Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Laboratorio de Comunicación y Redes

Desarrollo de un Conjunto de Benchmarks para la Evaluación de Desempeño del Simulador NCTUns en Redes Vehiculares

Trabajo Especial de Grado
presentado ante la Ilustre
Universidad Central de Venezuela
por el Bachiller:

Moises E. Chachati R.
V-19.089.339
moiseschachati@gmail.com

para optar al título de Licenciado en Computación

Tutor**:** Prof. Eric Gamess

Caracas, Abril 2014

Agradecimientos

Primero que nada gracias a Dios por darme salud durante esta etapa de mi vida y permitirme trabajar para cumplir el sueño de graduarme como licenciado en computación en la casa de estudios más grande de Venezuela, la UCV.

A mis padres Marcos y Yolanda. Sin ellos absolutamente nada de esto hubiese sido posible. Los quiero mucho, gracias por siempre brindarme todo el apoyo que he necesitado.

A mi tía Jima Chachati por el apoyo incondicional que siempre me ha brindado para cumplir todas las metas que me he trazado. Muchas gracias, te quiero tía.

A Emily Tenias por estar siempre allí a mi lado tanto en las buenas como en las malas. Gracias por tu paciencia y por el amor que me brindas. Gracias por soportarme cuando estoy de mal humor, te amo mi bella. Gracias al Sr. Pedro y la Sra. Isabel por su apoyo incondicional y por sus consejos que me ayudan a tomar siempre buenas decisiones.

A mis hermanos por siempre estar pendiente de mi progreso en la universidad y en este trabajo. Gracias Manuel y Miguel por compartir esos momentos de recreación a través de la música y los juegos de video que me permiten salir un poco de la rutina. Gracias Yesenia por servir siempre de ejemplo para nosotros. Los quiero mucho.

Al profesor Eric Gamess por transmitirme tanto conocimiento y por servir de guía durante los últimos años de mi carrera. Gracias por darme su confianza y por permitirme trabajar en esta área tan interesante. Es usted un gran profesor.

A Karima Velasquez y Francisco Sans por ser mi compañía en el laboratorio. Gracias por estar pendiente de mi progreso en este trabajo y por darme ánimo. Gracias Karima por poner música en tu oficina para que el ambiente en el laboratorio no sea tan monótono.

A los profesores de computación de la UCV por ayudar en mi formación como profesional. Gracias a ustedes nosotros los estudiantes de esta universidad podemos dejar en alto el nombre de esta casa de estudios.

Quiero expresar mi gratitud al Consejo de Desarrollo Científico y Humanístico de la UCV (CDCH-UCV) quien financió parcialmente este trabajo con el proyecto PG 03-8066-2011/1.

A todas aquellas personas que de una manera u otra colaboraron para la realización de este trabajo especial de grado. Mil gracias.

Moises Chachati

Resumen

TÍTULO:
Desarrollo de un Conjunto de Benchmarks para la Evaluación de Desempeño del Simulador NCTUns en Redes Vehiculares.

AUTOR:
Moises E. Chachati R.

TUTOR:
Prof. Eric Gamess

El presente Trabajo Especial de Grado consiste en el desarrollo de un conjunto de benchmarks para realizar un análisis de desempeño del simulador NCTUns v6.0 en el área de las redes vehiculares.

Las redes vehiculares se han convertido en una importante área de investigación debido a que procuran mejorar la seguridad vial, la eficiencia en el tráfico y la comodidad para los conductores y pasajeros de los vehículos. Los simuladores de redes son considerados como la mejor herramienta para validar algoritmos y protocolos en el campo de las redes vehiculares debido a que abaratan los costos, evitan riesgos de pérdidas de vidas humanas y permiten adaptar y repetir los experimentos con mayor facilidad. La simulación de redes vehiculares requiere tomar en cuenta dos aspectos: la comunicación entre nodos y el movimiento de los nodos. Existen simuladores que proveen las herramientas necesarias para la simulación de redes vehiculares tomando en cuenta ambos aspectos, como por ejemplo el simulador NCTUns (National Chiao Tung University Network Simulator).

NCTUns es un simulador y emulador de redes que permite la simulación de varios dispositivos y protocolos usados en redes cableadas o inalámbricas. Su diseño está basado en una metodología de simulación novedosa que permite utilizar aplicaciones de la vida real en las simulaciones. Utiliza la pila de protocolos TCP/IP para generar resultados de alta fidelidad.

En este trabajo se desarrollan dos tipos de benchmarks: (1) un benchmark circular, en el cual la topología de la red vial está basada en una carretera circular, (2) un benchmark de una ciudad, basado en una red de carreteras tomada de mapas reales utilizando trazas de ns-2. Los benchmarks pueden ser configurados mediante una gran variedad de parámetros (número de vehículos, protocolo de enrutamiento, *bitrate*, rango de transmisión de la señal, etc.) y se reportan estadísticas acerca del desempeño de los protocolos de enrutamiento (*delay*, número de saltos, PDR, NRL, etc.). Además se reportan estadísticas acerca del desempeño del simulador en sí como el tiempo real de simulación, el consumo de memoria y el uso del CPU. Finalmente se realiza un análisis de los resultados obtenidos en las distintas pruebas realizadas y se llega a la conclusión que ADV es el protocolo de enrutamiento más adecuado para redes vehiculares implementado en NCTUns. Entre AODV y DSDV, la elección dependerá de muchos factores, como la topología vial y la velocidad de los vehículos.

