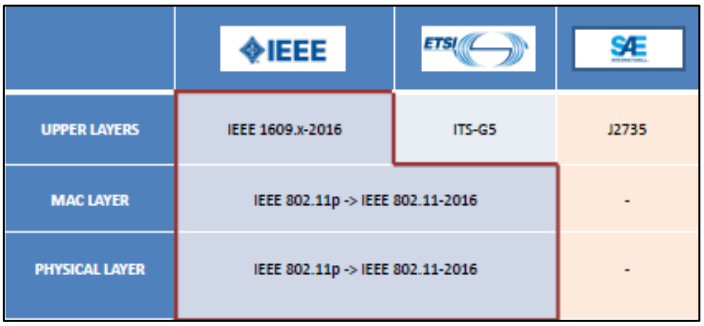


Figura 12:ETSI ITS-G5.

ITS-G5 introduce una asignación de canales más específica compuesta por 4 canales: Uno para seguridad, otro en el que no se envíen tramas de seguridad, un tercero para infraestructura y un cuarto y último para futuras aplicaciones de la ITS.

Implementa también un sistema de control de congestión descentralizado (DCC), en el que se pueden configurar parámetros en el AC de potencia, sensibilidad del receptor y velocidades binarias. Este sistema tiene como objetivo:

- Una asignación de recursos y un acceso al canal equilibrado y justo para todos los vehículos que estén en la misma zona de comunicación.
- Carga baja de mensajes en el canal.
- Reserva de recursos para eventuales mensajes que conlleven una alta prioridad.

Implementación de un sistema de máquina de estados en el que sus estados sean: inactivo, activo y restrictivo. Esto se traduce en que cada estado tiene parámetros asociados a él, entonces nodos que están situados en áreas con alta densidad de vehículos van a utilizar un nivel de potencia reducido ya que el rango de comunicación es pequeño (Restricted State).

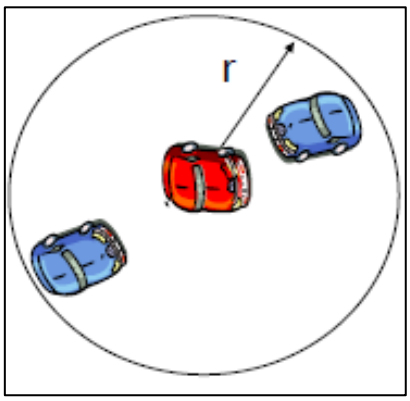


Figura 13: Relaxed state.

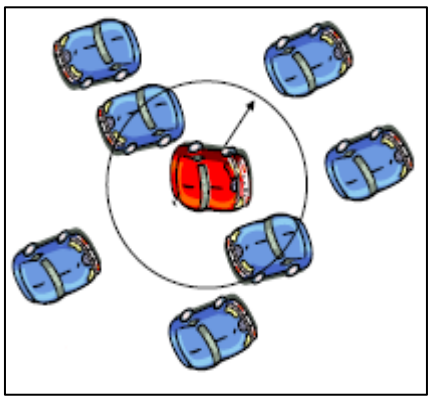


Figura 14: Restricted State

Los cambios de estados en la máquina funcionan de acuerdo al ratio de ocupación del canal (CBR). Si la carga del canal es mayor del 15%, el canal pasaría de inactivo a activo. Si en ese estado es mayor del 40% pasaría a restrictivo. Lo mismo pasaría a la inversa, con los mismos porcentajes.

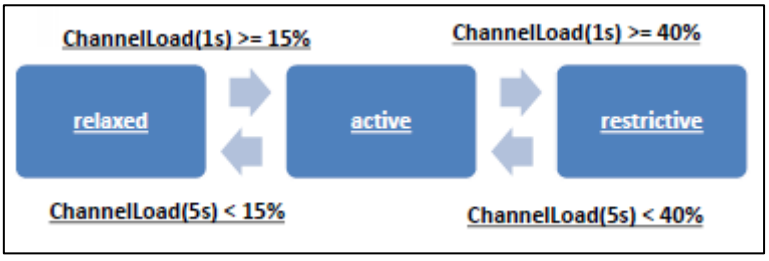


Figura 15: Máquina de estados DCC.

Antes de la capa de aplicación, tenemos una capa llamada Facilities, compuesta por 3 partes:

- Information support.
- Communication support.
- Application support.

En application support tenemos dos tipos de mensajes: Mensajes periódicos de tipo CAM (Cooperative Awareness Messages) por el canal CCH que se mandan a 1-10 Hz y llevan la posición del nodo e información de seguridad, y mensajes de tipo Event-driven DENM (Descentralized Environmental Notification Message), que llevan información de riesgos que se pueden producir en la carretera y notificación de eventos en general.

En el ejemplo de la siguiente Figura, el coche detecta que se aproxima una motocicleta y reacciona para no seguir con su trayectoria y evitar un choque. Sin este mecanismo, el coche no habría podido reaccionar ya que el camión le quita la visibilidad necesaria para cruzar con seguridad. Por ello también hay ciertas normas para el envío de CAMs las cuales son comprobadas cada 100ms.

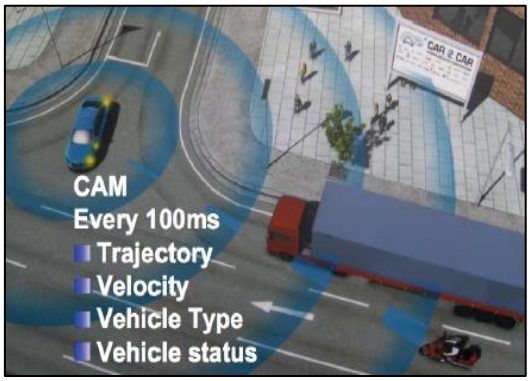


Figura 16: Ejemplo CAM.

En el ejemplo siguiente de DENM, el coche rojo ha tenido un accidente contra una farola. En el momento en el que se produce el accidente, envía un broadcast a vehículos que estén en su rango de comunicación para informarles y evitar que se produzcan otros. Otros casos en los que se utilizan este tipo de mensajes pueden ser, por ejemplo, cuando hay obras o en una frenada de emergencia.

Figura 17: Ejemplo de DENM.

El information support gestiona LDM (Local Dynamic Maps) para aprendizaje cooperativo e información recibida a través de mensajes CAM o DENM. Por último, el communication support se ocupa de opciones de interoperabilidad futura para redes IPv6 o redes de comunicaciones móviles como LTE.

Capítulo 2: Análisis de los algoritmos de enrutamiento

Estos algoritmos nacen de la necesidad de solucionar algunos problemas que se presentan en las comunicaciones entre nodos y por ello, existen distintos tipos según queramos optimizar ciertos aspectos que pueden ser importantes para nuestra configuración de red. Algunos de los problemas se basan en: cambios en la topología debidos a la movilidad de los nodos que hace que se pueda perder la comunicación, formación de ciclos provocados por el fallo de algún enlace, enlaces asimétricos, ancho de banda reducido, alto consumo de energía etc. Todos estos problemas en el enrutamiento hacen que tengamos que idear soluciones en las que el trade-off tiene que ser positivo para nuestro sistema. Por ello podemos distinguir dos tipos de algoritmos: Basados en topología o basados en posicionamiento.

2.1 Basados en la topología

Dentro de los basados en la topología tenemos dos tipos: Proactivos y Reactivos.

Los Proactivos realizan un intento de mantener actualizada la información de cada nodo hacia el resto de los nodos de la red. Para ello realiza una propagación por toda la red de los cambios producidos en ella. Es un tipo de protocolo muy adecuado para tráfico inalámbrico ya que puedes enviar información a cualquier nodo en cualquier momento. Estos protocolos están basados en enrutamiento del vector-distancia y su uso es apropiado cuando los tiempos de llegada de los paquetes son importantes ya que tiene baja latencia frente a un alto overhead. [19]

Los Reactivos tienen la particularidad de que solo se crean las rutas cuando la fuente lo desea. Para ello se implementa el Route Discovery Protocol, el cual termina cuando una ruta se ha descubierto o cuando todas las posibles rutas de comunicación han sido examinadas. Una vez se ha descubierto la ruta, dicha ruta es mantenida usando el Route Maintenance Protocol. Este tipo de protocolos son más apropiados para un entorno móvil el cual está limitado en ancho de banda ya que tienen bajo overhead.

- Proactivos: DSDV, OLSR.
- Reactivos: DSR, AODV.
-

2.1.1 Proactivos

2.1.1.1 OLSR

Protocolo basado en el estado del enlace inventado por Jacquet en el año 2000. Este protocolo hereda la estabilidad de un protocolo con un algoritmo orientado al estado del enlace y tiene la ventaja de tener las rutas inmediatamente operativas cuando se necesitan debido a su naturaleza proactiva. Minimiza el overhead usando el método de flooding para enviar paquetes de control y avisar del estado de su enlace. Esa información es diseminada por toda la red y cada nodo elige una serie de nodos vecinos y los utiliza como nodos MultiPoint Relays (MPR). En OLRS solo los nodos seleccionados como MPRs tienen la responsabilidad de propagar todo el tráfico de control

por la red. El mecanismo de flooding que utilizan es muy eficiente gracias a que reducen el número de retransmisiones requeridas. Estos nodos declarados como MPRs también tienen la responsabilidad de mostrar el estado del resto de los enlaces de la red a los nodos que les han declarado MPRs. Para poder diferenciar un MPR de un nodo normal, los MPRs mandan una información adicional en sus paquetes de control en el que anuncian que son MPRs y los nodos a los que sirve de MultiPoint Relay. Un nodo selecciona como MPR a otro nodo cuando está a la distancia de un salto, y cuando el nodo al que apunta tiene links simétricos y bidireccionales. Este mecanismo evita el problema de la transferencia de paquetes por enlaces unidireccionales que provocan una ausencia de confirmaciones, provenientes de la capa de enlace, a cada salto. A continuación, vamos a poner un ejemplo de MPRs: [19][8]

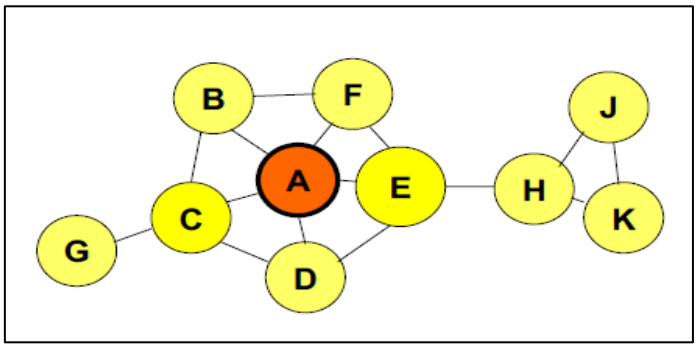


Figura 18: Ejemplo MPRs (1). [19]

En este grafo, todos los nodos son bidireccionales. Los nodos C y E son MultiPoint Relays de A. C selecciona A como MPR, mientras que E selecciona A y H. El nodo E es MultiPoint Relay también de H. Si el enlace H-J fuera unidireccional, K también sería seleccionado como MPR por H. Por eso, OLSR nunca usa links unidireccionales. La idea de los MPR es minimizar la carga de mensajes decrementando la redundancia.

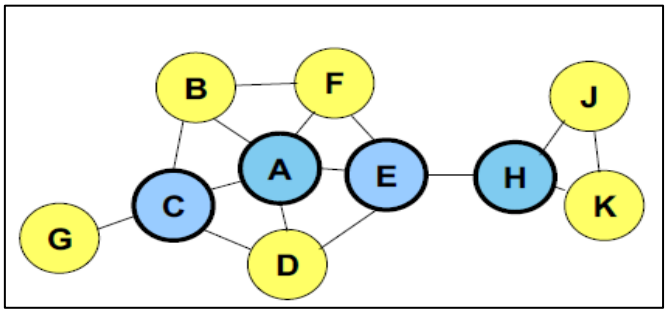


Figura 19: Ejemplo MPRs (2). [19]

OLSR está particularmente pensado para redes densas. En redes poco densas, cada nodo vecino es un MultiPoint Relay y OLSR se reduce a un protocolo de estado de enlace simple.

Para que un nodo “x” seleccione su MPR, primero tiene que formar una neighbor table. Para ello realiza un broadcast de mensajes tipo Hello a los nodos que tiene a un salto. Después, cada hello enviado contendrá el 1-hop del nodo anterior haciendo que cada nodo pueda construir una tabla con cada 1-hop y, por consiguiente, sabiendo si el 1-hop al que apuntan tiene otros nodos enlazados o no (lo que sería el 2-hop node). “X” selecciona su MPR entre los nodos que tiene a

un salto y que a su vez contiene más nodos a 2 saltos de él. En el siguiente mensaje hello que envíe, notificara su nuevo MPR. [8]

Cada nodo mantiene una tabla de enrutamiento la cual está construida en base a la tabla de topología y la tabla de vecinos. Las partes de la tabla de enrutamiento de cada nodo son:

- Dirección del posible nodo destino.
- Next hop para llegar al nodo destino.
- Distancia estimada del nodo destino.

Hay que tener en cuenta que estos datos varían dinámicamente ya que la tabla de enrutamiento es recalculada constantemente.

Formato de trama

Centrándonos un poco en el formato de un paquete OLSR, podemos diferenciar dos partes: La cabecera el paquete y la cabecera del mensaje. En la cabecera del paquete tenemos el tamaño del paquete y el número de secuencia. El número de secuencia (PSN) debe ser incrementado uno cada vez que un nuevo paquete se transmite. Además, también se mantiene un PSN para cada interfaz y así los paquetes transmitidos a esa interfaz son enumerados. [8]

Por otra parte en la cabecera del mensaje tenemos el tipo de mensaje, el tiempo de validación que indica cuanto tiempo se le puede dar de validez a la información de un mensaje antes de que se quede anticuado; tenemos el tamaño del mensaje, el nodo que ha originado el mensaje (que es diferente del nodo del cual lo hemos recibido), el time-to-live, que contiene el máximo número de saltos que un mensaje será transmitido, el contador de saltos (hop count) y el Message Sequence Number (MSN), que es análogo al PSN, pero en este caso también se usa para asegurarse de que un mensaje no es retransmitido más de una vez por un nodo. [8]

Reenvío de mensajes

En el algoritmo OLSR, está implícito el procedimiento de envío de mensajes para que la recepción de ellos por los nodos sea adecuada y en caso de no serlo, reenviarlos. Sin embargo, para poder tener un poco más de control sobre la actuación de los nodos en ciertos casos, se definen 3 tipos de mensajes que sirven para extender el protocolo. Estos mensajes forman parte del core de OLSR:[8]

- HELLO-messages: Se encargan de tareas como escucha del canal, detección de nodos vecinos y señalización de los MPRs.
- TC-messages: Información sobre el estado del enlace que pueda implicar un cambio en el enrutamiento.
- MID-messages: Declaran la presencia de múltiples interfaces en un nodo.

2.1.1.2 DSDV

El protocolo DSDV (Destination-Sequenced Distance Vector) utiliza el algoritmo de Bellman-Ford para calcular las rutas. Es un algoritmo de tipo table-driven, es decir que la extracción de la información relacionada con las rutas se hace mediante una tabla y no mediante lógica de programación. Por ello, mantiene la información de las tablas de enrutamiento de todos los nodos y no solo de los vecinos. Los cambios son propagados por toda la red de forma periódica o disparada por algún evento en dicha red. Debido a estos cambios, hay posibilidad de que se produzca algún ciclo en el grafo de la red, por tanto, para evitar eso, se utiliza un tag con el número de secuencia para cada cambio. Este número es independiente totalmente, pero tiene que ser incrementado cada vez q se produzca una actualización. Dicho incremento, para una actualización normal, es de un número par ya que cada vez que ocurre esto, el nodo incrementa su número de secuencia para el siguiente mensaje en 2. Un nodo no puede cambiar el número de secuencia de

otros nodos. Si un nodo quiere enviar una actualización a sus nodos vecinos de que la ruta ha expirado, entonces solo ahí incrementa su número de secuencia en 1. Los nodos que reciban este aviso verán que el número de secuencia es impar y por ello lo quitaran de la correspondiente entrada en su tabla de rutas. Cada tabla de rutas contiene la dirección del nodo destino, la distancia más corta para llegar a ese nodo y la dirección del nodo que está a un salto y que pertenece al camino más corto. Para minimizar el tráfico generado, hay dos tipos de paquetes: El "full-dump", que contiene toda la información del cambio producido, y el "incremental" que solo contiene los cambios realizados. Este último, por ende, incrementa la eficiencia del sistema. [10]

Está diseñado para redes con pocos nodos ya que tienen que mantener la ruta constantemente y eso requiere un alto gasto de batería, y también proporciona un reducido overhead.

2.1.2 Reactivos

2.1.2.1 AODV

AODV (Ad-Hoc On Demand Distance Vector) fue creado por Charles Perkins en 1999. Es un algoritmo que construye rutas on-demand, es decir, según sea requerida la activación del nodo. Ofrece rápida adaptación a enlaces dinámicos, reduce la utilización de la red y determina las rutas unicast a través de la red ad-hoc. Usa secuencia de números de nodos destino para asegurar que no se producen ciclos, incluso en el evento de un fallo o rotura del enlace. Esto evita problemas como el "count to infinity" asociado con algoritmos de distancia clásicos. No necesita a los nodos que no están activos para mantener las rutas hacia los nodos destino.

Para el funcionamiento de este algoritmo se utilizan 3 tipos de mensajes: Route Requests (RREQs), Route Replies (RREPs) y Route Errors (RERRs). Estos mensajes son recibidos vía UDP y procesado de solicitudes IP. El rango de difusión de estos mensajes está marcado por el TTL en la cabecera IP y la fragmentación normalmente no es requerida. En el momento en que dos end-points mantienen una conexión con una ruta válida para los dos, AODV deja de tener un rol. Cuando se necesita una nueva ruta hacia el nodo destino, ese nodo realiza un broadcast de RREQ hasta que llega a dicho nodo y después, para que la ruta sea comprobada como válida, el nodo destino envía de vuelta un unicast de tipo RREP. Cada nodo que ha recibido el RREQ antes del nodo destino, almacena la ruta de vuelta hacia el nodo origen consiguiendo así que la respuesta por parte del nodo destino sea de tipo unicast. [11]

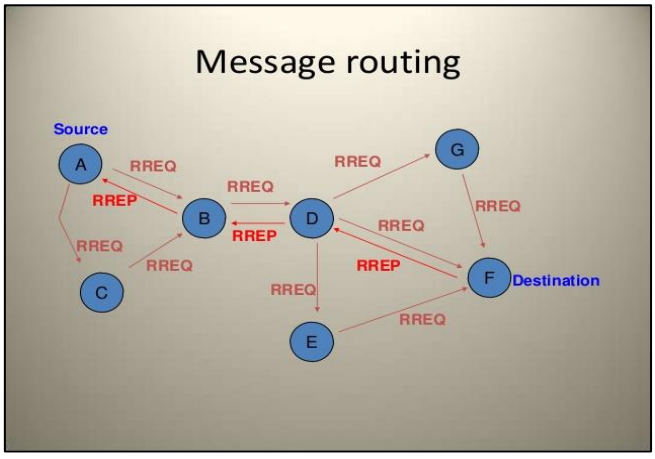


Figura 20: Ejemplo RREQ y RREP.