Palabras Claves: Simuladores de Redes, NCTUns, Redes Vehiculares,Benchmarks.

Tabla de Contenido

Índice de Figuras

Índice de Tablas

1. Introducción

Las redes vehicularesson un campo de investigación que ha ido creciendo en los últimos años debido a los avances en las tecnologías de comunicación.Estas redespueden ser utilizadas para una amplia gama de aplicaciones, permitiendo servicios como: seguridad vial, pago automático de peajes, gestión de tráfico vehicular, mejora de la navegación, servicios de localización y aplicaciones para el entretenimiento como acceso al Internet, entre otros.Sin embargo, realizar experimentos de redes vehiculares en la vida real puede ser un trabajo difícil, peligroso y costoso. Por estasrazones los simuladores se han convertido en herramientas importantespara las investigaciones en el área de redes vehiculares.

La simulación de redes vehiculares está relacionada con dos tipos de simulaciones: la simulación de redes y la simulación de tráfico. Los simuladores de tráfico se utilizan para simular la movilidad de los vehículos en las carreteras. Los simuladores de redes se utilizan para el envío de paquetes. Existen varias herramientas para la simulación de redes vehiculares: (1) simuladores de redes, (2) simuladores de tráfico y (3) herramientas que integran ambos tipos de simuladores como por ejemplo el simulador NCTUns.

NCTUns no solamente integra ambos tipos de simulación sino que también posee características que lo hacen único en comparación con otras herramientas disponibles para la simulación de redes vehiculares. Es importante que los resultados arrojados por los simuladores de red sean resultados realistas. Además, es deseable que las herramientas de simulación sean lo más eficientes posible para reducir el tiempo que toma realizar las simulaciones. Por estas y otrasrazones sonde gran importancialos estudios de desempeño realizados a simuladores de redes vehiculares existentes.

El resto del trabajo se encuentra organizado de la siguiente manera:

- **Capítulo 2:** Presenta la descripción del problema.
- **Capítulo 3:**Hace una introducción a las redes vehiculares.
- **Capítulo 4:**Presenta los enfoques existentes para la simulación de redes vehiculares y se describen algunos simuladores.
- **Capítulo 5:**Describe la arquitectura y las características más resaltantes del simulador y emulador de redes NCTUns.
- **Capítulo 6:**Presenta trabajos relacionados con la evaluación de desempeño de simuladores de redes vehiculares y estudios que utilizan herramientas de simulación para el desarrollo de tecnologías para redes vehiculares.
- **Capítulo 7:**Describe elMarco Metodológico.
- **Capítulo 8:**Plantea el Marco Aplicativo.
- **Capítulo 9:** Describe las pruebas realizadas, el análisis de los resultados y muestra las especificaciones técnicas de los benchmarks desarrollados.
- **Capítulo 10:**Se discuten las conclusiones obtenidas producto de la investigación realizada y se muestra la propuesta de trabajos futuros.

2. El Problema

En este capítulo se exponen los argumentos que justifican el diseño y desarrollo de un conjunto de benchmarks ideados para simular redes vehiculares y evaluar el desempeño de los simuladores de red, así como el planteamiento de nuevos objetivos y alcances para obtener una solución eficaz.

2.1 Planteamiento del Problema

Las redes vehiculares son un campo de investigación que ha ido creciendo en los últimos años. Las redes vehiculares son de gran importancia ya que su objetivo principal es mejorar la seguridad física y prevenir la congestión vehicular. Se han desarrollado estándares para la comunicación V2V (*Vehicle-to-Vehicle*) y V2I (*Vehicle-to-Infrastructure*) en redes vehiculares como el estándar IEEE 802.11p y la familia de estándares IEEE 1609.X. También se han propuesto protocolos de enrutamiento para las redes vehiculares y aplicaciones para mejorar la seguridad vial y evitar el congestionamiento vehicular.

Realizar pruebas a redes vehiculares en la vida real es una tarea difícil y costosa. Para que las pruebas sean realistas se debe incluir un gran número de vehículos los cuales deben estar equipados con la tecnología que permita la comunicación inalámbrica. Además, algunas pruebas pueden llevar riesgos para los conductores involucrados. Es por esto que la simulación pasa a ser parte importante en el campo de investigación de las redes vehiculares. Las simulaciones permiten involucrar un gran número de vehículos para generar resultados realistas. Permiten repetir los experimentos numerosas veces cambiando los parámetros necesarios sin exponer la integridad de los conductores.

Existen varias herramientas para realizar simulaciones de redes vehiculares: (1) simuladores de tráfico, (2) simuladores de redes y (3) herramientas que integran ambos tipos de simulación. Para los investigadores, es importante tener referencias de los simuladores existentes para escoger el más adecuado para su simulación. Otro factor importante es escoger un protocolo de enrutamiento que cumpla lo mejor posible con las simulaciones a realizar. Por esta razón, se propone hacer un estudio del simulador NCTUns en el caso de las redes vehiculares. Los Benchmarks realizados reportarán resultados como:

- El tiempo real de simulación: Es el tiempo (en segundos) que transcurre desde que se inicia la simulación hasta que termina. Es diferente al tiempo de simulación que es el tiempo virtual que transcurre en la simulación.
- Consumo de memoria: Es la cantidad de memoria consumida por el simulador durante la ejecución de la simulación.
- El *One-Way Delay* (OWD): Representa el tiempo que le toma a un paquete para ir del emisor hasta el receptor. Se calcula tomando una muestra de tiempo antes de enviar el paquete y otra muestra de tiempo cuando llega el paquete a su destino final. La diferencia entre las dos muestras de tiempo es el OWD. Para esta métrica son de interés el OWD mínimo, máximo y promedio.
- El número de saltos: Corresponde al número de saltos que hacen los paquetes desde su envío hasta llegar al destino. Se calcula examinando el campo TTL (Time to Live) de los paquetes en el receptor. Para esta métrica, son de interés el número de saltos mínimo, máximo y promedio.