Los nodos monitorizan el estado del enlace de los next hops en rutas activas para que cuando un enlace se rompe, se pueda enviar automáticamente un mensaje de error RERR y poder calcular las rutas teniendo en cuenta ese enlace caído. Para poder implementar este sistema de control de errores, cada nodo mantiene una lista de predecesores con la dirección IP de cada vecino que pueda ser un next-hop.

Mantenimiento de las rutas

Para mantener las rutas se utiliza el envío periódico de mensajes HELLO. Si en una ruta activa, el mensaje HELLO no es recibido puede considerarse como un error del enlace y por consiguiente un envío de RERR. Cada vez que un nodo recibe este tipo de mensaje, tiene que determinar si se trata de una ruta activa al nodo vecino y si es necesario crearla. Si la ruta ya existe entonces deberá incrementar el campo Lifetime. A parte de los mensajes de tipo HELLO, los nodos pueden comprobar si un nodo next-hop es accesible usando reconocimientos MAC. [11]

Fallo en el enlace

Como hemos dicho, ante un fallo en un enlace se envía un mensaje de tipo RERR. Este sería el procedimiento: EL nodo C detecta que el enlace ha fallado e incrementa el número de secuencia. Este número de secuencia se incluye en el mensaje RERR que es enviado por C a los nodos vecinos. En la Figura 20 se puede ver dicho funcionamiento. [11]

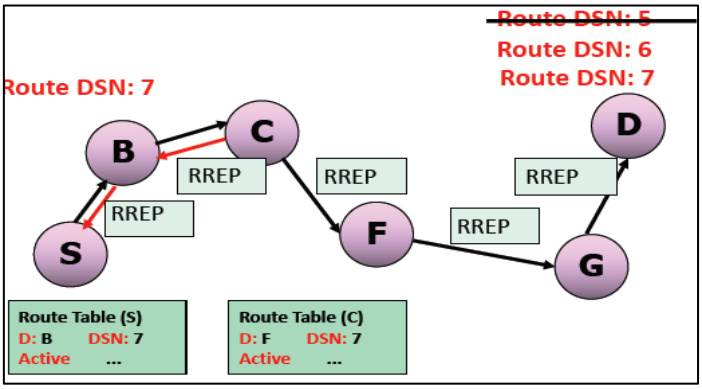


Figura 21: Fallo en el enlace.[19]

Ventajas y desventajas

Como puntos a favor en este tipo de enrutamiento tenemos: Evita que las rutas se incluyan en las cabeceras de los paquetes. El mantenimiento de las rutas solo es necesario para nodos activos en dicha ruta, los números de secuencia de los nodos evita rutas con enlaces rotos y formación de ciclos.

Los factores en contra al usar este protocolo son: Las rutas múltiples entre nodo origen y destino no están soportadas y necesidad imperiosa de enlaces simétricos. [19]

2.1.2.2 DSR

El protocolo Dynamic Source Routing fue creado por David Johnson en 1996. Es un protocolo enteramente “on demand” que no admite mensajes periódicos o avisos en ninguna de sus capas. Usando este protocolo, hace que la red se autogestione y se autoconfigure sin necesidad de ninguno tipo de infraestructura y por ello es tan eficiente en entornos ad-hoc. Los nodos que forman la red cooperan de manera que se puedan transmitir los mensajes de un nodo a otro a través de múltiples saltos. Es un protocolo con muy poco overhead y por lo tanto con mucha rapidez de reacción ante cambios en la red incluso si dicha red consta de movilidad. [11]

DSR está compuesto por dos mecanismos principales que trabajan en sintonía para permitir el descubrimiento y mantenimiento de rutas:[11]

- Route Discovery: Es el mecanismo mediante el cual un nodo origen S que quiere transmitir un paquete a un nodo destino D obtiene una ruta de enlace hasta él. Este método solo se utiliza cuando se quiere realizar un envío y todavía no se conoce la ruta. En este proceso, el algoritmo DSR no solo obtiene una ruta, sino que obtiene varias, aunque implemente antes la óptima. Esto es para que, si en caso de que un nodo falle, el algoritmo tenga en cache más rutas y pueda elegir una de ellas sin aumentar su overhead.
- Route Maintenance: Mecanismo en el cual un nodo es capaz de detectar, teniendo una ruta hacia el nodo destino, si la topología de la red ha cambiado tanto como para que no se pueda mantener esa ruta por culpa de un enlace roto u otro problema. En caso de enlace roto, hay dos opciones: Intentar otra ruta que se sepa que llega al nodo destino, o volver a ejecutar el route discovery.

En la Figura 22 se puede ver como la ruta del nodo S al nodo D se ve "interrumpida" por un fallo en el enlace C-F y como el nodo S tiene guardadas varias rutas en su cache, eligiendo otra de ellas que llega al nodo destino D, y así evita tener que volver a realizar otro route discovery.

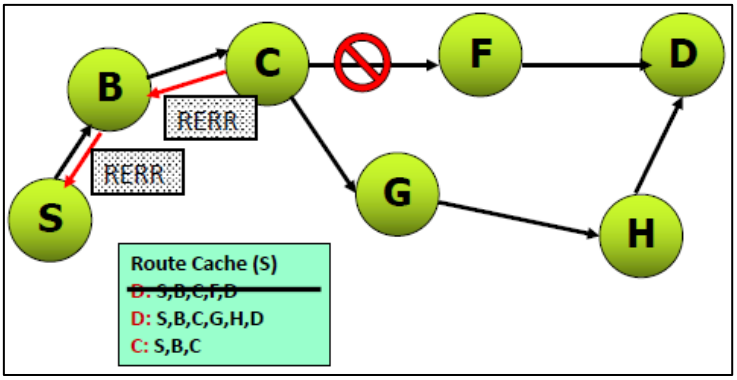


Figura 22: Mantenimiento de ruta. [19]

Dado que es un algoritmo que trabaja totalmente on-demand y tiene ausencia de procesos periódicos, su overhead tiende muchas veces a cero cuando los nodos están aproximadamente estacionarios unos respecto de los otros y las rutas hacia el nodo destino están ya descubiertas. Este valor del overhead solo aumenta cuando hay un cambio en la topología y se necesita rastrear el resto de las rutas disponibles. Todo estado mantenido en DSR es considerado "soft state" ya que, aunque se pierda alguno, no interfiere con el funcionamiento del protocolo y solo afecta a su eficiencia. Este modo de funcionamiento hace que sea muy robusto ante problemas como paquetes perdidos o retardos y enlaces rotos. Particularmente, un nodo que ha tenido un error y debe reiniciarse, puede incorporarse otra vez de forma fácil a la red sin casi o ninguna interrupción en el proceso. [12]

Packet Salvagin

Cuando un nodo reenviando un paquete detecta a través del proceso de mantenimiento de ruta que el siguiente salto para ese paquete está roto, si el nodo tiene otra ruta alternativa para que ese paquete llegue a su destino, el nodo debería "salvar" el paquete enviándolo por esa ruta, en lugar de descartarlo. Cuando se salva un paquete, hay un contador que gestiona el número de veces que un paquete ha sido salvado para evitar que ocurra el error de ser salvado infinitas veces. [19]

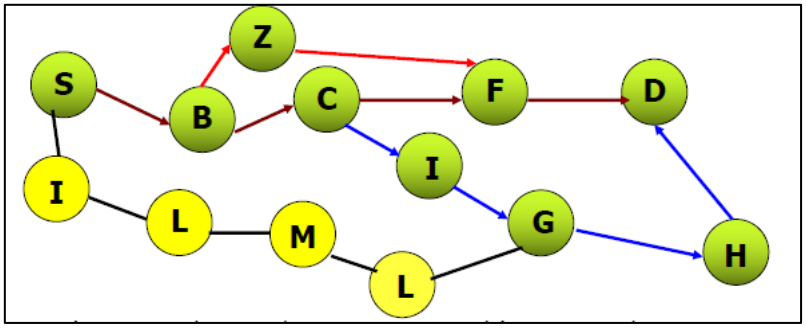


Figura 23: Ejemplo packet salvaging.

En el ejemplo de la Figura 23, la ruta de S a D es S-B-C-F-D. Sin embargo, S y C tienen una ruta alternativa en su cache para llegar a D. En un momento dado, F desaparece de la red, por problemas en su enlace o cualquier otro evento, y tanto S como B no reciben el mensaje de error RERR de respectivamente C o Z, y por lo tanto S sigue mandando paquetes hacia C. C puede salvar los paquetes remplazando la ruta original por la alternativa que tenía en cache evitando que se tire ningún paquete. Ahora imaginemos que el enlace que se rompe es el de G-H. Si G intenta salvar el paquete se produciría un ciclo porque G no sabe que el paquete es originado en S, solo ve al nodo C. Sin embargo, C puede detectar ese fallo y hacer que S responda con la ruta S-B-Z-F-D.

Ventajas y desventajas

Las ventajas de este protocolo son: Totalmente reactivo que soporta Soft state y links unidireccionales, enrutamiento de fuente sin necesidad de toma de decisiones por parte de nodos intermedios, y caching para reducir overhead en redescubrimiento de rutas.

Las desventajas de este protocolo son relacionadas con colisiones entre RREQs propagados por nodos vecinos, delays y jitters debido al enrutamiento on demand y caches invalidas que pueden afectar a la eficiencia.[19]

2.2 Basados en la posición

Los nodos determinan su posición geográfica mediante GPS; Usando mecanismos como tramas Beacon o mensajes CAM un nodo puede saber la posición de los nodos vecinos. Se asume que las coordenadas del nodo destino son sabidas, y más fácil si se trata de una RSU, y se incluyen en el paquete de datos. Los next-hops se van determinando según el cambio en la topología debido al movimiento y según eso eligiéndolo para tener un camino más corto. [13]

Hay dos tipos de algoritmos para este tipo de enrutamiento. Los que implementan servicios de ubicación (Location Services) y los que se dedican más a la estrategia de reenvío (Forwarding Strategy). [13]

- Location Services: DREAM, Quorum-based, GLS, Home zone.
- Forwarding Strategy: Greedy, GPSR, RDF, Hierarchical.

2.2.1 Location services

Para saber la posición exacta de un nodo, tenemos que hacer uso de los servicios de ubicación. Cuando un nodo no sabe la posición de otro nodo, contacta con el location Service. En las redes ad-hoc no es posible centralizar este método de la localización ya que sería complicado solicitar una dirección de un nodo a un servidor que está dentro de la propia red, del cual desconocemos su posición y, además, puesto que hablamos de una arquitectura con movilidad, sería complicado garantizar que, dentro del rango de este nodo, exista un servidor que nos de dicha localización. Por lo tanto, la única forma de hacerlo es descentralizando los servicios.

2.2.1.1 GLS

El Grid Location Service es un protocolo que divide el área que conforma la red ad-hoc en cuadrados con niveles de jerarquía. En esta jerarquía se forman 4 cuadrados del orden de (n-1), formando lo que se llama "quadtree". Cada nodo mantiene una tabla de los otros nodos que están dentro del cuadrado local. La tabla se construye con la ayuda de mensajes broadcast periódicos en los que se anuncia la posición de cada nodo del cuadrado local. Los datos de la tabla se conforman con los IDs de los nodos. [13]

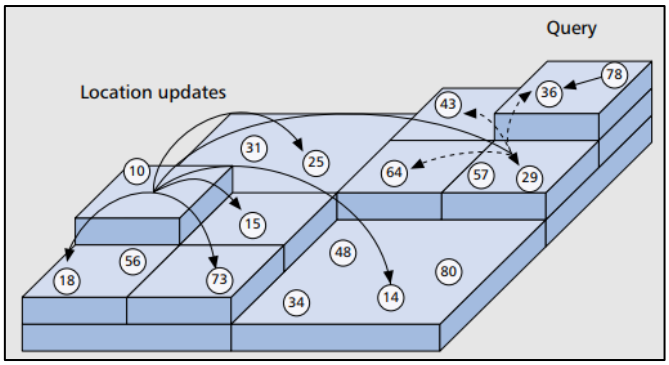


Figura 24: Jerarquía en forma de cuadrado de GLS. [13]

Fijándonos en la figura 24, cuando el nodo 10 quiere distribuir la información de su posición, la envía al nodo con el ID más cercano dentro de los 3 cuadrados de primer orden que le rodean, es decir, en este caso la recibirían los nodos 18,15 y 73. Después este proceso se repite con los 3 cuadrados de segundo orden más cercanos al nodo 10, que serían los nodos 14,25 y 29, y con el resto de ordenes hasta poder cubrir toda la red. Supongamos ahora que el nodo 78 quiere saber la posición del nodo 10. Como él no tiene ninguna información, debe de buscarla en un nodo cercano a él. En el ejemplo este es el nodo 29. El nodo 29 tiene en su mismo orden los nodos 64 y 43, y en los nodos cercanos el 36. Entonces, el nodo 78 puede preguntar al 36 que a su vez preguntara al más cercano que es el 29, que ya sabe la localización de 10. [13]

2.2.1.2 Quorum-Based Location Service

El concepto de los sistemas quorum es bien conocido en ámbitos de las bases de datos para copias y sistemas distribuidos. La información que se va a actualizar, es decir, una operación de escritura, primero se envía a un subconjunto de nodos (quorum) y cuando se recibe una solicitud (lectura), se envía a otro subconjunto diferente. Cuando un subconjunto designado tal que sus intersecciones son distintas de cero, es decir, que existe algún elemento, esto quiere decir que siempre va a haber una actualización disponible. [13]

Vamos a explicar este protocolo con ayuda de la siguiente imagen.

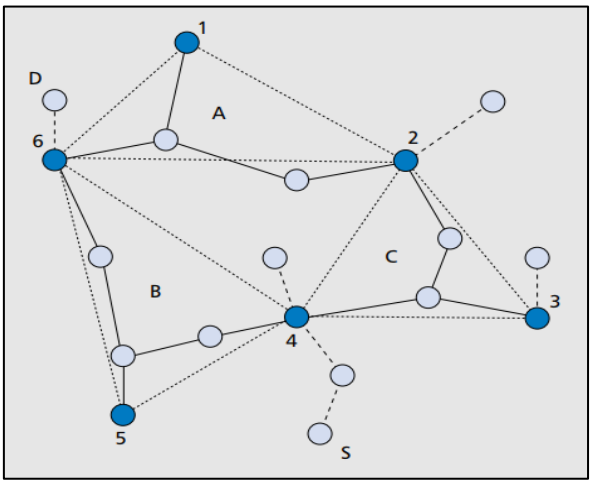


Figura 25: Ejemplo quorum-based.

Un grupo de nodos es el encargado de formar el subconjunto de nodos que se encargará de albergar la información de las bases de datos, en este caso los nodos 1 a 6 y que formaran una backbone dentro de la red. Un nodo móvil envía su posición al nodo de la backbone más cercano a él, el cual escoge un quorum de nodos de dicha backbone para albergar la información de la posición de ese nodo. Por lo tanto, en el ejemplo, el nodo D envía su posición al nodo 6 el cual puede que seleccione quorum con el nodo A que a su vez cuenta con los nodos 1 y 2, que junto con el 6 guardarían la información de la posición de D. Es importante tener tiempo en el que se ha hecho una actualización porque si no, al solicitar información a un nodo dentro de un quorum, puede que ese nodo tenga una posición antigua de otra actualización a la que haya pertenecido dicho quorum. Un aspecto importante de esta forma de posicionamiento basada en quorum es que cuanto más largo es el quorum, más alto es el coste por obtener actualizaciones de posicionamiento, pero también más alto es el número de nodos en una intersección entre dos quórums lo que hace que mejore la resiliencia ante la búsqueda de nodos en la backbone. En este esquema, hay tres posibles modos de configuración: All-for-all, all-for-some o some-for-some, dependiendo del tamaño de la backbone y del quorum elegido. [13]

2.3 Forwarding Strategy

2.3.1 GPSR

El algoritmo GPSR (Greedy Perimeter Stateless Routing) realiza decisiones “avariciosas” (Greedy) en el reenvío usando las posiciones de los routers vecinos y la dirección de destino del paquete que está enviando. En él se describen dos tipos de reenvíos: Greedy Forwarding, usado siempre que sea posible, y Perimeter Forwarding, que se usa cuando hay regiones en las que el anterior no puede funcionar.

Greedy forwarding

En GPSR los nodos fuente marcan los paquetes con las direcciones de destino. Como resultado, el nodo que reenvía el paquete en uno de los saltos puede escoger localmente el siguiente salto de forma óptima. Especialmente, si un nodo sabe la posición radio de sus vecinos, la opción óptima para el siguiente salto es el vecino geográficamente más cercano al nodo destino del paquete. Un ejemplo de siguiente salto con el algoritmo Greedy se puede ver en la siguiente imagen. [14]

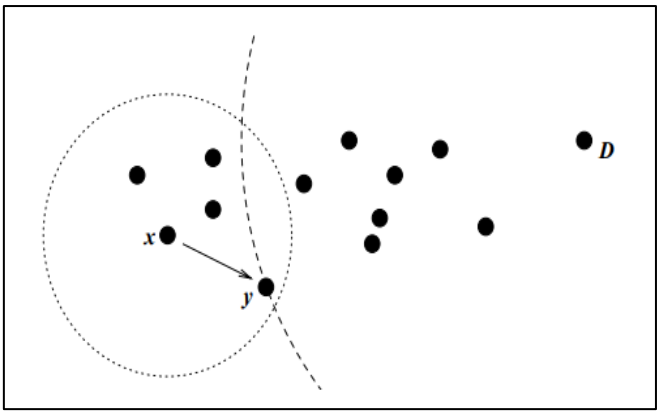


Figura 26: Greedy Forwarding. [14]

Aquí "x" recibe un paquete con destino a D. "x" reenvía el paquete a "y" ya que es la menor distancia que hay de todos los nodos que hay en su rango de señal. Este proceso se repite hasta que el paquete llega a D. Un periódico envío de beacons mantiene informado de las posiciones de los vecinos a todos los nodos transmitiendo en su trama la dirección MAC que contiene su IP, y la posición. Para evitar la sincronización entre los beacons de los vecinos, se añade un jitter del 50% entre envíos. Este mecanismo de beaconing, también existente en algoritmos de tipo proactivos, es mejorado de forma que, para minimizar el coste, se utiliza el sistema de piggybacking para enviar la posición del nodo emisor en cada paquete de datos que se envía. Este método reduce el tráfico de beacons y solo añade a cada paquete 12 bytes.

El Greedy Forwarding tiene el problema de que normalmente esta estrategia como reside en coger el nodo más cercano al destino, este suele ser un Edge node. El problema de los Edge nodes es que tienen más probabilidad de en un momento dado, salirse del rango de acción del nodo desde donde enviamos. Por lo tanto, tendría más posibilidad de fallar. En dicho caso, pasaríamos a la otra estrategia que utiliza GPSR que es el Perimeter Forwarding. [14]

Perimeter Forwarding

El Perimeter Forwarding se utiliza en el caso de que el Greedy Forwarding no pueda ser implementado por problemas de alcance al nodo destino. Para llevar a cabo esta estrategia se utiliza la técnica de la mano derecha para recorrer el grafo y se realiza hasta que el Greedy Forwarding pueda volver a utilizarse. El problema de este método es que, al ser no estar calculado ni optimizado, lo que lleva es a que se produzca redundancia y el camino elegido sea mucho más largo que si se llevara a cabo de otra forma. [14]

Capítulo 3: Descripción del software Ns3

Para la simulación de este proyecto teníamos varias opciones de software para representar los algoritmos de enrutamiento como por ejemplo Veins, Sumo y Omnet entre otros. Finalmente, debido a los requerimientos que tenía este proyecto, decidimos utilizar Ns3 por ser un programa de simulación de eventos discretos, y por tener incorporados muchos de los algoritmos que necesitábamos para dicha implementación. También decidimos que era buena idea porque puede trabajar en sintonía con otro programa como Netanim, que mediante la creación de un archivo con extensión XML, realiza una animación de lo simulado.

Ns3 es un simulador de redes de eventos discretos enfocado principalmente para su uso en la investigación y en la educación. Es un simulador de código abierto en el que se utilizan los lenguajes C++ y Python. Ha sido desarrollado para proveer de un simulador de redes extenso que nos proporciona modelos de redes de paquetes para ver su funcionamiento y poder usarlo como motor de simulación en nuestros experimentos. Algunas razones por las que se ha creado este software y que motivan su utilización, son que hay sistemas que, en la realidad, para poder estudiarlos en profundidad, puede llegar a ser bastante complicado el llevarlo a acabo. Sin embargo, con esta herramienta, que además es muy extensa como veremos en su variedad de modelos de simulación, podemos simular no solo escenarios basados en internet, sino que también se pueden simular otro tipo de escenarios y de más nivel de complejidad. Está compuesto por un conjunto de librerías que se pueden combinar entre ellas y con otras librerías de software externo y, aún sigue en permanente desarrollo ya que se beneficia de los reports que hacen los usuarios y se ayuda de los comentarios de la comunidad. Para su uso es necesario instalar Ubuntu, (en nuestro caso hemos utilizado una máquina virtual) y gran parte de las configuraciones y de ciertos parámetros han de hacerse mediante la consola de comandos. Al contrario de lo que podríamos pensar, el simulador Ns3 no es compatible con su antecesor, el Ns2 ya que este está escrito en C++, pero además está también en Phyton y, sin embargo, el Ns2 está en C++ y OCTL.

La organización del software se divide en seis niveles que ahora detallaremos:

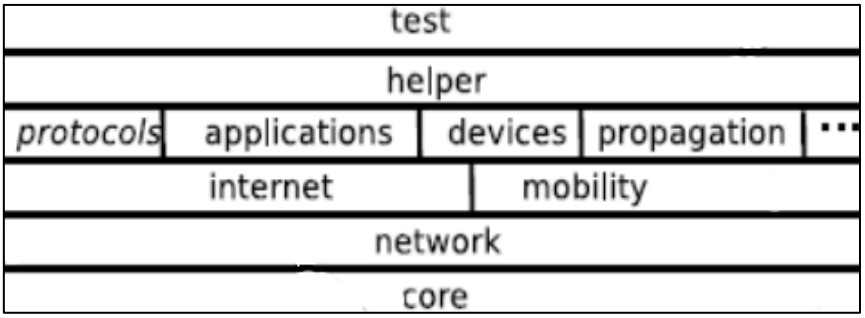


Figura 27: Organización Ns3.

En la imagen anterior se aprecia el funcionamiento de cada módulo y el núcleo de la simulación que se lleva a cabo en el directorio src/core y la base que se utiliza para construir el motor de simulación. Estos tres módulos están hechos para formar parte del núcleo genérico de simulación que puede ser utilizado para distintos tipos de redes. En el core también se implementan programación de eventos para simulaciones de aplicaciones dentro de la propia simulación. En el tercer nivel se encuentran los más importantes, nodo y movilidad, que serán los encargados de

configurar los datos relacionados con los nodos y los dispositivos inalámbricos para el control y la movilidad que tienen los nodos y que serán guardados dentro de DevicesContainers.

Para terminar, los últimos dos escalones (test y helper) son contenedores de alto nivel para ayudar dentro del script con ciertas funcionalidades más específicas.

3.1 Módulos Ns3

En los módulos Ns3 se implementan diferentes categorías y cada uno tiene ciertos atributos. Para cada módulo hay "helpers" para que la programación sea más sencilla y a nivel más bajo de API. Los módulos que se pueden encontrar son: [15]

- Antenna
- AODV
- Applications
- Bridge
- Buildings
- Config-store
- Core
- Csma
- Csma-layout
- DSDV
- DSR
- Emu
- Energy
- Fid-net-device
- Flow monitor
- Internet
- LTE
- Mesh
- Mobility
- Mpi
- Netanim
- Network
- Nix-vector-routing
- OLSR
- Point-to-point
- Propagation
- SixIowpan
- Spectrum
- Stats
- Tap-bridge
- Test
- Topology-read
- Uan
- Virtual-net-device
- Visualizer
- Wave
- Wifi
- Wimax

3.2 Namespace Ns3

Los namespaces hacen referencia a los contextos en los que se ve envuelto el código. Si no definimos ninguno, por defecto se considerará como un espacio global. Como veremos a la hora de describir el código, el espacio de nombres que utilizaremos será el propio del Ns3 que, evidentemente, es un espacio totalmente distinto al global. Por ello, cada vez que queramos hacer uso de dicho espacio, lo indicaremos de la siguiente forma: "ns3::". Así, también, podremos hacer referencia a otros módulos que están dentro de dicho espacio. Un ejemplo parecido es cuando queremos en C++ sacar algo por pantalla. Si queremos utilizar directamente el *cout*, sin tener que declararlo módulo, podemos indicarlo así: "std::cout". Esto significa que estamos en el espacio de std que es un objeto de la librería, iostream. [15]

3.3 Estructura de un script en Ns3

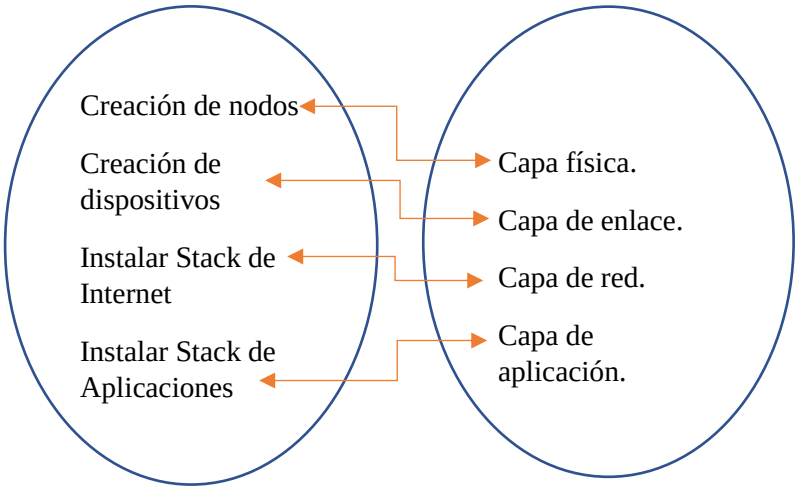
Para entender un poco el funcionamiento del software Ns3, vamos a proceder con la explicación de la estructura de un script que nos viene como ejemplo para implementar el algoritmo AODV, y al cual le fuimos haciendo alguna modificación en el proceso de aprendizaje, antes de poder juntar el resto de los algoritmos para la simulación importante en este proyecto. Más adelante, trataremos con más profundidad como se ha procedido hasta llegar a la idea final de simulación

que se ha realizado. De momento iremos solo nombrando ciertas partes por encima y lo utilizaremos como guía.

Las primeras líneas que nos encontramos en el archivo son simples indicaciones de formato, como el estilo de código que usamos que ayudará en caso de que utilicemos Emacs. Luego, más abajo, podemos encontrar normalmente un texto en el que se resume un poco el contenido del código y su función. Pasando a la parte del código, encontramos las definiciones de las librerías que se van a utilizar como son en este caso: El módulo del AODV, el core, el módulo de red y los módulos wifi. Este programa ejemplo trata de realizar un enrutamiento usando este algoritmo y, además, realizar un pin desde el nodo 0 al nodo 9. Para ello hay dos partes de simulación: La primera en la que se implementa el algoritmo en los nodos y la segunda en la que se hacer un evento de simulación para poder realizar el ping del nodo 0 al 9. Primeramente, se crea una clase para poder declarar parámetros como la cantidad de nodos que vamos a crear, el tiempo total de simulación o la distancia entre los nodos. Posteriormente, se crea una parte de configuración la cual nos servirá para poder modificar, a través de la consola de comandos, ciertos parámetros en el momento de ejecución. Seguido a esto, tenemos la parte por la que el sistema va a empezar a "leer" el programa y a ejecutarlo. Aquí es donde irán todas las funciones creadas en el orden que queramos ejecutarlas.

Lo primero que se crea a la hora de empezar el escenario, son los nodos; luego se les da una posición, que como veremos más adelante, puede ser fija o estratégicamente colocada (en grid, por ejemplo); a continuación, junto con la posición, le damos una movilidad que, en el caso de este escenario, es estática. Configuramos la capa física y la MAC de nuestros nodos, así como las características del canal. Una vez instalado todo esto en nuestros nodos, viene la parte de aplicarles el algoritmo de enrutamiento y de darles una dirección IPv4 para que se puedan comunicar. Por último, como el objetivo de este sencillo programa, aparte de ejecutar el algoritmo, es el de enviar un ping del primer nodo al último, se utiliza uno de los módulos que tiene Ns3 para poder ejecutarlo y se crea un programador de eventos para ejecutarlos durante solo un tiempo dentro de la propia ejecución.

Como podemos ver, un programa en Ns3 tiene bastantes partes diferenciadas, prácticamente podríamos decir que está estructurado según las capas OSI. [15]



3.4 NetAnim

Es un software que se descarga junto con los archivos de Ns3. Trabaja como herramienta de ayuda para simular de forma animada los escenarios creados con Ns3 creando un archivo XML que se forma durante la simulación del programa. Para que se cree el archivo es necesario añadir unas líneas dentro del código y ocasionalmente también es necesario modificar un archivo de tipo Wscript añadiéndolo como objeto del programa. Todo esto lo detallaremos más adelante cuando expliquemos el desarrollo del proyecto.

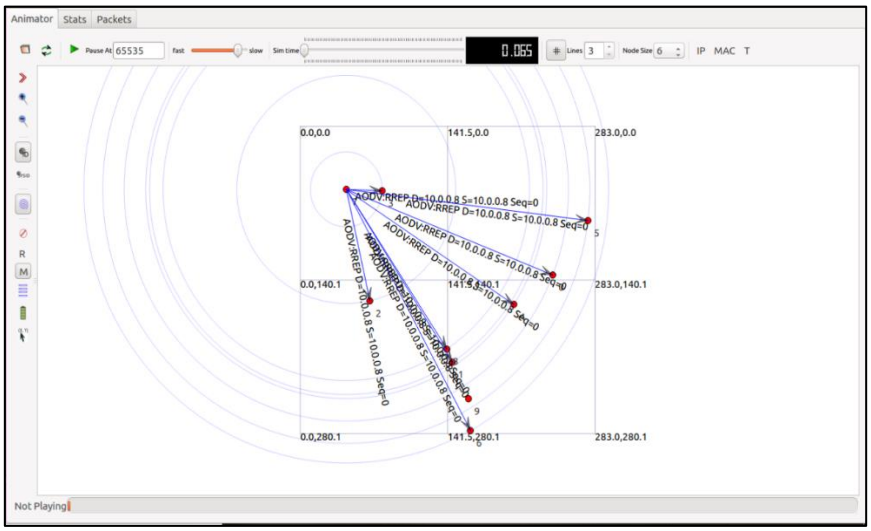


Figura 28: Simulación en NetAnim. [17]

En la Figura 28 podemos apreciar un ejemplo de simulación en el software NetAnim. En este caso, estamos simulando el protocolo AODV usado en el script de aodv.cc. El simulador nos deja ver las tramas que se envían entre los nodos, así como los rangos de transmisión de los que consta cada uno. En la barra superior, tenemos el tiempo de simulación y una barra a la izquierda para poder modificar la velocidad y también poder saltar de un punto a otro de la simulación. Si cambiamos de pestaña a la siguiente, podemos ver las estadísticas de los paquetes que han sido transmitidos y las direcciones IP de los nodos transmisor y receptor. En la última pestaña también podemos hacer un tracking de las tramas que se han ido enviando y seleccionar la visualización de un tipo u otro mediante un traffic flow stream. [17]

3.5 Instalación y configuración de Ns3

Para instalar el software tenemos varias opciones: Descargarlo en un archivo comprimido de la propia página de Ns3, descargarlo con git o por último con bake. En mi caso, lo descargué directamente en un archivo comprimido de la página mediante el siguiente comando:

```
$ wget https://www.nsnam.org/release/ns-allinone-3.29.tar.bz2
$ tar xjf ns-allinone-3.29.tar.bz2
```

A continuación, hay que realizar un comando “build” para poder crear todos los archivos del programa incluyendo los ejemplos, los test y los módulos. De hecho, cada vez que queramos añadir un módulo nuevo al programa, incluso si lo creamos nosotros mismos, hay que volver a ejecutar este comando para que se pueda luego simular con él y no dé error.

```
$ ./build.py --enable-examples --enable-tests
```

Una vez ejecutado este comando, nos saldrá, después de unos minutos, una pantalla con todos los módulos que han sido creados y otros que no lo han podido ser por cuestiones de dependencias o incompatibilidades. En nuestro caso no hubo dicho problema o por lo menos no afectaba a ninguno de los módulos de los que vamos a necesitar. Una vez hecho esto, tenemos que configurar el comando “waf” que será el que utilicemos cada vez que queramos realizar una simulación y además también nos permitirá activar la opción del debugger.

```
$ ./waf clean
$ ./waf configure --build-profile=optimized --enable-examples
```

Todo este proceso de instalación y configuración del entorno Ns3 ha sido relativamente sencillo ya que en la propia página del software viene bien detallado mediante una especie de tutorial. Además, este tutorial consta de una explicación más o menos detallada de unos ejemplos que vienen en uno de los directorios descargados. [15]

Capítulo 4: Desarrollo del proyecto

El objetivo desde el primer momento ha sido realizar un análisis y simulación de dos tipos de protocolos que se usan en VANET: Los reactivos y los proactivos. Para ello, hemos intentado simular dichos protocolos de la manera más realista posible, mediante el software Ns3, y también entresacar ciertos datos que nos sean de ayuda para poder sacar conclusiones acerca de su funcionamiento y cual o cuales son más convenientes.

Lo primero, fue instalar una máquina virtual en el ordenador con Ubuntu versión 16.04 ya que versiones anteriores se quedaban muy antiguas y versiones posteriores provocaban cierta ralentización en los procesos. A continuación, en la Figura podemos ver la configuración de la máquina que hemos creamos llamada “TFG”.

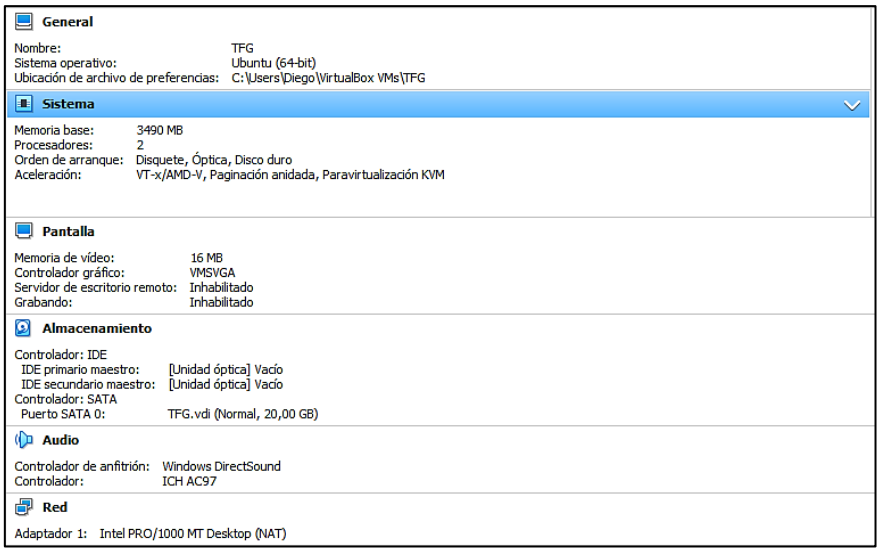


Figura 29: Configuración VM Ubuntu.