- El *Packet Delivery Ratio* (PDR): Es la relación entre el número de paquetes de datos recibidos por los receptores y el número de paquetes enviados por los emisores. Se evalúa en términos de porcentaje (%).
- El *Normalized Routing Load* (NRL): Es usado para calcular la eficiencia del protocolo de enrutamiento. Es la relación entre el número de paquetes enviados por el protocolo de enrutamiento y el número de paquetes de datos recibidos.

2.2 Justificación

El simulador y emulador de redes NCTUns es una herramienta que integra estrechamente la simulación de redes y la simulación de tráfico. Además, posee características únicas en comparación con otros simuladores debido a la novedosa metodología de simulación que implementa. Por esta razón resulta interesante desarrollar un conjunto de benchmarks para evaluar el desempeño del simulador y de sus protocolos de enrutamiento en el área de redes vehiculares.

2.3 Objetivos

En esta sección se definen los objetivos que se desean alcanzar a través de la realización del trabajo propuesto.

2.3.1 Objetivo General

Desarrollar un conjunto de benchmarks para evaluar el desempeño del simulador NCTUns en el área de las redes vehiculares.

2.3.2 Objetivos Específicos

- Estudiar el simulador de redes NCTUns para determinar sus ventajas y desventajas en cuanto a su uso en investigaciones de redes vehiculares.
- Desarrollar e implementar una herramienta para traducir trazas de movilidad ns-2 a un formato reconocido por el simulador NCTUns.
- Desarrollar un conjunto de benchmarks en el campo de las redes vehiculares que permita evaluar el desempeño del simulador NCTUns.
- Evaluar el desempeño de los diferentes protocolos de enrutamiento implementados en el simulador NCTUns.
- Desarrollar pruebas de los benchmarks variando parámetros de entrada de las simulaciones.
- Analizar los resultados de las pruebas para evaluar el desempeño de NCTUns con respecto a las métricas elegidas.

2.4 Alcance

Los benchmarks desarrollados permitirán evaluar el desempeño del simulador NCTUns y reportarán resultados como: tiempo real de simulación, consumo de memoria, *One-Way Delay*, Número de Saltos, *Packet Delivery Ratio* y *Normalized Routing Load*. Esto permitirá evaluar el desempeño del simulador NCTUns en el área de simulaciones de redes vehiculares.

3. Redes Vehiculares

Las redesvehiculares o VANETs (Vehicular Ad-Hoc Networks) son redes que permiten la comunicación entre vehículos o entre vehículos y la infraestructura vial para mejorar seguridad[1]. Por ejemplo, si un vehículo sufre un accidente, este puede comunicarse con otros vehículos para alertarlos y que tomen las precauciones necesarias. Otras aplicaciones de las redes vehiculares tienen como objetivo prestar servicios a los usuarios, como por ejemplo proveer acceso al Internet.

Las redes vehiculares poseen ciertas características únicas en comparación con otras redes móviles similares. Por ejemplo, las redes vehiculares son bastantes dinámicas debidoa la alta velocidad que pueden alcanzar los vehículos. A pesar de estar relacionadas con otras tecnologías como las redes MANET (Mobile Ad Hoc Network), las características únicas de las redes vehiculares hacen que sea un reto el desarrollo de protocolos de enrutamiento, la difusión de datos, losmecanismos de seguridad, entre otros.

Un aspecto importante en el estudio de las redes vehiculares es el uso de modelos de movilidad en las simulaciones. Los modelos de movilidad permiten reflejar el comportamiento del tráfico vehicular[1]. Existen modelos de movilidad sencillos que no toman en cuenta aspectos importantes del tráfico real como intersecciones, semáforos y señalizaciones de tráfico.

En los últimos años se ha mostrado un alto interés en las redes vehiculares. Una muestra de ello es el surgimiento de DSRC (Dedicated Short Range Communications)en los Estados Unidos por la FCC (Federal Comunications Comission) en el 2003 y el inicio del C2C-CC (Car2CarCommunicationConsortium) en Europa por los fabricantes de automóviles, con el objetivo principal de aumentar la seguridad vial y la eficiencia del tráfico a través de la comunicación entre vehículos. El IEEE también está tomando parte a través de la familia de estándares para la comunicación inalámbrica entre vehículos IEEE 1609 (WAVE: Wireless Access on Vehicular Enviroments)[2].

3.1 Arquitectura de las Redes Vehiculares

Las redes vehiculares pueden ser desplegadas por los operadores de red y proveedores de servicios en conjunto con entes gubernamentales. Los recientes avances en las tecnologías inalámbricas permiten establecer una serie de arquitecturas de despliegue de las redes vehiculares, teniendo en cuenta todas las áreas posibles, como las carreteras, áreas rurales, y entornos de la ciudad[2]. Estas arquitecturas permiten la comunicación V2V y la comunicación V2I, donde se plantean dos alternativas que incluyen:

- Una red inalámbrica totalmente ad hoc que permita la comunicación entre vehículos de forma independiente, es decir, sin apoyo de la infraestructura.
- Una arquitectura híbrida que no dependa de una infraestructura fija de una forma constante, pero pueda hacer uso de la misma para mejorar el rendimiento y proveer acceso a servicios adicionales cuando estén disponibles.