4.1 Simulación del script AODV y cambios realizados.

Posteriormente, una vez configurado el distro de Ubuntu en nuestra máquina, descargamos la versión 3.29 de Ns3 (la más actual). Como hemos comentado en el capítulo anterior, realicé todo el tutorial de iniciación del programa, que viene publicado en la página web de Ns3. Viendo algunas de las publicaciones que había en internet vimos que lo mejor sería realizar la animación con NetAnim que ya venía por defecto con el programa. Otras opciones como SUMO, de la que ya hablaremos, requerían otro software que hiciera de unión entre Ns3 y él. Además, para configurar un escenario en SUMO, se requerían demasiadas líneas de código en XML y no nos pareció ni practico ni abordable en el tiempo que tenemos para realizar este proyecto. Por lo tanto, lo primero que estudiamos fue como se enlazaba un archivo cualquiera con el programa de animación para poder generar el archivo XML. Para ello, dentro del directorio de Ns3, hay una carpeta que viene con ejemplos en los que se cargan archivos en NetAnim y para probar a ver cómo funcionaba, lo probé en uno de los ejemplos del tutorial. Solo era necesario añadir el módulo correspondiente de NetAnim y unas líneas antes de la orden de ejecución del programa, que servían para indicar el nombre del archivo que queríamos crear para esa simulación y otros parámetros que quisiéramos implementar.

```
AnimationInterface anim ("aodv-anim.xml"); //Archivo xml para generar netanim.


anim.EnablePacketMetadata (); // Optional
anim.EnableIpv4RouteTracking ("routingtable-aodv.xml", Seconds (0), Seconds (5), Seconds (0.25)); //para ruta paquetes ipv4.
anim.EnableWifiMacCounters (Seconds (0), Seconds (10)); //Optional
anim.EnableWifiPhyCounters (Seconds (0), Seconds (10)); //Optional
```

Figura 30: Líneas para poder ejecutar NetAnim.

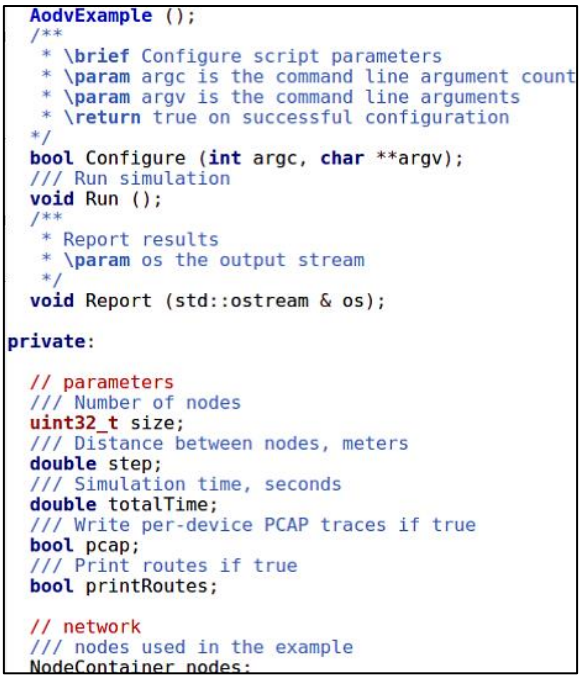
La Figura de arriba se refiere al programa de AODV en el que también las pusimos para poder estudiar el funcionamiento del script. La línea importante es la primera para poder generar el archivo; las siguientes son para que veamos datos de las tramas de cada paquete en el simulador, trackear las rutas que realizan los paquetes mediante la dirección IPv4, y contadores de MAC y capa física. Sin embargo, cuando me dispuse a simularlo, me apareció un error bastante tedioso de solucionar al principio ya que todavía me estaba haciendo al entorno de simulación y su funcionamiento, pero que, a base de buscar en foros, conseguí solucionarlo. Resulta que, aparte de tener que instanciar el módulo y crear el archivo XML, hay otro archivo dentro de la carpeta que opera con los scripts en forma de objetos. Esto es el archivo "wscript".

```
## -*- Mode: python; py-indent-offset: 4; indent-tabs-mode: nil; coding: utf-8; -*-

def build(bld):
    obj = bld.create_ns3_program('aodv',
                                 ['wifi', 'internet', 'aodv', 'internet-apps','netanim','wave'])
    obj.source = 'aodv.cc'
```

Figura 31: Archivo wscript.

Como se ve en la Figura, lo que contiene el archivo es una definición de objeto para el archivo AODV. En él, vemos los módulos que aparecen en el script original y el que tuvimos que añadir para poder simular (netanim), subrayado. Después de añadirlo ya pudimos simular sin errores. Como el proyecto se trata de simular los algoritmos de enrutamiento, este programa venía bastante bien como punto de partida para poder empezar a entender cómo funcionaba y poder desarrollar el resto y dictaminar el punto de vista desde el cual íbamos a simularlos.

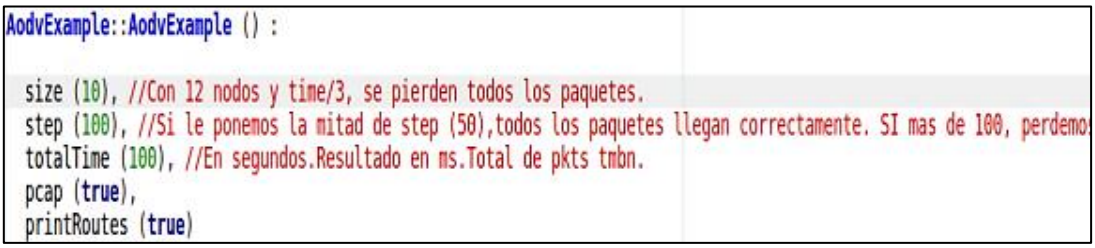
Como he dicho, el primer script que me sirvió como punto de partida fue el de AODV que ahora analizaremos. Lo primero que encontramos son los módulos utilizados para su desarrollo. Los más importantes y específicos para este tipo de simulación son: el módulo de AODV, los módulos de movilidad, los de wifi y los de la aplicación ping. El módulo de AODV contiene todos los archivos necesarios para realizar el enrutamiento como los scripts de las tramas RREQ y RREP, y las de error RERR. El último de los módulos añadidos fue el de netanim como hemos dicho anteriormente, para poder ir viendo los cambios que hacíamos de forma animada. Lo siguiente que encontramos es la creación de una clase llamada "AodvExample" con las declaraciones de los contenedores para los nodos, direcciones IP y dispositivos.



```
  AodvExample ();
  /**
   * \brief Configure script parameters
   * \param argc is the command line argument count
   * \param argv is the command line arguments
   * \return true on successful configuration
  */
  bool Configure (int argc, char **argv);
  /// Run simulation
  void Run ();
  /**
   * Report results
   * \param os the output stream
   */
  void Report (std::ostream & os);

private:

  // parameters
  /// Number of nodes
  uint32_t size;
  /// Distance between nodes, meters
  double step;
  /// Simulation time, seconds
  double totalTime;
  /// Write per-device PCAP traces if true
  bool pcap;
  /// Print routes if true
  bool printRoutes;

  // network
  /// nodes used in the example
  NodeContainer nodes;
```

Figura 32: Declaración de la clase AodvExample.

Esta declaración es muy importante ya que son los objetos que van a contener todos los nodos y sus parámetros de la simulación. Después se declaran las funciones de manera privada las funciones para nodos, dispositivos y aplicaciones. Más abajo, tenemos la declaración del número de nodos, distancia entre ellos, tiempo de simulación y si queremos que haya o no generación de ficheros de tráfico para analizar en Wireshark y además que nos imprima las rutas.



```
AodvExample::AodvExample () :

  size (10), //Con 12 nodos y time/3, se pierden todos los paquetes.
  step (100), //Si le ponemos la mitad de step (50),todos los paquetes llegan correctamente. SI mas de 100, perdemo
  totalTime (100), //En segundos.Resultado en ms.Total de pkts tmbn.
  pcap (true),
  printRoutes (true)
```

Figura 33: Parámetros de simulación.

Los comentarios que he escrito en la declaración de “size”, son por un evento de simulación para hacer el ping que comentaremos más adelante, y en el programa, con esa configuración, no se podían poner más de 11 nodos porque la perdida que había en los paquetes ICMP del ping era demasiado grande como para que el sistema fuera admisible. En la siguiente declaración para la distancia entre nodos, como con 100 metros entre ellos había una pequeña pérdida, probamos a ver en qué momento la entrega era 100% exitosa, y resultó que con la mitad de distancia la obtuvimos. Sin embargo, con más de 100 metros se perdían todos.

En las siguientes líneas encontramos la configuración de la clase, en la que está la “seed”. La seed (semilla) sirve para que la simulación tome un carácter u otro dependiendo del número que le pongamos, es decir, en cierto modo es aleatorio ya que, si la cambiamos, el resultado no

depende de los números que pongamos en su valor, pero si la dejamos fija siempre obtendremos el mismo output. También encontramos los parámetros del "cmd" para poder configurar desde la consola de comandos y poder cambiar ciertos valores como haremos en nuestra simulación final.

```
AodvExample::Configure (int argc, char **argv)
{
  // Enable AODV logs by default. Comment this if too noisy
  // LogComponentEnable("AodvRoutingProtocol", LOG_LEVEL_ALL);

  SeedManager::SetSeed (12345); // cambiar para mirar como fluctua.

  std::string traceFile;
  std::string logFile;

  CommandLine cmd;

  cmd.AddValue ("pcap", "Write PCAP traces.", pcap);
  cmd.AddValue ("printRoutes", "Print routing table dumps.", printRoutes);
  cmd.AddValue ("size", "Number of nodes.", size);
  cmd.AddValue ("time", "Simulation time, s.", totalTime);
  cmd.AddValue ("step", "Grid step, m", step);

  cmd.AddValue("traceFile","Aodv traceFile",traceFile);
  cmd.AddValue("logFile","Aodv logFile",logFile);

  cmd.Parse (argc, argv);
  return true;

}
```

Figura 34: Configuración AodvExample.

La siguiente función que encontramos es la de ejecución. En ella situamos las funciones en el orden en el que queremos que se ejecuten y le añadimos algún parámetro por pantalla para formatear la salida. Además, también añadimos los parámetros que ya comentamos del NetAnim para que podamos visualizar el resultado. La siguiente que encontramos ahora corresponde con la creación de los nodos que simplemente se hace con un ciclo de 0 a la cantidad de nodos menos uno. Después de la creación de los nodos, dado que estamos hablando de una red ad-hoc con movilidad, hay que definir dicha movilidad y después de qué manera queremos posicionar a los nodos.

En el programa, al principio, la movilidad era de tipo constante cero, es decir, sin movimiento. Podría decirse que era un programa más de tipo MANET. En cuanto a la posición, ésta era de tipo "Grid" (cuadrícula). Esto es que los nodos se situaban en una cuadrícula según los parámetros de los ejes "x" e "y" que estuvieran determinados.

```
// Create static grid
MobilityHelper mobility;
mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
                               "MinX", DoubleValue (0.0),
                               "MinY", DoubleValue (0.0),
                               "DeltaX", DoubleValue (step),
                               "DeltaY", DoubleValue (0),
                               "GridWidth", UintegerValue (size),
                               "LayoutType", StringValue ("RowFirst"));
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (nodes);
```

Figura 35: Posicionamiento grid.

Para movilidad, según los modelos que nos deja utilizar Ns3, tenemos primero 3 tipos de estructuras que son en rectángulo, en caja y waypoint y 4 tipos de clases:

- GetPosition ().
- Atributos de velocidad y posición.
- GetDistanceFrom ().
- CouseChangeNotification.

Dentro de estos 4 tipos tenemos 10 subtipos.

- ConstantPosition.
- ConstantVelocity.
- ConstantAcceleration.
- GaussMarkov.
- Hierarchical.
- RandomDirection2D.
- RandomWalk2D.
- RandomWaypoint.
- SteadyStateRandomWaypoint.
- Waypoint.

Primero hicimos la simulación de los parámetros que aparecen en la Figura 35 y simplemente eran los nodos en línea separados 100 metros cada uno. Después cambiamos los parámetros y designamos una movilidad de tipo Waypoint con la misma estructura de Grid. Lo que se nos mostraba en la simulación era una movilidad a partir de coordenadas de espacio-tiempo que se le daban a los nodos mediante un vector. Por último, se optó por una movilidad de tipo RandomWaypoint en la que cada nodo se mueve con unas coordenadas totalmente aleatorias y en cualquier dirección. En este modelo se puede aplicar una velocidad y una pausa cada cierto tiempo. En cuanto a la posición de los nodos, en este caso aplicamos un RandomBoxPositionAllocator. Hay 6 tipos de modelos para la posición de los nodos.

- ListPositionAllocator.
- GridPositionAllocator.
- RandomRectanglePositionAllocator.
- RandomBoxPositionAllocator.
- RandomDiscPositionAllocator.
- UniformPositionAllocator.

```
MobilityHelper mobilityAdhoc;

ObjectFactory pos;
pos.SetTypeId ("ns3::RandomBoxPositionAllocator");
pos.Set ("X", StringValue ("ns3::UniformRandomVariable[Min=0.0|Max=300.0]")); //change bounds to reduce loss packets
pos.Set ("Y", StringValue ("ns3::UniformRandomVariable[Min=0.0|Max=300.0]"));

// pos.Set ("Z", StringValue ("ns3::UniformRandomVariable[Min=1.0|Max=100.0]"));

Ptr<PositionAllocator> taPositionAlloc = pos.Create ()->GetObject<PositionAllocator> ();

std::stringstream ssSpeed;
ssSpeed<<"ns3::UniformRandomVariable[Min=0.0|Max=10.0]";
std::stringstream ssPause;
ssPause <<"ns3::ConstantRandomVariable[Constant=1.0]";


mobilityAdhoc.SetMobilityModel("ns3::RandomWaypointMobilityModel",
                               "Speed",StringValue(ssSpeed.str()),
                               "Pause",StringValue(ssPause.str()),
                               "PositionAllocator",PointerValue (taPositionAlloc));
```

Figura 36: Movilidad aplicada al ejemplo.

El comentario de la posición del eje de las “x” es debido a que, si ampliamos la anchura de los límites, se pierden paquetes. En la siguiente imagen se puede ver el resultado con este modelo de movilidad y de posición mostrado en la Figura 36.

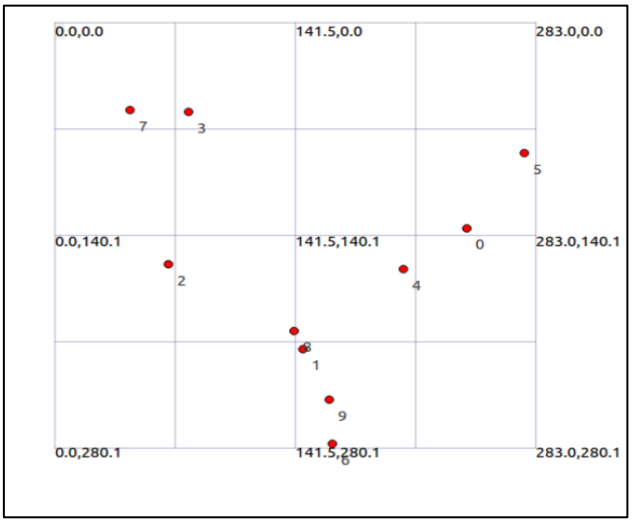


Figura 37: Simulación de la movilidad en NetAnim.

Como vemos, los límites de los ejes “x” e “y” no son exactamente 300 como habíamos puesto en los datos de movilidad, pero es normal porque se trata de una variable aleatoria con un máximo de 300. Si cambiáramos la semilla del programa, seguramente también cambiaría el máximo del límite de los ejes, aunque siempre sin llegar a los 300 metros.

Si seguimos con el código llegamos a la parte de los dispositivos. Aquí la configuración va a ser de dos tipos: Capa física y capa MAC. Por defecto en el script, la parte física utilizaba el helper de “YansWifi” y un retardo de propagación constante para las pérdidas. Además, para la caracterización del canal, se designaba una frecuencia típica de 802.11a, a 5Ghz (la que venía por

defecto) y una velocidad de 6 Mbps con modulación OFDM. La MAC en este caso era la que venía por defecto. Sin embargo, para nuestro ejemplo en VANET, lo que necesitamos es 802.11p que opere a una frecuencia de 5.9 GHz. Por lo tanto, cambiamos ese parámetro en la capa física, así como cambiamos los canales a una anchura de anchura 10 MHz. En cuanto a la MAC, probamos tanto una con QoS (Quality of Service), como una sin ella y el resultado fue mejor sin ella en cuanto a tiempo. Aun así, muchos de los parámetros que he nombrado se acabarían haciendo diferentes a la hora de realizar el proyecto en sí.

```
   //MAC
NqosWaveMacHelper waveMac =  NqosWaveMacHelper::Default(); //simple MAC w/o QoS.
WaveHelper waveHelper =  WaveHelper::Default();

   //HELPER
Wifi80211pHelper wifi80211p = Wifi80211pHelper::Default();
wifi80211p.SetStandard (WIFI_PHY_STANDARD_80211_10MHZ);
devices = wifi80211p.Install (wifiPhy,waveMac, nodes);
```

Figura 38: Configuración capa MAC y física.

En la función “InternetStack” se instala el algoritmo de enrutamiento AODV y se le puede configurar mediante ciertas opciones de parámetros que se nos proporcionan. En este caso no se efectuó ningún cambio. Después de instalarlo se instala en los nodos previamente creados y se les otorgan direcciones IPv4 y MAC, y se instalan en los dispositivos.

```
AodvExample::InstallInternetStack ()
{
  AodvHelper aodv;
  // you can configure AODV attributes here us
  InternetStackHelper stack;
  stack.SetRoutingHelper (aodv);
  stack.Install (nodes);
  Ipv4AddressHelper address;
  address.SetBase ("10.0.0.0", "255.0.0.0");
  interfaces = address.Assign (devices);
```

Figura 39: Función "InternetStack".

Como última función tenemos la de aplicación en la cual vamos a instalar un PING del nodo ‘0’ al nodo ‘9’. Primero la instalamos en todos los nodos y luego “obligamos” a que empiece en el nodo 0. A parte de configurar la aplicación PING, debemos de darle un tiempo de duración dentro de la propia simulación. En el programa original, estaba configurado un evento de simulación que ocurriera solo 1/3 del tiempo de simulación total, con lo cual solo se recibía 1/3 de los paquetes. Quitando esa línea, se envían el 100% de los paquetes y solo hay una pérdida del 11% como podemos ver en la siguiente Figura.

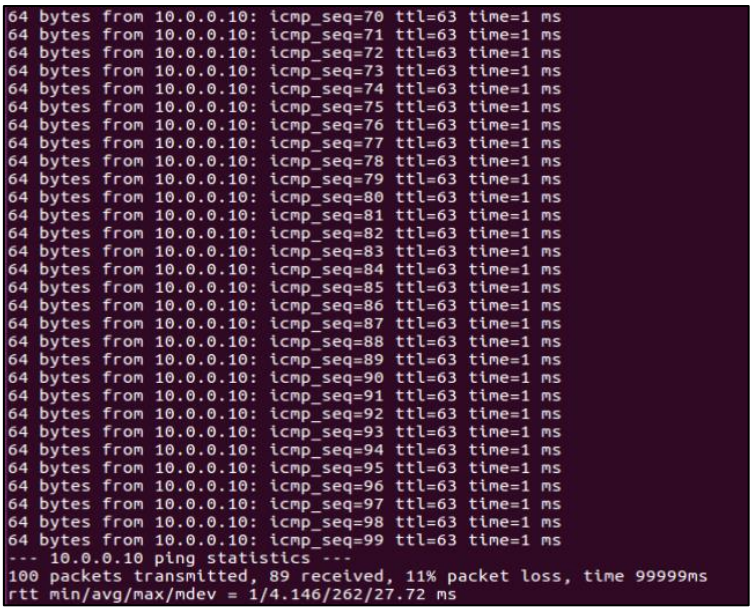


Figura 40: Ejecución del ping.

4.2 Configuración de los dos entornos para simulación de los algoritmos

Después de todo el análisis realizado con el script de AODV vamos a realizar el análisis que tiene como objetivo este proyecto. Para ello vamos a utilizar un script de comparación de algoritmos en un entorno VANET. En un principio, íbamos a analizar dos algoritmos reactivos (AODV y DSR) y dos proactivos (DSDV y OLSR). Sin embargo, debido a un problema en uno de los módulos de Ns3 con respecto al algoritmo DSR, es decir, un problema ajeno a nuestras posibilidades y del cual no era abordable una solución a corto plazo, decidimos realizar solo el análisis en los otros 3 restantes. El "bug" en el DSR hace que el PDR (Packet Delivery Ratio) no obtenga los valores que debiera siendo todos nulos. Este problema lleva presente en Ns3 sin que haya una solución que funcione desde al menos 2014.

Se trata de una autovía con nodos ad-hoc (VANET) configurable con diferentes parámetros y que consta de una aplicación de tráfico de mensajes BSM (Basic Safety Message). Partiendo de este script, podemos identificar dos partes o escenarios de él: El primero, que es el que nos interesa, parte de la idea de un escenario de autovía en el que se simula durante 10 segundos el comportamiento de 40 nodos a una velocidad de 20 m/s, con un modelo de movilidad de tipo RandomWaypoint, y un grid de 300x1500 metros. Usa la tecnología 802.11 con un ancho de banda e 10 MHz. Todos los nodos transmiten paquetes BSM de 200 bytes, 10 veces por segundo a una velocidad de 6 Mbps. Además, todos los nodos implementan una transmisión de paquetes de enrutamiento con un tamaño de 60 bytes a una velocidad de 2,048 Kbps. Estas transmisiones de paquetes son recibidas por un número de nodos que son designados de forma aleatoria como sumideros o sinks. Estos sumideros por defecto son 10, dado que tenemos 40 nodos, y no deben superar más de la mitad de los nodos existentes. En dicho caso, ocurrirá un error en la simulación. La potencia de transmisión es de 20 dBm. El segundo escenario, el cual no vamos a usar mucho ya que en nuestro proyecto no nos interesa, trata de un escenario que simula durante 300 segundos, 90 nodos en una calle de la ciudad de Zúrich, Suiza. El escenario está basado en un tráfico de datos real y necesita mucho tiempo para su simulación completa. Para su simulación, obtiene los datos de un archivo de movilidad que recoge dicho escenario.

Nuestra simulación, en un principio, iba a ser más realista en cuanto al movimiento del coche y a la dirección que le quisiéramos dar. Sin embargo, toda la información que encontraba iba dirigida a una movilidad aleatoria por parte de los nodos. Con lo cual, decidí que lo más apropiado era realizar una simulación centrándome en la densidad de los nodos. Esto quiere decir, que según las dimensiones de la carretera y de la cantidad de nodos, podríamos simular una situación real de circulación cambiando los parámetros según conviniera en cada una de ellas. Entonces, por un lado, vamos a realizar una simulación de los 3 protocolos para un escenario de autovía y por otro lado lo realizaremos para otro en una situación urbana. Además, debido a que designamos unas distancias máximas tanto en el eje "x" como en el "y", esto es lo que hemos utilizado para suponer el siguiente esquema. Como no hemos instalado, por así decirlo, un nodo fijo que actúe como RSU del resto de las OBUs, hemos hecho que la distancia máxima del eje "x" simule el alcance de cobertura que tendría una RSU y el eje "y" lo hemos utilizado para simular el escenario con 3 carriles de 3,5 metros cada uno, como así marca la legislación. Por lo tanto, para el entorno rural tendríamos una distancia máxima en el eje "x" de 500 metros y un ancho (eje "y") de 10,5 metros, y en el entorno urbano sería de aproximadamente 165 metros para el eje "x" y la misma anchura que en rural para el eje "y". En cuanto al código, éste sigue la misma estructura que el anterior mencionado, pero antes de entrar en detalle con cada simulación, voy a ir destacando ciertas partes del script que caracterizan algunos de los comportamientos que se verán en las simulaciones. Lo primero, como pasaba en el otro script de AODV, hay una declaración de módulos, una declaración de las funciones y variables globales que se van a utilizar. En este caso, hay una gran cantidad de líneas dedicadas a la declaración de dichas variables y funciones ya que hay muchos parámetros que se pueden extraer. Lo primero a destacar es la declaración de las distancias. Estas distancias se utilizan para que un nodo sepa con qué PDR llega un mensaje BSM y ver la variación que tiene ese porcentaje dependiendo de los metros que le separen de su destino. Así, en el entorno rural las distancias van desde los 50 metros hasta los 500 que alcanza la teórica RSU, y en urbanas, dado que las comunicaciones son mucho más próximas, van desde 1 metro a los 165 del rango de la RSU en ese caso. Esto es muy importante ya que los mensajes BSM son los que, como ya hemos comentado en la parte teórica de este documento, nos avisan de los peligros en la carretera y deben tener una alta fiabilidad y, por ende, un alto PDR. En urbano, sobre todo, cuanto más próximo esté el coche al que le enviamos la información más alto será su PDR, llegando casi a tener un resultado de 1. Según nos vayamos alejando con las distancias especificadas, este valor irá disminuyendo debido a factores como el desvanecimiento y pérdidas que ya comentaremos más adelante.

```
//Distancias para autovia.

double txDist1 = 50.0;
double txDist2 = 100.0;
double txDist3 = 150.0;
double txDist4 = 200.0;
double txDist5 = 250.0;
double txDist6 = 300.0;
double txDist7 = 350.0;
double txDist8 = 400.0;
double txDist9 = 450.0;
double txDist10 =500.0;

//Distancias urbanas.
/* double txDist1 = 1.0;
double txDist2 = 5.0;
double txDist3 = 25.0;
double txDist4 = 45.0;
double txDist5 = 65.0;
double txDist6 = 85.0;
double txDist7 = 105.0;
double txDist8 = 125.0;
double txDist9 = 145.0;
double txDist10 = 165.0;*/
```

Figura 41: Distancias para los dos entornos.

A continuación de haber comentado las distancias para poder recoger la efectividad de los mensajes BSM, vamos a ver qué opciones teníamos para modificar en nuestra simulación. Esto se hace mediante la consola de comandos. Hay ciertos parámetros que utilizando la forma "cmd.Addvalue" se pueden cambiar en la simulación, en la ventana de comandos, con solo añadir dos guiones y el nombre del objeto que queremos modificar. Estos parámetros son, por ejemplo, el tiempo total de simulación, el número de nodos y su velocidad, el número de sumideros, el tipo de protocolo queremos usar o incluso que tipo de norma WiFi elegimos (en nuestro caso siempre será la 802.11p).

```
cmd.AddValue ("totaltime", "Simulation end time", m_TotalSimTime);
cmd.AddValue ("nodes", "Number of nodes (i.e. vehicles)", m_nNodes);
cmd.AddValue ("sinks", "Number of routing sinks", m_nSinks);
cmd.AddValue ("txp", "Transmit power (dB), e.g. txp=7.5", m_txp);
cmd.AddValue ("traceMobility", "Enable mobility tracing", m_traceMobility);
cmd.AddValue ("protocol", "1=OLSR;2=AODV;3=DSDV;4=DSR", m_protocol);
cmd.AddValue ("lossModel", "1=Friis;2=ItuR1411Los;3=TwoRayGround;4=LogDistance", m_lossModel);
cmd.AddValue ("fading", "0=None;1=Nakagami;(buildings=1 overrides)", m_fading);
cmd.AddValue ("phyMode", "Wifi Phy mode", m_phyMode);
cmd.AddValue ("80211Mode", "1=802.11p; 2=802.11b; 3=WAVE-PHY", m_80211mode);
```

Figura 42: Ejemplo de parámetros que se pueden cambiar en la consola.

También, una de las opciones que tenemos, es la de guardar el fichero de simulación, que en realidad no nos aporta mucho puesto que simplemente muestra los datos de configuración que hemos empleado en dicha simulación. Sin embargo, sí que hay dos ficheros que se nos descargan que merece bastante la pena de analizar y con los que hemos conseguido extraer los datos para poder sacar conclusiones que discutiremos al final de este proyecto.

Las siguientes líneas del código que encontramos de interés, son las de movilidad. Esta parte se abre con una sentencia "If" para poder elegir entre las dos opciones de movilidad que se nos presentan: Uno para el de la calle en Zúrich y otro para nuestros dos escenarios. Lo primero que

configuramos es el “grid” en el que vamos a colocar los nodos. Como ya hemos dicho, este va a contener los datos certeros de las medidas de los carriles, y una longitud que dependerá del escenario que estemos simulando, rural o urbano. Después también tenemos el rango de velocidad con el que esperamos que los nodos se desplacen. Esto también dependerá del entorno en el que estemos. En rural hemos dado varios valores dependiendo un poco de la cantidad de nodos que pudiera haber y de la situación en general de la carretera. Esto lo veremos todo en las simulaciones. Para urbano la velocidad es mucho menor, y la hemos configurado alrededor de 40 km/h. En este caso sí que podría haber más nodos puesto que existe una posibilidad de atasco a ciertas horas punta. Respecto al entorno urbano, hemos tenido en cuenta tanto un tráfico fluido a la entrada de una ciudad, como un tráfico en horas de entrada o salida del trabajo en los que la movilidad es intermitente y a una velocidad baja, pero la comunicación entre los nodos sigue teniendo que estar presente, aunque con más interferencias de nodos (vehículos) próximos. Centrándonos más en el modelo de movilidad, en este caso hemos usado el RandomWayPoint, puesto que, a mi modo de ver, y al modo de ver de más autores que han realizado proyectos de este tipo, es el más aconsejable. Al pensar esto, se produce una problemática:

Puesto que, en un tipo de movilidad como este, el vector de desplazamiento es totalmente aleatorio, alguno de los nodos puede que vayan hacia atrás y eso no es posible. Entonces, ¿cómo podemos analizar esto sin que incurramos en un absurdo? Pues la mejor manera, después de analizarlo bastante, es esta: Puesto que la longitud que hemos establecido para nuestra carretera representa el alcance de cobertura que tiene la RSU, en el supuesto de que algún nodo retrocediera en nuestra simulación significaría que un nodo nuevo ha entrado en el rango de nuestra RSU y otro ha salido. De esta forma, lo podemos razonar y, además, este suceso no influye en nuestro análisis puesto que la densidad de nodos no cambia en ningún momento. Esta forma de analizarlo y de verlo, fue el punto de inflexión a partir del cual se fue avanzando más con el proyecto, ya que una de las cosas que no cuadraban, era el modelo de movilidad. A parte de la movilidad y la velocidad, también se le podía poner un tiempo de pausa en el movimiento, aunque en nuestra simulación este valor es cero.

```
else if (m_mobility == 2)
  {
    MobilityHelper mobilityAdhoc;

    ObjectFactory pos;
    pos.SetTypeId ("ns3::RandomBoxPositionAllocator");
    pos.Set ("X", StringValue ("ns3::UniformRandomVariable[Min=0.0|Max=2000.0]")); //cambiamos
    pos.Set ("Y", StringValue ("ns3::UniformRandomVariable[Min=0.0|Max=10.5]"));//tamaño urbano
    // we need antenna height uniform [1.0 .. 2.0] for loss model
    pos.Set ("Z", StringValue ("ns3::UniformRandomVariable[Min=1.0|Max=2.0]"));

    Ptr<PositionAllocator> taPositionAlloc = pos.Create ()->GetObject<PositionAllocator> ();
    m_streamIndex += taPositionAlloc->AssignStreams (m_streamIndex);

    std::stringstream ssSpeed;
    ssSpeed << "ns3::UniformRandomVariable[Min=17.0|Max=" << m_nodeSpeed << "]"; //para autovia
    std::stringstream ssPause;
    ssPause << "ns3::ConstantRandomVariable[Constant=" << m_nodePause << "]";
    mobilityAdhoc.SetMobilityModel ("ns3::RandomWaypointMobilityModel",
                                    "Speed", StringValue (ssSpeed.str ()),
                                    "Pause", StringValue (ssPause.str ()),
                                    "PositionAllocator", PointerValue (taPositionAlloc));
    mobilityAdhoc.SetPositionAllocator (taPositionAlloc);
```

Figura 43: Movilidad.

Otra de las razones por la cual usamos este modelo de movilidad, es que, evidentemente, el movimiento de un vehículo tiene un vector más o menos definido, sin embargo, tiene también cierta aleatoriedad en cuanto a cómo va a proceder. Esto quiere decir que, que vaya a más o menos velocidad en un momento u otro no es predecible (por ello las velocidades también son variables aleatorias dentro de un rango) o hacia dónde se desplaza en ese rango que tiene la RSU. Tampoco

es predecible qué cantidad de vehículos entran o salen de ese alcance puesto que depende de la velocidad que permita el tráfico y la cantidad de vehículos. Por ello, una variable que no fuera pseudoaleatoria digamos, no era una opción recomendable.

Lo siguiente que encontramos que influye en gran medida que una simulación de un resultado u otro es el modelo de propagación. En este caso hemos usado los dos que nos parecían los más adecuados para cada entorno. Para el escenario urbano hemos utilizado el modelo LogDistance y para el entorno rural el modelo de dos rayos. Ahora explicaremos los dos:

- LogDistance: Es un modelo de propagación radio que predice la pérdida de señal en un entorno denso. Es usado para interiores de edificios o áreas con alta de densidad de objetos en una cierta distancia. Este último es nuestro caso urbano.

$$L = L0 + 10 * n\, log_{10}(\frac{d}{d0})$$

  - n: exponente de pérdida.
  - d0: distancia de referencia (m).
  - L0: Pérdidas por atenuación en distancia de referencia (dB).
  - d: distancia (m).
  - L: Pérdidas atenuación (dB).

- TwoRayGround: Este es el modelo de dos rayos usado para espacio libre de obstáculos. Este modelo no es apto para distancias muy cortas debido a la oscilación causada por la combinación constructiva y destructiva de los dos rayos. Este modelo también se le llama, modelo de tierra plana. Lo hemos usado para el escenario rural puesto que era el más apropiado por cuestiones de ausencia de desvanecimientos de señal por objetos como edificios. En este caso, necesitamos incluir la frecuencia de funcionamiento de la antena (5.9 GHz) para que, según la altura de la antena, tenga en cuenta también esas pérdidas. Eso lo hacemos con el atributo: "HeightAboveZ", que se refiere a la altura en el eje 'z'. [15]

$$Pr = \frac{Pt * Gt * Gr + (Ht^2 * Hr^2)}{d^4 * L}$$

  - Pt: Potencia transmitida.
  - Pr: Potencia recibida.
  - Ht: Altura antena transmisora.
  - Hr: Altura antena receptora.
  - Gt: Ganancia del transmisor.
  - Gr: Ganancia receptor.
  - d: distancia (m).
  - L: Pérdidas atenuación (dB).

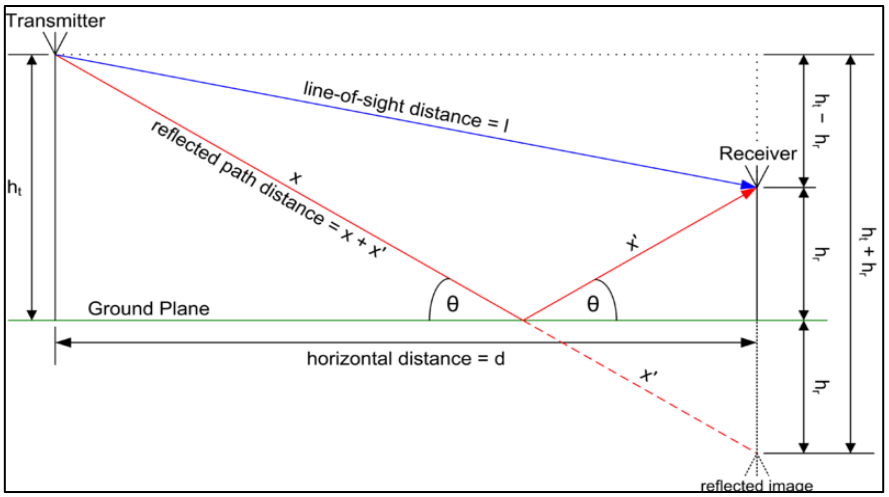


Figura 44: Modelo de dos rayos. [18]

A parte de los modelos de propagación para cada escenario, es necesario aplicar un modelo de atenuación de la señal en el caso del escenario urbano debido a los desvanecimientos producidos por los edificios. El modelo que utilizamos es el de Nakagami. Otra opción era utilizar el famoso modelo de Okumura Hata, pero este modelo es más apropiado para comunicaciones móviles que para lo nuestro.