En el último punto, los vehículos pueden comunicarse con la infraestructura, ya sea de modo *one-hop* o modo *multi-hop* de acuerdo a las posiciones de los vehículos en relación con los RSUs (Road Side Units). La arquitectura básica de una red vehicular debe incluir los siguientes tres ámbitos: (1) el vehículo, (2) la red ad hoc, y (3) la infraestructura. La Figura 3.1(tomada de [3]) muestra las capas de una arquitectura básica de red vehicular,

donde se deben definir claramente las tecnologías en la capa física y MAC que juegan un rol muy importante para la comunicación, así como también considerar los planos de gestión y seguridad, tener en cuenta el enrutamiento y regímenes de comunicación para poder establecer los servicios y aplicaciones.

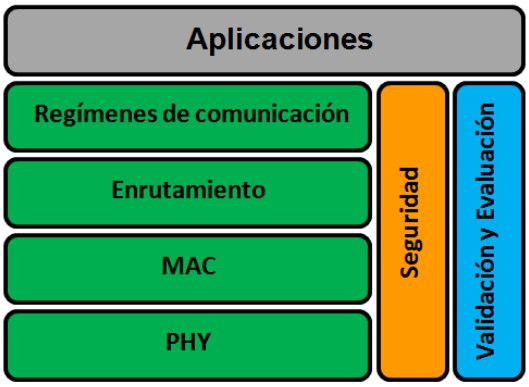


Figura 3.1: Capas de una Arquitectura Básica de Red Vehicular

Para la comunicación *In-Vehicle* cada vehículo tiene establecido un conjunto de ECUs (Electronic Control Units) que se comunican entre sí para lograr establecer ciertas funcionalidades claves, mientras que para la comunicación *Out-Vehicle* el vehículo tiene integrado dos tipos de unidades más[3]: (1) el OBU (On-Board Unit) y (2) uno o más OBCs (On-BoardComputers). Un OBU es un dispositivo integrado en el vehículo que tiene la capacidad de comunicación hacia el exterior mediante la tecnología que se implemente en las capas física y MAC. Mientras que un OBC es un dispositivo que tiene la capacidad de ejecutar una, o un conjunto de aplicaciones, haciendo uso de las capacidades que ofrece el OBU.

La red vehicular está compuesta por los vehículos equipados con OBUs y los RSUs que se ubican en la infraestructura vial de forma fija. Los OBUs de diferentes vehículos forman entre sí redes ad hoc o VANETs, donde cada OBU está equipado con elementos de comunicación, incluyendo al menos un dispositivo de corto y mediano alcance dedicado de comunicación inalámbrica. Un RSU puede estar conectado a una red de infraestructura y a su vez estar conectado al Internet[2]. Los RSUs también pueden comunicarse entre sí directamente o a través de múltiples saltos y su función principal es la mejora de la seguridad vial, mediante la ejecución de aplicaciones, y el envío y la recepción de datos. La Figura 3.2muestra la comunicación que se puede dar entre los OBUs y los RSUs para una red vehicular. Existen dos tipos de dominio de infraestructura[2]: (1) RSU y (2) *hot-spot*. Los RSUs permiten a los OBUs acceder a la infraestructura, tener acceso al Internet y a los servicios que ofrece la infraestructura. Los vehículos también pueden tener acceso al Internet y servicios a través de *hot-spots* públicos, comerciales o privados (*WiFihot-spots*).

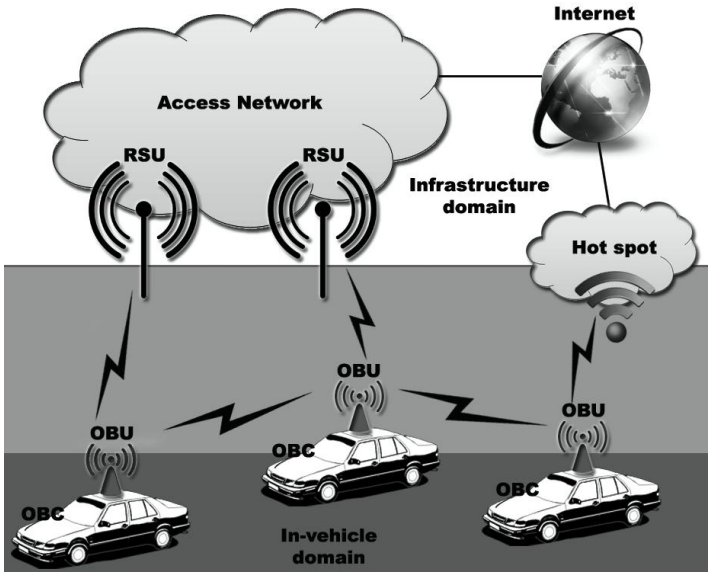


Figura 3.2: Componentes y Tipos de Comunicación en una Red Vehicular

3.2 Características de las Redes Vehiculares

Las redes vehiculares cuentan con características únicas y específicas, las cuales se nombran a continuación:

- Sin problemas de batería: los problemas de alimentación del dispositivo de comunicación no suelen ser un obstáculo en este tipo de redes como ocurre en el caso de redes ad hoc convencionales o redes de sensores, ya que la batería del vehículo puede proporcionar energía continua a los dispositivos de comunicación y de cómputo.
- Mayor capacidad de cálculo: los vehículos pueden proporcionar cómputo más significativo, ya que en forma general, no es un problema acomodar un computador dentro de un vehículo.
- Movilidad previsible: a diferencia de las redes ad hoc tradicionales donde es difícil predecir la movilidad de los nodos, los vehículos tienden a tener movimientos muy predecibles que son en general limitados por la infraestructura vial y los vehículos cercanos. La información de ubicación está disponible gracias a tecnologías como GPS (Global Positioning System), entonces, teniendo en cuenta la velocidad media, la velocidad actual y la trayectoria de la carretera, la posición futura de un vehículo se puede predecir.

Sin embargo, las redes vehiculares tienen que hacer frente a algunas características desafiantes, las cuales son:

- Escalabilidad: las redes vehiculares pueden tener tamaños muy grandes, en particular en horas pico, cuando aumenta el tráfico vial.
- Gran movilidad: el entorno en que operan las redes vehiculares es extremadamente dinámico, donde hay que considerar situaciones drásticas como velocidades de hasta 200 km/h en las autopistas.
- Topología dinámica: por el movimiento de los vehículos, la topología de la red cambia con frecuencia, por lo que se presenta mucho la conexión y desconexión de enlaces de comunicación. De hecho, puede ser que la duración de las conexiones sea muy corta debido al movimiento de los vehículos.

3.3 Aplicaciones

Las aplicaciones usadas en las redes vehiculares pueden ser divididas en dos categorías[4]: (1) aplicaciones de seguridad, que incrementan la seguridad física y (2) aplicaciones de usuario, que proveen servicios, como por ejemplo entretenimiento.

Las aplicaciones de seguridad se enfocan en reducir el número de accidentes viales. Por ejemplo, en caso de un accidente, un vehículo puede enviar una notificación para que esta sea propagada a los vehículos que vienen hacia el accidente para evitar colisiones (Figura 3.3). Estas aplicaciones también pueden ser usadas para prevenir congestiones de tráfico y proveer a los usuarios la mejor ruta disponible para llegar a sus destinos. Las aplicaciones de usuario pueden proveer a los usuarios información de viaje, o intercambio de contenido con otros usuarios.

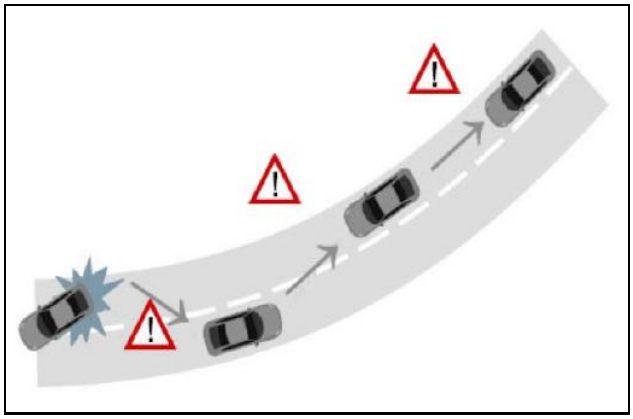


Figura 3.3: Aplicación de Seguridad en Redes Vehiculares

3.4 Estándares para la Comunicación en Redes Vehiculares

Los estándares simplifican el desarrollo de productos, ayudan a reducir costos y permiten a los usuarios comparar productos interoperables. Existen varios estándares que están relacionados con el acceso inalámbrico en ambientes vehiculares. Estos estándares incluyen protocolos que aplican a equipos de comunicación, los protocolos que especifican servicios de enrutamiento, seguridad, etc.

3.4.1 DSRC

DSRC (Dedicated Short Range Communications) [5]es un servicio de comunicación desarrollado para proveer comunicaciones V2V y V2I a corto o mediano alcance. DSRC está dirigido a proporcionar altas transferencias de datos y baja latencia de comunicación en pequeñas zonas de comunicación. En el año 1999, la FCC de los Estados Unidos asignó el rango de frecuencias 5.850GHz – 5.925 GHz para ser usados por DSRC.

El espectro DSRC está dividido en 7 canales, cada uno de 10 MHz. Un canal, el canal CCH (Control Channel), está restringido para comunicaciones de seguridad únicamente, mientras que dos de los otros canales están reservados para propósitos especiales tales como seguridad crítica de la vida. Todos los canales restantes, los SCHs (Service Channels), son canales de servicio los cuales pueden ser usados para aplicaciones sean de seguridad o no. Las aplicaciones de seguridad tienen mayor prioridad para evitar degradaciones en el desempeño y al mismo tiempo salvar vidas advirtiendo a los conductores de peligros inminentes o eventos que permitan tomar acciones correctivas a tiempo.

La capa física especificada por el DSRC se estructura en súper-tramas de duración de 100 ms. Cada intervalo asociado a una súper-trama se divide en dos períodos y se dedica a un aspecto particular de la comunicación[6]. El primero de ellos es el CCH, cuya duración por defecto es de 50 ms y que se encarga del envío de información importante, normalmente relacionada con la seguridad vial. El segundo es el SCH, compuesto por la multiplexación temporal cada 100 ms de canales que operan a distintas frecuencias dentro de la banda asignada. El SCH permite la transmisión de información relacionada con seguridad, entretenimiento, y administración remota a través de paquetes IP.