El modelo de Nakagami que utilizamos implementa lo que llama el modelo de Nakagami para desvanecimiento rápido que lo que hace es tener en cuenta la variación de la potencia de la señal debido al desvanecimiento multicamino. Debido a que este modelo solo tiene en cuenta esto, y para nada tiene en cuenta la perdida de señal debido a la distancia recorrida por ella, lo usamos en combinación con el modelo logarítmico explicado previamente. Este modelo calcula las pérdidas en función de una densidad de probabilidad y está relacionada con la función gamma. Tiene dos parámetros importantes: m y Ω, las cuales una tiene que ser mayor o igual que ½ y la otra mayor que 0. Cuando m=2, la curva es igual a la densidad de probabilidad de la variable aleatoria de Rayleigh.

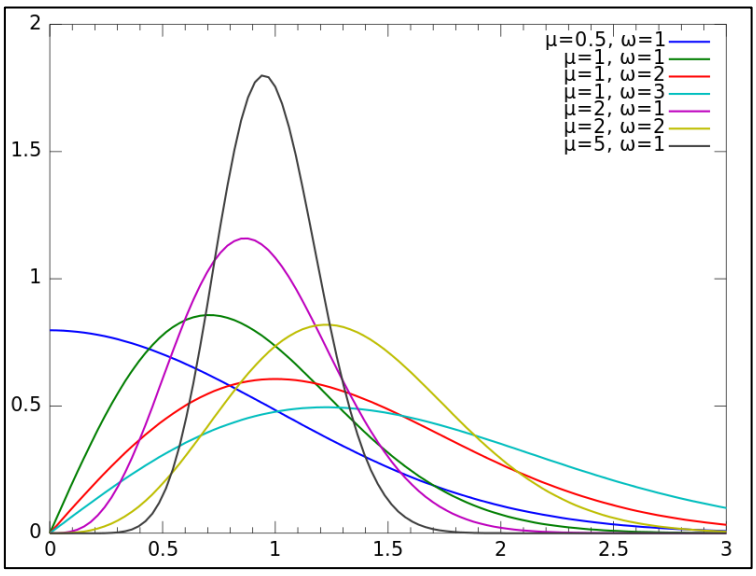


Figura 45: fdp de Nakagami.

4.3 Resultados

A continuación, vamos a exponer los resultados obtenidos en las simulaciones que hemos realizado en cada uno de los escenarios. En cada escenario realizamos varias interpretaciones de forma gráfica que nos ayudarán a analizar los distintos protocolos. Primero vamos a ver el urbano:

4.3.1 Escenario urbano

Primero vamos a detallar en una tabla la configuración que vamos a implementar para este escenario:

Parámetros de simulación	**Valores**
Duración	100 s
Número de nodos	20, 30, 40
Longitud X-axis	165 metros
Longitud Y-axis	10.5 metros
Velocidad de los nodos	7, 12, 16 (m/s)
Pausa	0 segundos
Protocolo MAC	802.11p
Protocolos	AODV, OLSR, DSDV
Potencia de transmisión	7.5 dB
Modelo de movilidad	RandomWayPoint
Tamaño paquete WAVE	200 bytes
Velocidad de TX WAVE	6 Mbps
Tamaño paquete enrutamiento	64 bytes
Velocidad de Tx enrutamiento	2.048 kbps
Modelo de propagación	LogDistance
Patrón de fading	Nakagami

Tabla 1: Escenario Urbano.

Lo primero que analizaremos es el Packet Delivery Ratio (PDR) respecto de la densidad de los nodos a las diferentes velocidades, que como veremos va disminuyendo conforme se le vayan aumentando los nodos.

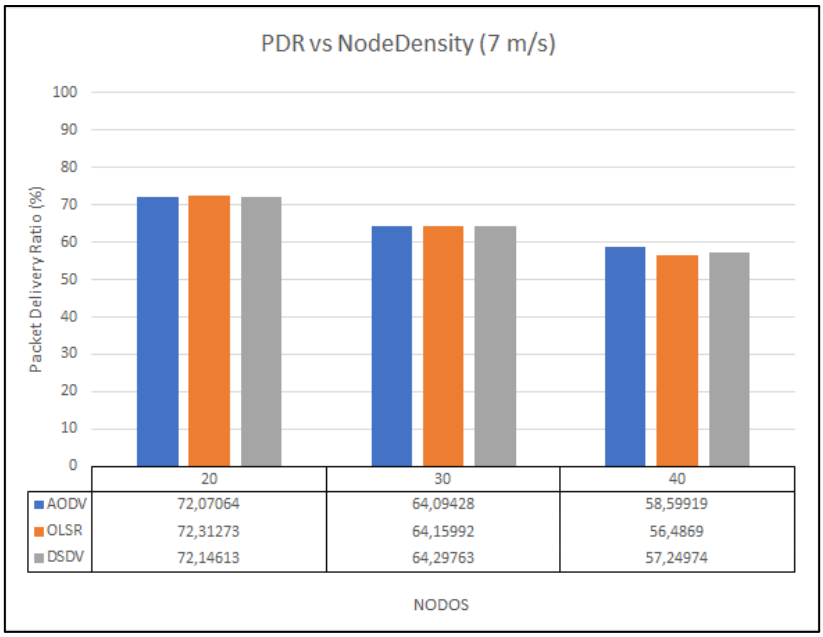


Figura 46: PDR vs NodeDensity @ 7 m/s.

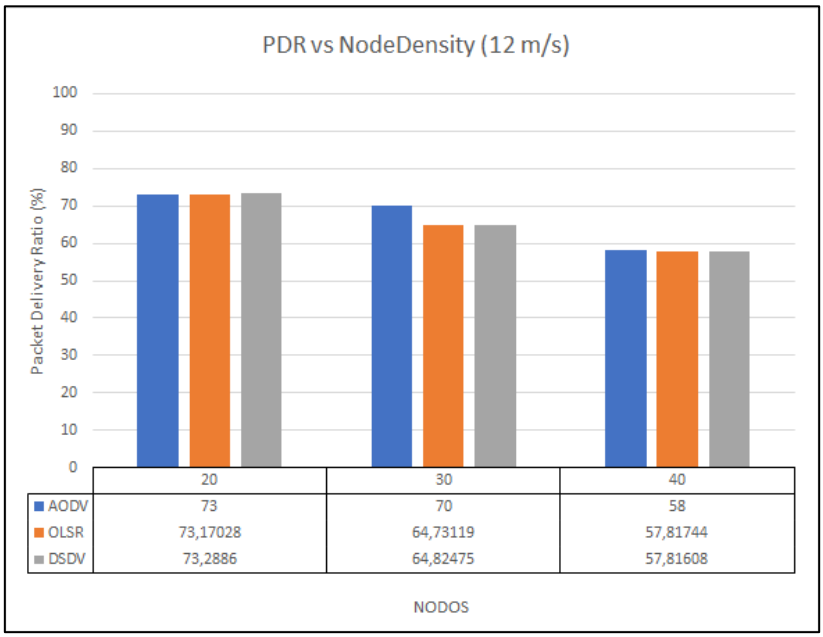


Figura 47: PDR vs NodeDensity @ 12 m/s.

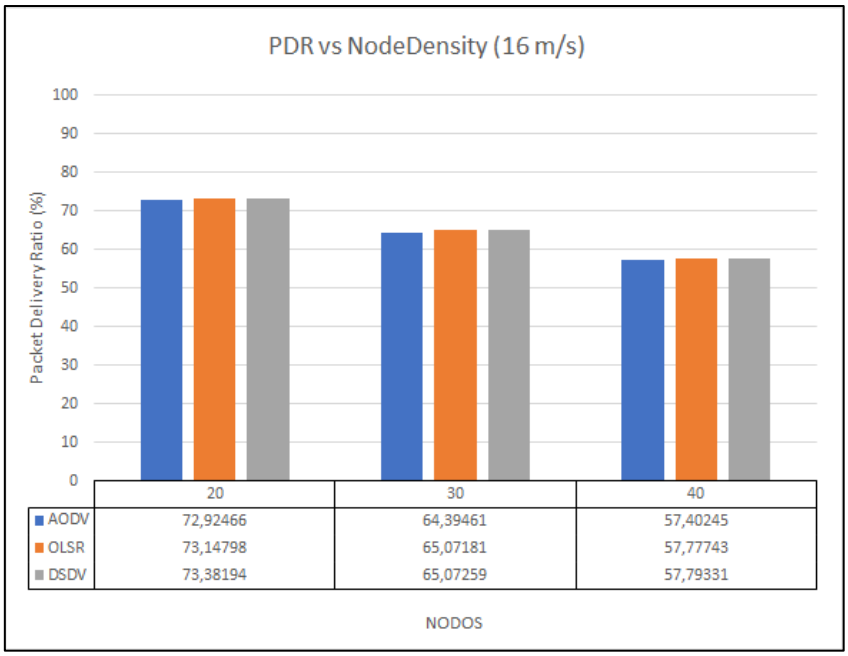


Figura 48: PDR vs NodeDensity @ 16 m/s.

Como podemos ver, en estas tablas se representa el Packet Delivery Ratio en función de la velocidad de los nodos y del número de ellos. Los valores del PDR se tratan de valores medios puesto que, como hemos explicado antes, hay diferentes distancias medidas. Evidentemente, a pocos metros del nodo, salían valores con el porcentaje cercano al 100%. Sin embargo, la media de los valores relacionados con las distancias es esa. Por lo tanto, como vemos, un aumento de nodos, incluso de un 10%, provoca una importante bajada en los valores del PDR (de 72,9% al 64%). Este cambio es mucho más visible cuando le añadimos 20 nodos más puesto que al aumentar el número de nodos en un espacio ciertamente reducido, hay más interferencia entre nodos y por lo tanto más pérdidas en los paquetes transmitidos. En cuanto a la variación de la velocidad en el entorno urbano, no entrañaría una gran bajada en el PDR. De hecho, en algunos casos, se produciría una ligera subida del PDR, aunque insignificante como para sacar algún dato concluyente. En cuanto a qué protocolo sería mejor en cuanto a su PDR, la elección sería algo complicada ya que, el AODV tiene ligera ventaja a velocidades de 7 y 12 m/s, pero el DSDV es mejor a 16 m/s. Aun así, las diferencias son tan mínimas, que no es razón para desechar ninguno de los tres.

A continuación, vamos a ver en qué medida afecta el OH y que protocolo puede ser más conveniente o menos según este parámetro.

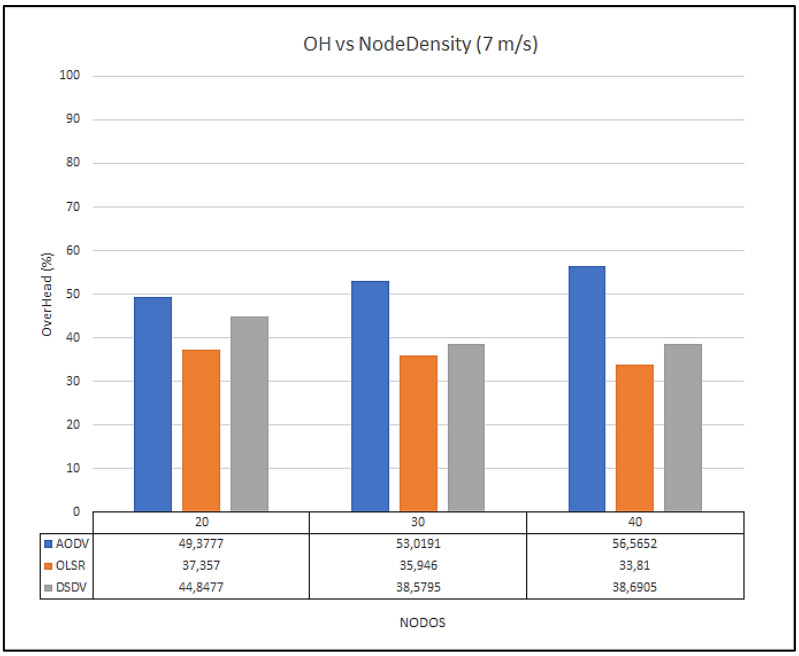


Figura 49: OverHead vs NodeDensity @ 7m/s.

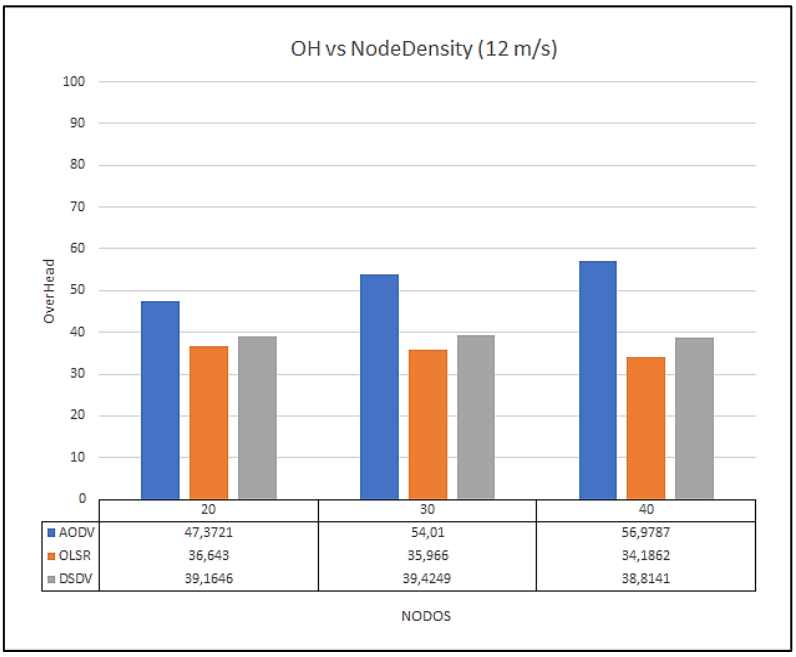


Figura 50: OverHead vs NodeDensity @ 12m/s.

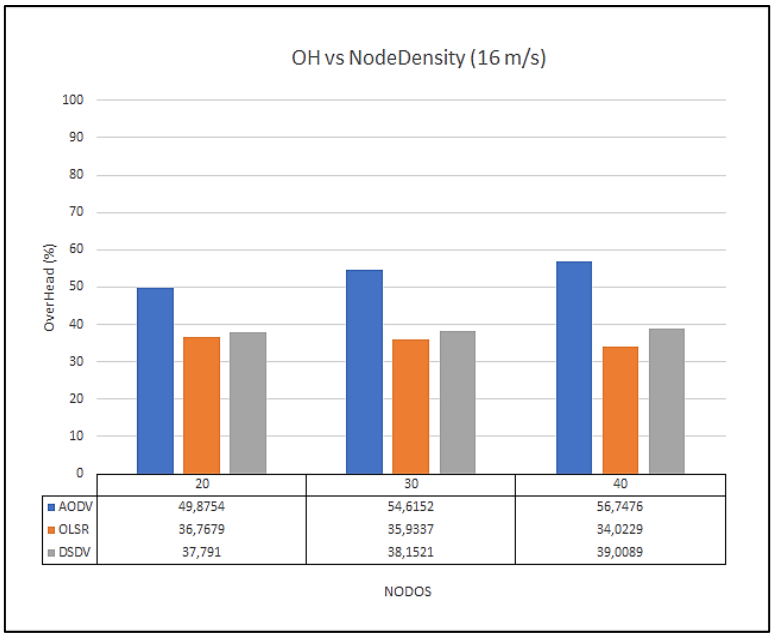


Figura 51: OverHead vs NodeDensity @ 16m/s.

Como se puede apreciar en las 3 gráficas, el protocolo más destacado por su alto porcentaje de overhead es el AODV. Es el peor protocolo en este campo, llegando a alcanzar valores de casi un 60% en el caso de 40 nodos a una velocidad de 16 m/s. Dicho valor, no es apropiado como podíamos sospechar ya que sabíamos, por su forma de enrutamiento usando route discovery y route maintenance, que tenía un alto overhead. Después, en cuanto a funcionalidad negativa, le sigue el DSDV y, con el que sacaríamos más partido en este sentido, sería con el OLSR que apenas llega a un 37%. Por lo tanto, fijándonos en este campo, elegiríamos o el OLSR o el DSDV.

Por último, en este escenario vamos a ver como es el GoodPut siguiendo el mismo tipo de análisis.

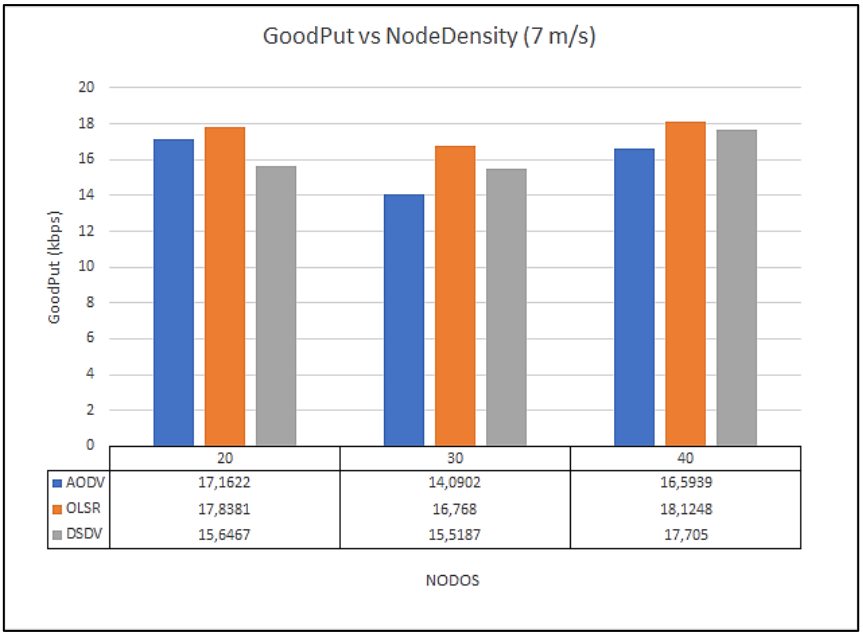


Figura 52: GoodPut vs NodeDensity @ 7m/s.

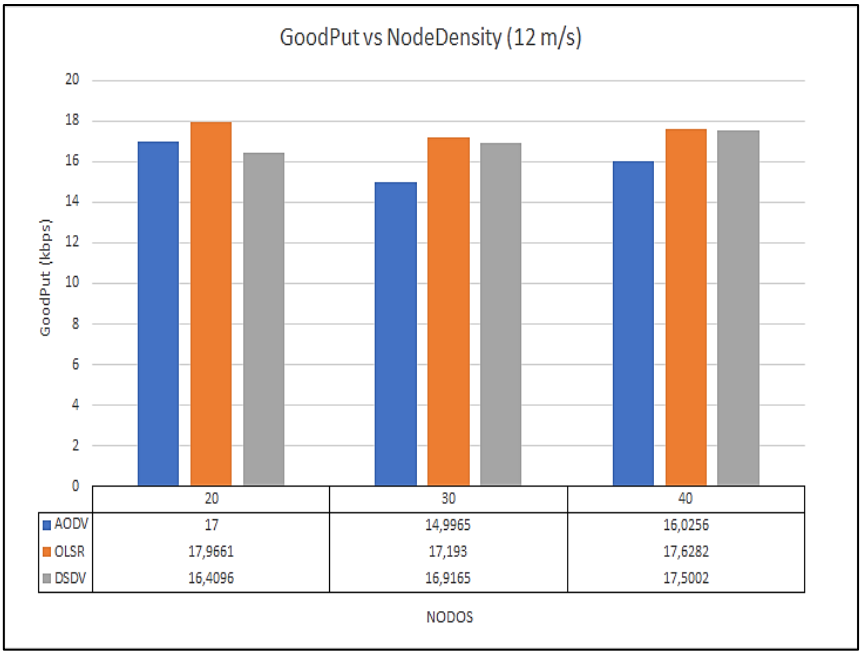


Figura 53: GoodPut vs NodeDensity @ 12m/s.

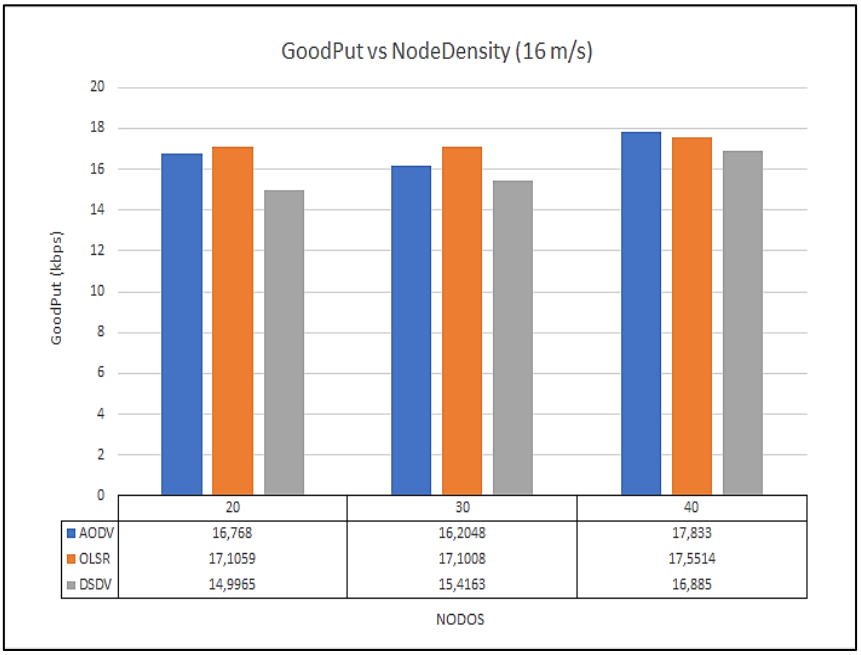


Figura 54: GoodPut vs NodeDensity @ 16 m/s.

Echando un vistazo a las tres tablas anteriores, podemos ver que, por lo general, el protocolo OLSR es el que mejor GoodPut tiene, independientemente del número de nodos y de la velocidad, salvo a 16 m/s que el AODV lo supera ligeramente con 40 nodos. Sin embargo, en el resto de las velocidades, el AODV es la velocidad más baja, ya que como su overhead es tan grande, le penaliza que aumente el número de nodos. A efectos de utilización, destacaría el OLSR o en ciertos momentos el DSDV, aunque nunca por decisión unánime. Una de las cosas interesantes

que podemos sacar del análisis del GoodPut es que decrece con la velocidad de los nodos. Esto se debe a que, a velocidades pequeñas, los nodos tienden a estar más alejados y a velocidades más altas, tienden a "acercarse". Por lo tanto, si están a más distancia, se necesita más velocidad para que la señal llegue sin que se desvanezca y a menor distancia, no es tan necesaria. Evidentemente, la diferencia no es muy grande porque las diferencias entre velocidades tampoco lo son.

4.3.2 Escenario rural

En el entorno rural el análisis no va a ser respecto a las distintas velocidades puesto que, en autovía tampoco hay tanta variación y al realizar las comprobaciones, las diferencias no eran tan evidentes como para poder analizarlas. Una de las diferencias más significantes en este escenario, es que no configuramos ningún patrón de fading puesto que una autovía se trata de un espacio abierto sin edificios colindantes que puedan interferir en la señal.

Analizaremos respecto de esta configuración:

Parámetros de simulación	**Valores**
Duración	100 s
Número de nodos	10 (5 sinks), 20, 30
Longitud X-axis	500, 1000 y 2000 metros.
Longitud Y-axis	10.5 metros
Velocidad de los nodos	33 m/s
Pausa	0 segundos
Protocolo MAC	802.11p
Protocolos	AODV, OLSR, DSDV
Potencia de transmisión	7.5 dB
Modelo de movilidad	RandomWayPoint
Tamaño paquete WAVE	200 bytes
Velocidad de TX WAVE	6 Mbps
Tamaño paquete enrutamiento	64 bytes
Velocidad de Tx enrutamiento	2.048 kbps
Modelo de propagación	TwoRayGround
Patrón de fading	Ninguno

Tabla 2: Escenario Rural.

Lo primero que vamos a analizar, como hicimos en el caso urbano, es el PDR respecto de la densidad de nodos. Esta vez, también vamos a añadir otra variable que va a ser el rango de acción de la RSU que va a tener 3 valores: 500 metros, 1 km y 2 km. Con esto, queremos reflejar cómo reacciona el sistema ante un cambio en la distancia entre nodos, puesto que la densidad disminuye.

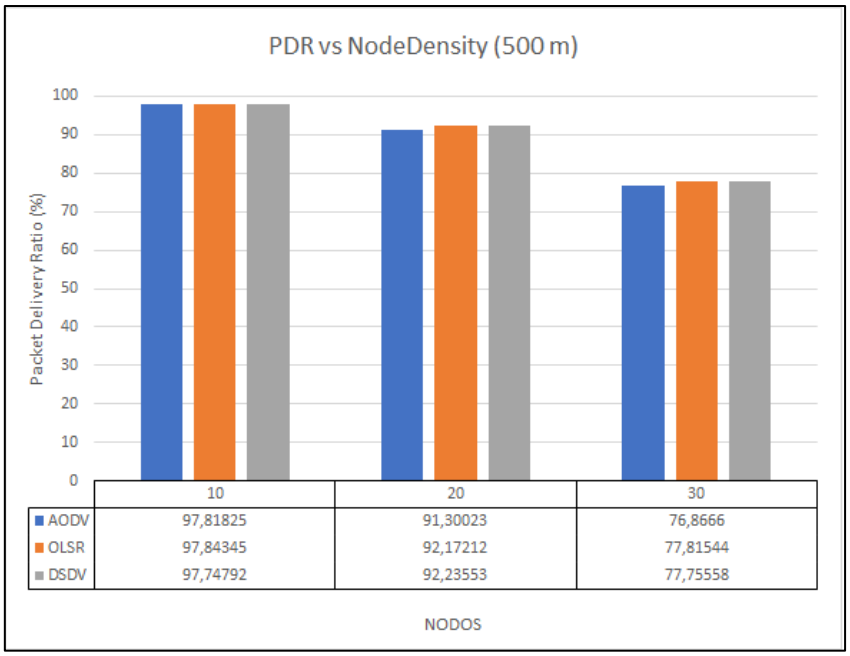


Figura 55: PDR vs NodeDensity @ 500 metros.

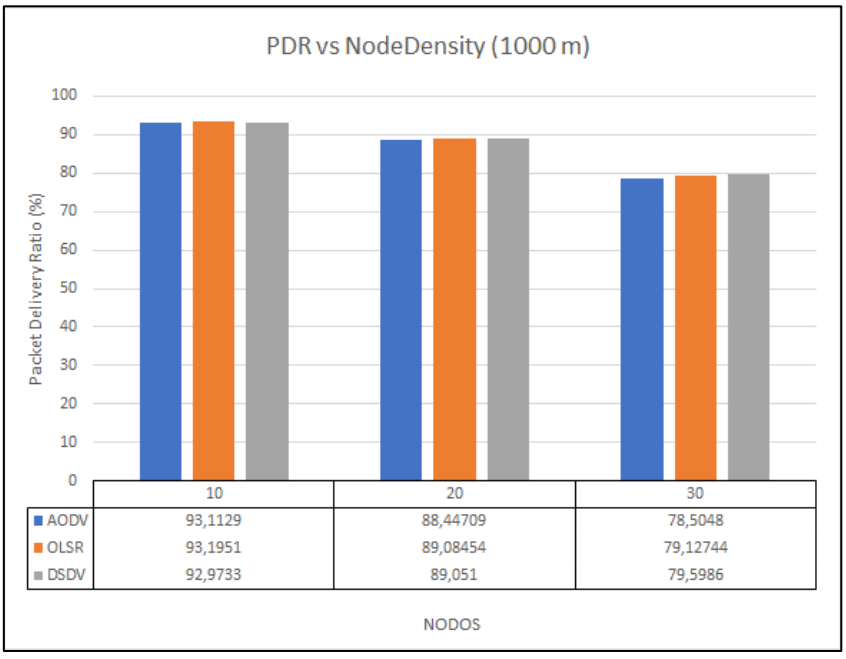


Figura 56: PDR vs NodeDensity @ 1000 metros.

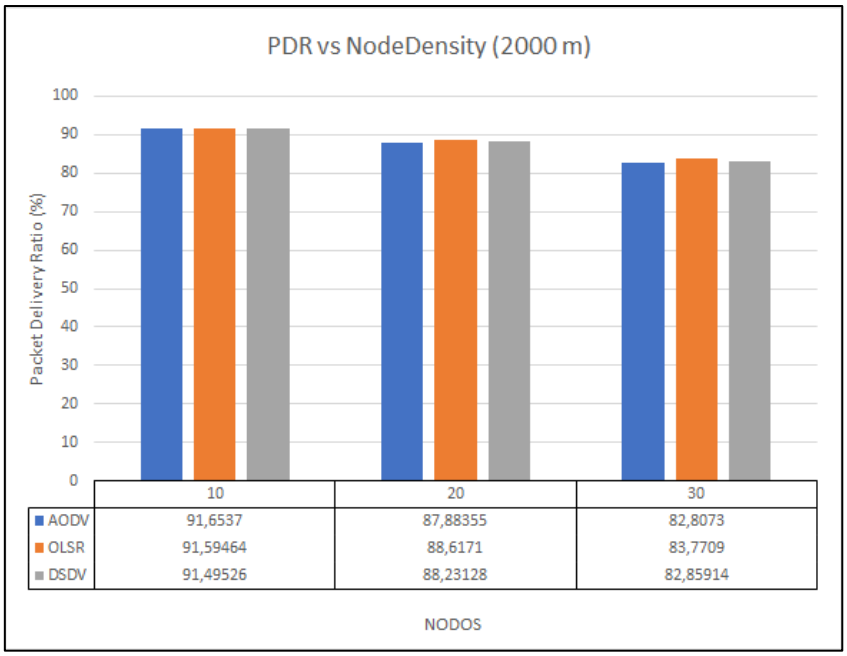


Figura 57: PDR vs NodeDensity @ 2000 metros.

Como hemos explicado en el caso urbano, el Packet Delivery Ratio (PDR), disminuye con el aumento de la velocidad. Aquí vemos que también disminuye si ampliamos el rango de la RSU. Como hemos dicho antes, se trata solo de la media de 10 distancias diferentes y para el caso rural vemos que el peor valor que obtenemos a una distancia bastante considerable y que es asumible en el peor de los casos. En cuanto a qué protocolo recoge los mejores resultados, tenemos varios casos, aunque las diferencias no son prácticamente importantes. Por una parte, para la distancia de 500 metros y 2 kilómetros, el que da un poco más de sí, es el OLSR y para 1 kilómetro es el DSDV. Sin embargo, como hemos dicho, no se trata de una diferencia importante, y se pueden considerar aptos los 3. La mayor diferencia que vemos es en la bajada que se produce cuando pasamos de tener 20 nodos a 30. Si observamos, de 10 a 20, la bajada de PDR es a lo sumo de un 6% y en el otro caso (aumentando de 20 a 30 nodos) la bajada llega a ser casi de un 20%, salvo en la distancia de 2 kilómetros, que parece que vuelve a mantenerse un poco. Aquí la diferencia es menor, aunque sigue siendo algo mayor que de 10 a 20 nodos. La razón por la cual a 2000 metros se vuelve a recuperar cuando el número de nodos es de 30, no es del todo clara. Podría ser porque el modelo de propagación desborde en ese punto y haga una medición algo errónea.

A continuación, vamos a hablar del Overhead a las distintas distancias.

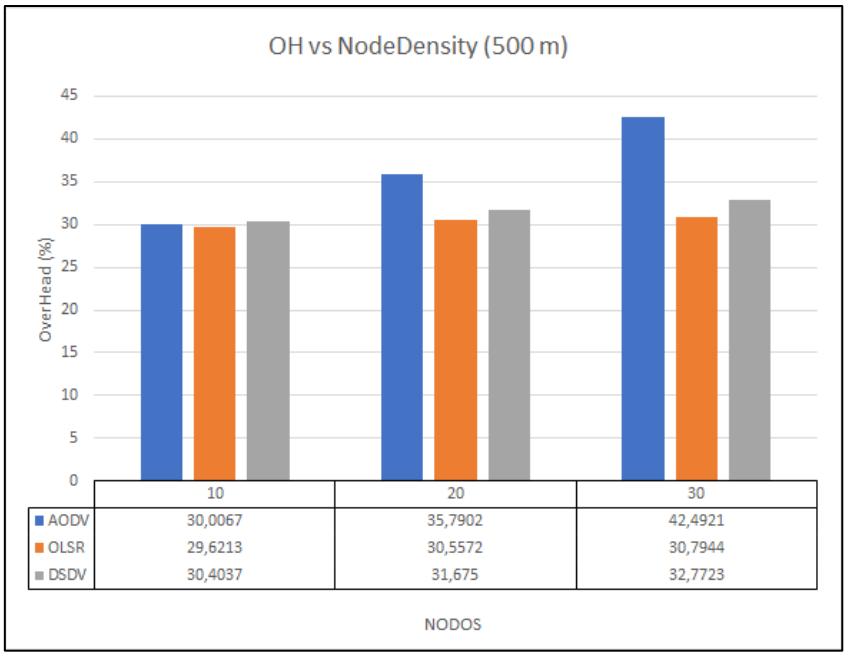


Figura 58: OverHead vs NodeDensity @ 500 metros.

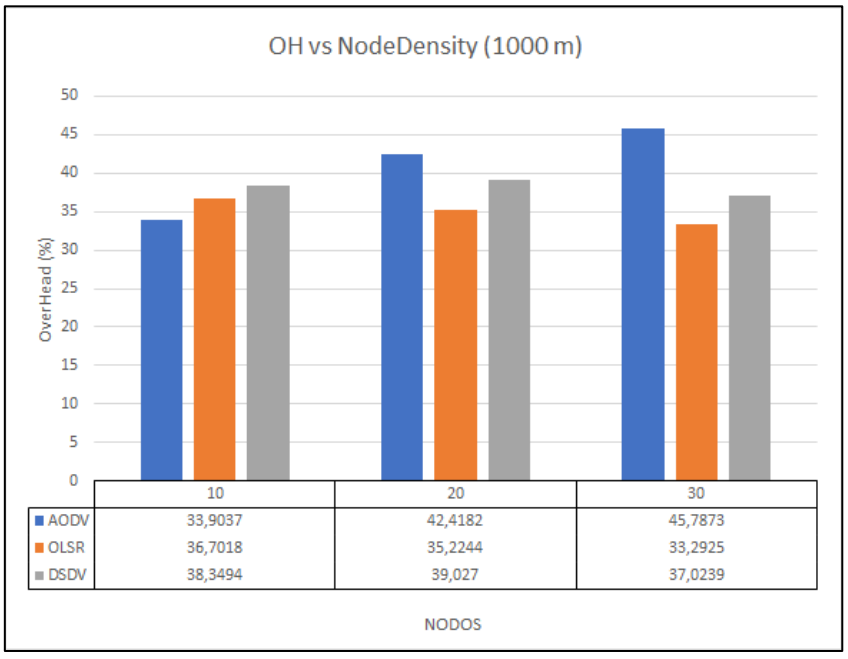


Figura 59: OverHead vs NodeDensity @ 1000 metros.

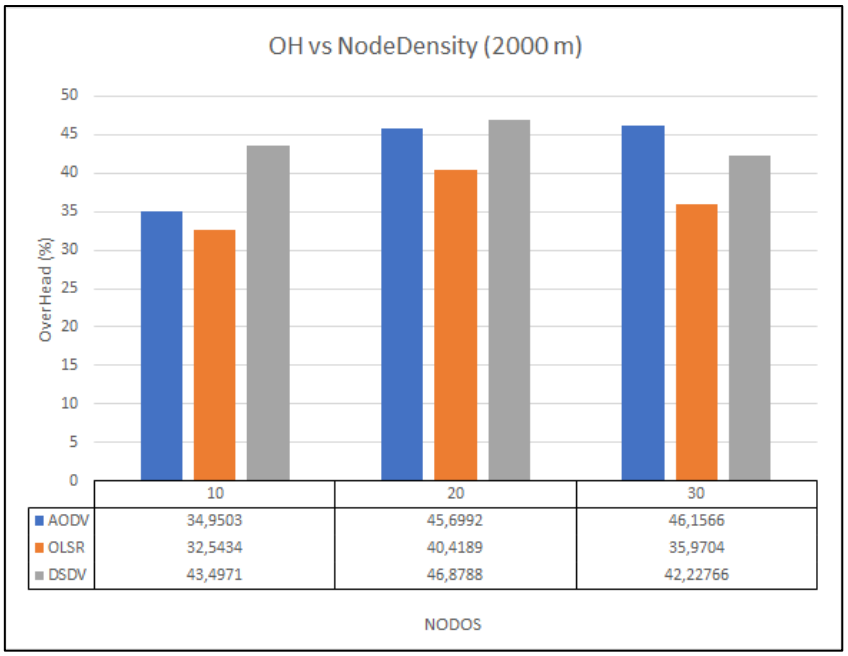


Figura 60: OverHead vs NodeDensity @ 2000 metros.