La diferencia fundamental entre CCH y SCH radica en el hecho de que el primero no puede usar IP para la transmisión de paquetes. Para ello recurre a un protocolo de propósito específico que opera al mismo nivel que IP (Internet Protocol) conocido comoWSMP (WAVE Short Message Protocol). WSMP toma en consideración las características especiales que definen a los entornos de tránsito vehicular y reduce sustancialmente la carga de los paquetes para mejorar las transmisiones. Los OBUs que se ubican en los vehículos soportan la comunicación a través de DSRC y así pueden establecer conexión con otros vehículos a su alrededor. En la Figura 3.4se muestra el espectro DSRC asignado por el FCC para el uso de las redes vehiculares donde se especifican cada uno de los siete canales.

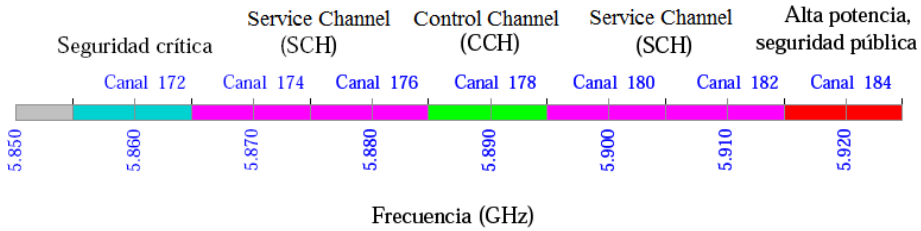


Figura 3.4: Espectro de Frecuencias de DSRC

3.4.2 WAVE

Un entorno vehicular requiere de un conjunto de nuevas exigencias en los sistemas modernos de comunicación inalámbrica. Aplicaciones de comunicación para seguridad vehicular no pueden tolerar largos retrasos para el establecimiento de la conexión antes de comunicarse con otros vehículos encontrados en el camino[2]. El estándar IEEE 802.11p, también conocido como WAVE (Wireless Access in Vehicular Environments), está diseñado para resolver estos problemas. El protocolo WAVE ofrece mejoras en la capa física (PHY) y en la capa MAC de los actuales estándares inalámbricos 802.11.

El Departamento de Transporte de los Estados Unidos promueve y soporta el ITS (Intelligent Transportation Systems) sobre la base de comunicaciones DSRC. El ITS se centra principalmente en permitir que las aplicaciones de seguridad pública puedan salvar vidas y mejorar el flujo de tráfico.

Para comunicaciones V2X (V2V ó V2I), los estándares IEEE 1609.X son desarrollados para suministrar los servicios necesarios en una capa superior dentro de la carga útil de tramas IEEE 802.11p[7].LaFigura 3.5muestra la arquitectura de protocolos de WAVE.

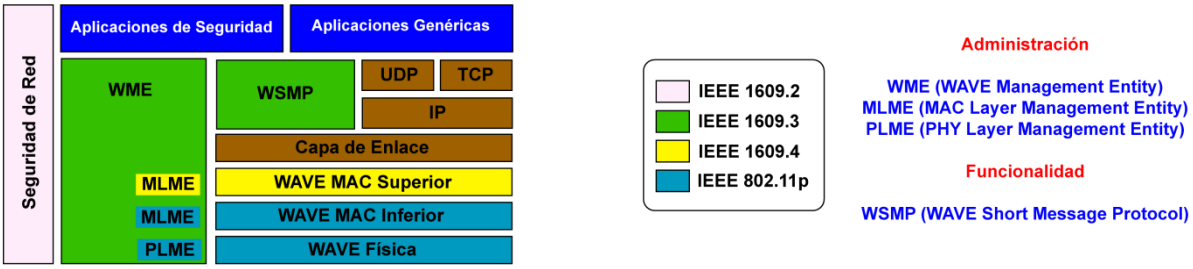


Figura 3.5: Arquitectura de Protocolos de WAVE

Cada uno de estos estándares tiene una función específica en la arquitectura WAVE:

- IEEE 1609.1: define un administrador de recursos que permite la interoperabilidad de las aplicaciones que se ejecuten en las redes vehiculares. Forma parte de la capa de aplicación.
- IEEE 1609.2: establece los servicios de seguridad para las comunicaciones V2X, como la autenticación de las estaciones y el cifrado de mensajes.
- IEEE 1609.3: especifica los servicios de redes para las comunicaciones V2X, con un nuevo protocolo llamado WSMP que permite el intercambio de mensajes WSM (WAVE Short Message).
- IEEE 1609.4: permite la operación de múltiples canales. Tiene una fuerte relación con los mecanismos EDCA (EnhancedDistributedChannelAccess).
- IEEE 802.11p: define operaciones de acceso al medio a nivel inferior de la MAC y establece las adaptaciones del medio físico de comunicación.

WAVE define dos tipos de dispositivos: (1) el dispositivo estacionario RSU y (2) el dispositivo móvil OBU. Estos dispositivos pueden ser tanto proveedores como usuarios de servicios y pueden intercambiar entre tales modos. Normalmente los dispositivos WAVE estacionarios corren una aplicación que provee un servicio, y los dispositivos móviles corren aplicaciones que usan dicho servicio. También puede haber aplicaciones en dispositivos remotos al RSU cuyo propósito es proveer los servicios al OBU.

3.5 Enrutamiento y Difusión

Debido al dinamismo presente en las redes vehiculares, el enrutamiento debe ser eficiente y adaptarse a sus características y las aplicaciones que se ofrecen en ellas[2], permitiendo la transmisión de las tramas con diferente prioridad de acuerdo con el tipo de aplicación (relacionadas con la seguridad o no). Hasta ahora, la mayor parte de la investigación de las redes vehiculares se ha centrado en el análisis de los algoritmos de enrutamiento para manejar el problema de la gran cantidad de mensajes que se deben enviar en una red muy dinámica. En la actualidad, la penetración de la tecnología de redes vehiculares es un tanto débil, y por lo tanto, estas redes deben contar con el apoyo de la infraestructura existente para el despliegue a gran escala. Sin embargo, en el futuro, se espera observar un mayor uso de estas redes y con un soporte de infraestructura menor. Es por ello, que se debe considerar el problema de las desconexiones de enlaces en cada momento por la gran movilidad, que es un reto para la investigación fundamental para el desarrollo de un protocolo de enrutamiento confiable y eficiente. En cuanto a los envíos por broadcast de mensajes, los algoritmos de enrutamiento dependerán del tamaño de la red, así como del tipo de aplicación.

Los protocolos de enrutamiento ad hoc tienen como objetivos de diseño la optimización de la red, la simplicidad, el bajo costo operativo, la robustez, la estabilidad, la convergencia rápida y la flexibilidad. Sin embargo, ya que los nodos móviles sufren de problemas de suministro de energía, la velocidad de procesamiento y memoria, y el bajo costo operativo se vuelve más importante que en las redes fijas convencionales, aunque este problema no se presenta en las redes vehiculares[5]. La alta movilidad presente en la comunicación V2V también otorga gran importancia en la rápida convergencia. Por lo tanto, es imperativo que los protocolos de enrutamiento ad hoc compensen efectivamente los retrasos inherentes a la tecnología subyacente, se adapten a los diferentes grados de movilidad y sean lo suficientemente sólidos para hacer frente a la pérdida potencial de transmisión debido a la deserción. Además, estos protocolos deben enrutar los paquetes con mayor eficacia que los algoritmos tradicionales de red con el fin de compensar los recursos de ancho de banda limitados[2]. Varios algoritmos de enrutamiento para redes ad hoc han surgido para hacer frente a las dificultades relacionadas con el enrutamiento unicast. Estos algoritmos pueden clasificarse ya sean proactivos o reactivos, en función de su mecanismo de descubrimiento de ruta. Es vital el estudio de estos protocolos de enrutamiento y establecer un esfuerzo de desarrollo para incorporar las redes inalámbricas ad hoc en la industria automotriz.

En los protocolos proactivos, cada nodo actualiza continuamente las rutas a todos los demás nodos de la red. En consecuencia, la ruta está disponible de inmediato cuando un nodo necesita enviar un paquete a otro nodo en la red. La principal ventaja de los algoritmos proactivos es que tienen un retardo más corto[2]. Como ejemplos de algoritmos proactivos se pueden mencionar OLSR (Optimized Link State Routing) y TBRPF (Topology Dissemination Based on Reverse-Path Forwarding). La desventaja de los protocolos OLSR y TBRPF es su estrategia de difusión del estado de enlace de enrutamiento. Los cambios de enlace reconocidos causan que los nodos inunden paquetes de control a través de toda la red, los cuales comprometen los recursos de la misma.

Por el contrario, los protocolos de enrutamientoreactivos descubren las rutas por demanda. Por lo tanto, una ruta se descubre cuando un nodo origen necesita comunicarse con un nodo destino para el cual la ruta no ha sido aún establecida. El descubrimiento se basa en las inundaciones[2], donde los nodos emisores difunden un mensaje de solicitud de ruta para todos los vecinos inmediatos y éstos a su vez retransmiten la solicitud de ruta hacia sus vecinos. Cuando la solicitud llega al destino o a un nodo que tiene una ruta válida para el destino, un mensaje de respuesta de ruta se genera y se transmite al origen. Por lo tanto, tan pronto como el origen reciba la respuesta de ruta, se crea una ruta desde el origen al destino y viceversa. La ventaja de los algoritmos reactivos es que no hay mensajes de control de las rutas no activas. El mayor inconveniente es la latencia en establecer las rutas de transmisión. Ejemplos de algoritmos reactivos incluyen AODV (Ad hoc On-Demand Distance Vector Routing) y DSR (Dynamic Source Routing).

A pesar que los algoritmos mencionados fueron desarrollados para redes ad hoc, no se adaptan correctamente a las redes vehiculares, debido al alto dinamismo de los vehículos al transitar por las carreteras. Por esta razón, el enrutamiento es un problema abierto para las redes vehiculares y algunos investigadores han propuesto algoritmos como AGF (Advanced Greedy Forwarding) y PGB (Prefered Group Broadcasting).