En la distancia de 500, con 10 nodos tenemos un overhead parecido en todos los protocolos. Sin embargo, al pasar a los 20 nodos, el valor del protocolo AODV se dispara en un 6%, mientras que los otros suben como mucho un 1%. En el último caso en esta distancia, (para 30 nodos) el valor del AODV se dispara hasta el 42,49%, creciendo casi un 7% respecto al valor anterior. Esto hace que este protocolo no sea válido si tomamos en cuenta que la cantidad de nodos o vehículos es muy variable en una situación real. En los otros dos protocolos, aunque aumentemos la cantidad de nodos, su crecimiento sigue siendo ínfimo. En la distancia de 1000 metros con 10 nodos, sorprendentemente, el protocolo AODV es el que menos overhead genera. Sin embargo, cuando aumentamos el número de nodos, vuelve a sobrepasar al resto de los protocolos con un crecimiento del casi 9%. Los otros dos protocolos siguen bastante dentro de sus datos, sin grandes crecimientos. Pasando a la figura 60, donde analizamos el caso a una longitud de 2000 metros, vemos que en el primer caso el DSDV se dispara, y los otros dos va casi a la par con el AODV salvo con 30 nodos, que vuelve a imponerse. El crecimiento del AODV llega a ser hasta de un 10%, lo cual es bastante. Por lo que se ve en estas gráficas, aquí el mejor protocolo sería el OLSR que solo llega al 40% de overhead en el peor de los casos.

Ahora vamos a por el último análisis que va a ser sobre el GoodPut. Antes de mostrar las gráficas cabe destacar que, para los valores de 10 nodos, hemos tenido que realizar una normalización ya que el número mínimo de sinks que nos dejaba el programa utilizar para dicha cantidad de nodos era 5, y para el resto son 10. Por lo tanto, dichos valores están transformados de esa manera. Esto tiene sentido ya que, si tenemos 10 nodos, no pueden ser los 10 sinks.

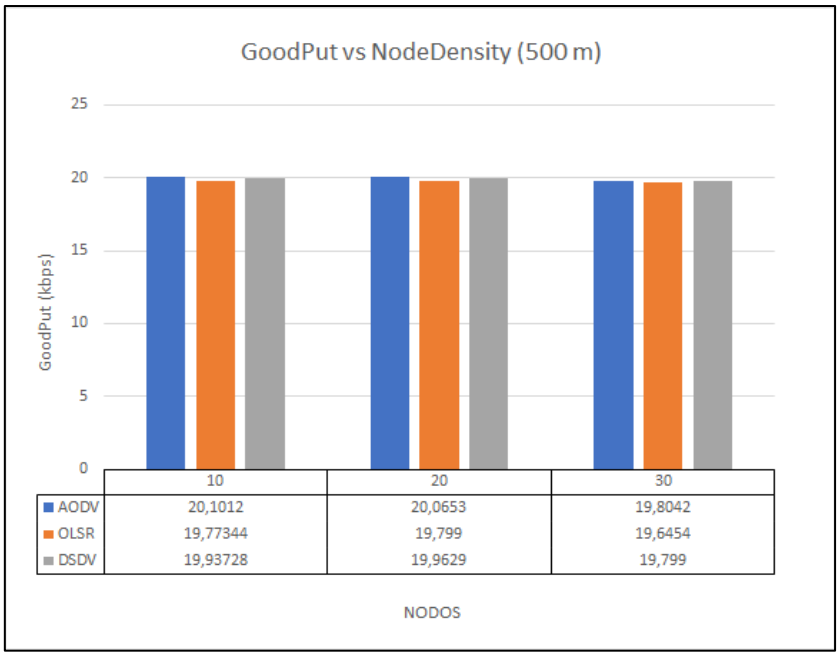


Figura 61: GoodPut vs NodeDensity @500 metros.

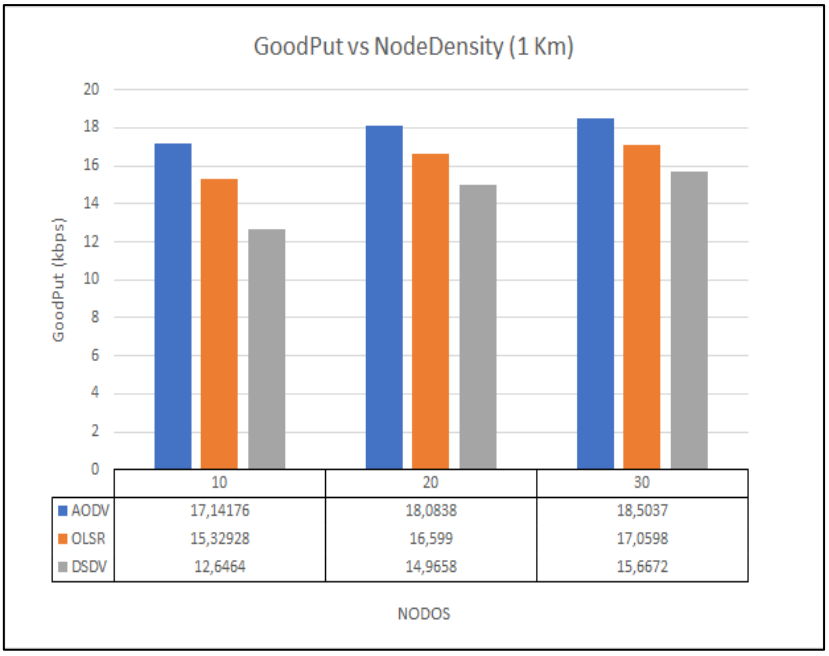


Figura 62: GoodPut vs NodeDensity @1000 metros.

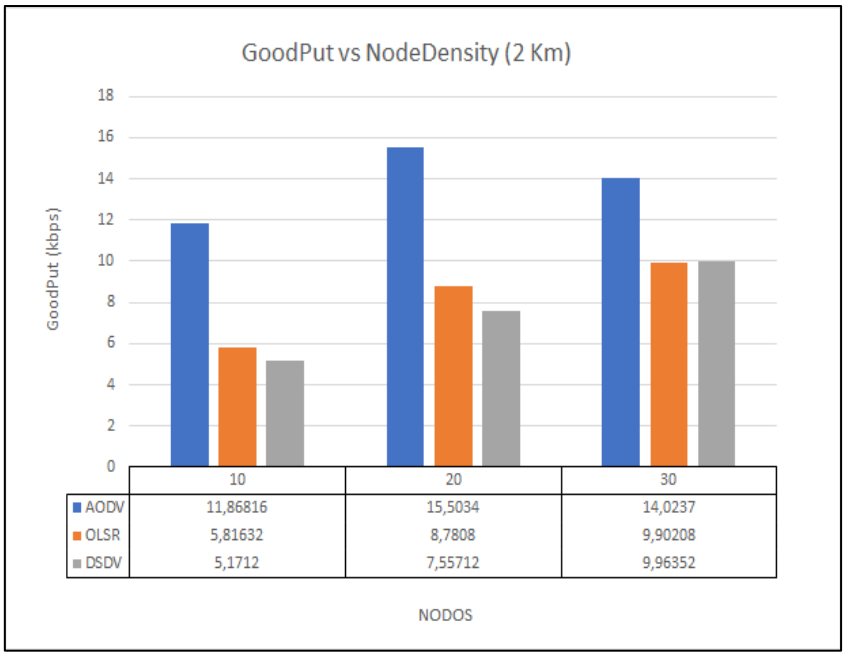


Figura 63: GoodPut vs NodeDensity @2000 metros.

El GoodPut en este escenario toma bastante importancia puesto que, al aumentar la distancia, se debe de incrementar la velocidad si no, habría una sustancial pérdida en la recepción. Para el valor de 500 metros, las velocidades son bastante parejas rondando los 20 kbps. Sin embargo, para valores de 1 kilómetro, tenemos que tanto OLSR como DSDV tienen algo más de bajada en su rendimiento, sobre todo DSDV que pierde hasta 7 kbps. AODV, apenas baja y mantiene una más que aceptable velocidad. Por último, en la amplitud de 2 kilómetros es donde se ve la mayor diferencia sobre todo en los protocolos OLSR y DSDV. AODV pierde aun así bastante pero aun así aguanta un mínimo de 12 kbps. Sin embargo, los otros dos protocolos no llegan ni a 10 kbps e incluso DSDV casi baja de 5 kbps. Esto lleva a pensar que estos dos protocolos, no tienen cabida en una simulación con esta amplitud. Por ello, el mejor protocolo y el más robusto en cuanto velocidad y cambios en la longitud, es claramente el AODV. Otra de las cosas que también podemos comentar que ya hemos dicho antes, es que, en la mayoría de los casos, si aumentamos la densidad de nodos, aumenta también el GoodPut. Esto seguramente es porque, para el caso por ejemplo de 20 nodos, tenemos 10 sinks; si aumentamos 10 nodos más, seguimos teniendo las mismas sinks, pero el número de transmisores ha aumentado un 10% y por lo tanto necesitan más velocidad binaria.

Capítulo 5: Conclusiones y líneas futuras

5.1 Conclusiones

Las conclusiones vamos a estructurarlas primero según cada uno de los escenarios y luego comunes, es decir, de lo específico a lo general.

En la parte urbana, vemos que en cuanto al PDR no podemos priorizar ninguno de los 3 puesto que no hay tanta diferencia como para poder hacerlo. Sí es verdad que en las dos velocidades más bajas podemos ver que el AODV es ligeramente mejor, pero en la velocidad más alta prevalece el DSDV. Pero en este caso, no sería significativo. En cuanto al overhead, está claro que el AODV sale perdiendo debido a su "route maintenance" y "route discovery" que eleva mucho dicho valor. En este caso, según los resultados, elegiríamos OLSR. En cuanto al GoodPut, en todas las medidas menos en una, el algoritmo OLSR está por encima de los otros dos. Si no, con la menor cantidad de nodos (20) le seguiría el AODV, pero según vamos aumentando los nodos, DSDV está por encima de AODV. Con lo cual, podemos decir que OLSR es mejor y AODV funciona bien con poca densidad, y DSDV le puede intentar seguir el ritmo a OLSR en el resto de las densidades establecidas.

En cuanto al escenario rural, pasa un poco lo mismo en cuanto al PDR. Las diferencias son tan pequeñas que no podemos determinar que un protocolo sea mejor que otro. En el overhead, sigue imponiéndose AODV de forma negativa por su elevado porcentaje, y sigue aguantando mejor este parámetro el OLSR que es el que menor porcentaje tiene en prácticamente todas las mediciones. Sin embargo, podemos decir que DSDV empieza a tener problemas a distancias superiores al 1 kilómetro y se dispara casi tanto como el AODV. OLSR también crece, pero lo hace mucho menos. Llegando al GoodPut, vemos que a 500 metros todos los protocolos reaccionan de forma parecida pero una vez que se sobrepasa esa longitud, AODV mantiene prácticamente las velocidades, mientras que OLSR y DSDV pierden bastante. Podemos decir que el protocolo más rápido para distancias largas es el AODV y DSDV es, previsiblemente, el peor.

En resumen:

- En cuanto a PDR, no tenemos limitación de elección entre los protocolos.
- Podemos ver, que a mayor cobertura de la RSU (distancias rurales) el PDR es mayor en general por la dispersión en el espacio por parte de los nodos (disminuye la densidad y la interferencia entre ellos).
- El peor protocolo para OverHead en distancias cortas es el AODV y el mejor OLSR y DSDV. A partir de 1 km, AODV y DSDV son los peores y OLSR se mantiene.
- En el GoodPut a distancias pequeñas y poca densidad el AODV funcionaria bien, sin embargo, cuando aumenta la densidad de nodos, funciona mejor el OLSR.
- A grandes distancias, indiscutiblemente el mejor GoodPut es el del protocolo AODV y el peor el DSDV.
- AODV alto end-to-end delay y DSDV bajo end-to-end delay [16].

PROTOCOLO	VENTAJAS	INCONVENIENTES
AODV	Alto GoodPut para distancias largas.	Alto Overhead Alto end-to-end delay [16].
OLSR	Bajo OverHead	Bajo GoodPut a distancias largas.
DSDV	Bajo end-to-end delay [16]	Bajo GoodPut a distancias largas. Alto OverHead a distancias >1km.

Tabla 3: Conclusiones.

5.2 Líneas futuras

En cuanto a este tema, el enfoque de este trabajo ya trata un campo que, en el caso de los sensores, se está empezando a implementar, pero que en el caso de los algoritmos y los sistemas de tráfico autónomo no hay todavía nada. Sí que se están implementando coches autónomos, como por ejemplo el Tesla, pero sin grandes resultados y con algunas polémicas por fallos en experimentos reales. Por lo tanto, una de las líneas generales que se está llevando a cabo, es la de corregir los errores y optimizar dicha conducción autónoma, así como introducir multitud de sensores para su mejora. Otra de las líneas de investigación relacionadas con la conducción autónoma también es la seguridad y la detección de vulnerabilidades. Como en todo ámbito en donde se utiliza la tecnología, hay que protegerlo de amenazas externas que, en este caso, serían de gran importancia puesto que significaría una pérdida del control del vehículo y, por consiguiente, un accidente. Otra de las cosas más significativas también es la cuestión de si cuando salga el 5G (en un futuro cercano) va a ser una tecnología más apta que el 802.11p o no. Esto supondría poder utilizar la infraestructura ya realizada para redes móviles y además poder combinarlo con SDN, lo cual es otra línea de investigación futura. De momento, los datos se siguen decantando por la tecnología WiFi, pero habrá que ver cómo se desarrolla todo. En cuanto a los algoritmos, se está intentando ir más hacia los algoritmos de posicionamiento que tengan en cuenta la tecnología GPS que ya iría integrada en el coche, y que facilitaría el posicionamiento.

Resumiendo, las líneas de investigación serían las siguientes:

- Optimización de vehículos autónomos.
- Desarrollo de seguridad contra ataques externos.
- Algoritmos de posicionamiento.
- 5G combinado con SDN.

Bibliografía

[1] Fotografía escenario VANET. [Fecha de última consulta: 17-06-2019].

https://www.car-2-car.org/

[2] Historia y primeros proyectos VANET. [Fecha de última consulta: 17-06-2019].

https://www.researchgate.net/publication/308084262_Redes_vehiculares_Ad-hoc_aplicaciones_basadas_en_simulacion

[3] Carla Fabiana Chiasserini, Connected Cars (2018). [Fecha de última consulta: 17-06-2019]

Connected-Cars.pdf (Archivo local de una asignatura impartida en Erasmus en el Politecnico di Torino)

[4] Datos de DSRC.

E. C. Eze, S. Zhang and E. Liu, "Vehicular ad hoc networks (VANETs): Current state, challenges, potentials and way forward," 2014 20th International Conference on Automation and Computing, Cranfield, 2014, pp. 176-181.

URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6935482&isnumber=6935445

[5] Foto canals DSRC.

D. J. Jerry, L. Thomas, S. T. Panicker, S. R, J. T. Mathew and B. V. Joe V S, "Safety Alert Systems Using Dedicated Short-Range Communication for on Road Vehicles," 2018 International CET Conference on Control, Communication, and Computing (IC4), Thiruvananthapuram, 2018, pp. 216-219.

URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8530930&isnumber=8530878

[6] OBU y RSU. [Fecha de última consulta: 17-06-2019]

http://bibing.us.es/proyectos/abreproy/12196/fichero/PFC_Jose_M_Sampedro.pdf

[7] RSU. [Fecha de última consulta: 17-06-2019]

https://w3.usa.siemens.com/mobility/us/en/road-solutions/traffic-management/Documents/Siemens%20RSU%20Brochure_NEW.pdf

[8] OLSR. [Fecha de última consulta: 17-06-2019]

https://www.rfc-editor.org/rfc/pdfrfc/rfc3626.txt.pdf

[9] Algoritmos proactivos y reactivos. Formulas. [Fecha de última consulta: 17-06-2019].

https://www.matec-conferences.org/articles/matecconf/pdf/2018/09/matecconf_mucet2018_06024.pdf

[10] DSDV. [Fecha de última consulta: 17-06-2019].

https://www.academia.edu/21197068/Destination-Sequenced_Distance_Vector_DSDV_Routing_Protocol_Implementation_in_ns-3

[11] AODV. [Fecha de última consulta: 17-06-2019].

https://www.ietf.org/rfc/rfc3561.txt

[12] DSR. [Fecha de última consulta: 17-06-2019].

https://www.rfc-editor.org/rfc/rfc4728.html

[13] Position-Based Routing. [Fecha de última consulta: 17-06-2019].

http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=967595&isnumber=20878

[14] GPSR. [Fecha de última consulta: 17-06-2019].

https://www.eecs.harvard.edu/~htk/publication/2000-mobi-karp-kung.pdf

[15] Ns3. [Fecha de última consulta: 17-06-2019].

https://www.nsnam.org

[16] Review protocolos.

T. E. Ali, L. A. Khalil al Dulaimi and Y. E. Majeed, "Review and performance comparison of VANET protocols: AODV, DSR, OLSR, DYMO, DSDV & ZRP," *2016 Al-Sadeq International Conference on Multidisciplinary in IT and Communication Science and Applications (AIC-MITCSA)*, Baghdad, 2016, pp. 1-6.

URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7759934&isnumber=7759899

[17] NetAnim. [Fecha de última consulta: 17-06-2019].

https://www.nsnam.org/wiki/NetAnim_3.108

[18] Foto modelo 2 rayos. [Fecha de última consulta: 17-06-2019].
https://en.wikipedia.org/wiki/Two-ray_ground-reflection_model

[19] Carla Fabiana Chiasserini, Routing in Wireless D2D Networks (2018).
D2D.pdf (Archivo local de una asignatura impartida en Erasmus en el Politécnico di Torino).