3.6 Seguridad

La seguridad es de gran importancia en las redes vehiculares ya que problemas de seguridad en dichas redes implicarían la posibilidad de pérdida de vidas humanas. La autenticidad es de gran importancia para las redes vehiculares, ya que por ejemplo, un atacante pudiera reportar accidentes falsos, lo cual podría activar los frenos de un vehículo posiblemente causando un accidente real[4]. Aunque por un lado la autenticación es una propiedad obligatoria en las redes vehiculares, por el otro la privacidad también es una propiedad deseable. La privacidad también es de importancia, ya que se pudiera conocer el recorrido de un vehículo. Otra propiedad importante que debe ser tomada en cuenta es la disponibilidad.Las aplicaciones relacionadas con la seguridad física de las personas, como por ejemplo las aplicaciones para evitar colisiones, requieren un tiempo de retardo mínimo. Si un atacante afecta la disponibilidad, estaría poniendo en riesgo la seguridad de los usuarios.

Las características únicas de las redes vehiculares (descritas en la Sección 3.2) dificultan el campo de la seguridad. El aspecto más notable con respecto a la seguridad en redes vehiculares es lo contradictorio que pueden ser las propiedades como por ejemplo privacidad contra anonimato y autenticación contra la no repudiación.

3.7 Modelos de Movilidad

Los modelos de movilidad están diseñados para describir los patrones de movimiento de los nodos móviles como su ubicación, velocidad y aceleración cambian a través del tiempo. Los modelos de movilidad se clasifican en dos tipos: (1) los modelos de movilidad macroscópicos y (2) los modelos de movilidad microscópicos [4].Los modelos macroscópicos toman en consideración restricciones de movimiento como calles, caminos, intersecciones y semáforos durante la generación de las trazas. Definen la generación de tráfico vehicular como el flujo, la densidad y la distribución inicial de los vehículos. Los modelos microscópicos se enfocan en el movimiento de cada vehículos individualmente y en el comportamiento del vehículo con respecto a los otros vehículos.

Uno de los primeros modelos de movilidad desarrollados fue el modelo RWP (*Random WayPoint*). En este modelo los nodos escogen un destino de manera aleatoria y se mueven hacia dicho destino a una velocidad uniforme. Cuando el nodo alcanza el destino, se escoge un nuevo destino y así sucesivamente. El modelo RWP se utiliza con frecuencia en simulaciones ad hoc.

Existen herramientas para la generación de trazas de movimiento vehicular que se basan en los modelos mencionados anteriormente. Estas herramientas son de gran utilidad para las investigaciones de redes vehiculares, especialmente cuando los estudios se basan en la simulación. En el Capítulo 4 se presentan algunas de estas herramientas y su aplicación en la simulación de redes vehiculares.

4. Simulación de Redes Vehiculares

En este capítulo se presenta un estudio acerca de la simulación de redes vehiculares y los tipos de simuladores utilizados para simulaciones de redes vehiculares. Adicionalmente, se describen las características más resaltantes de algunos simuladores de redes, simuladores de tráfico y simuladores integrados usados en el área de simulación de redes vehiculares.

4.1 Introducción

La simulación de redes vehiculares requiere tomar en cuenta dos aspectos importantes: la comunicación entre nodos y el movimiento de los nodos. Los simuladores de redes han sido diseñados para ser usados en estudios en el área de redes de comunicación. Naturalmente pueden ser utilizados en simulaciones de redes vehiculares para tomar en cuenta el aspecto de la comunicación. Los simuladores de tráfico, también conocidos como generadores de tráfico, son diseñados para realizar estudios en el área de tráfico vehicular. Estos simuladores generan trazas de movimiento basadas en modelos de movilidad de vehículos en una red de carreteras. Los simuladores de tráfico son de importancia para la simulación de redes vehiculares ya que proveen a la simulación de trazas realistas de movilidad.

Un enfoque en la simulación de redes vehiculares consiste en generar trazas de movimiento utilizando un simulador de tráfico y luego exportar estas trazas a un simulador de redes para ser utilizadas por los nodos móviles que representan los vehículos. Otro enfoque consiste en el uso de herramientas que integran un simulador de red con un simulador de tráfico. Estas herramientas se encargan de la comunicación en tiempo real entre el simulador de tráfico y el simulador de redes. También existen simuladores que contienen componentes tanto para la simulación de tráfico como para la simulación de redes.

4.2 Simuladores de Redes

Los simuladores de redes son herramientas utilizadas en estudios de comunicación en redes. Permiten estudiar el comportamiento de dispositivos de red y de protocolos de red. En las simulaciones de redes vehiculares estos simuladores se encargan de controlar la comunicación entre los vehículos y la comunicación entre vehículos y otros dispositivos.

4.2.1 ns-2

ns-2[1] (Network Simulator 2) [8]es un simulador de código abierto desarrollado en C++ que corre en la mayoría de los sistemas operativos incluyendo Unix, MacOS y Windows (a través del uso de Cygwin). ns-2 provee una interfaz de simulación a través del lenguaje OTcl, una variante orientada a objetos del lenguaje Tcl (Tool Command Language). Los usuarios utilizan el lenguaje OTcl para describir la red de interconexión y el tráfico de datos asociado, mientras que el mecanismo interno de los objetos de simulación está definido en C++.

LaFigura 4.1, tomada de[8], muestra la arquitectura básica de ns-2. ns-2 provee a los usuarios el comando ns, el cual toma como argumento el nombre de un script de simulación Tcl con la descripción de la simulación. Luego de realizar la simulación, ns-2 arroja resultados en texto o para interpretar estos resultados gráficamente o

[1] http://www.isi.edu/nsnam/ns

interactivamente a través del uso de herramientas como NAM para ver las animaciones de la transferencia de paquetes y XGraph para crear representaciones gráficas de los resultados obtenidos.

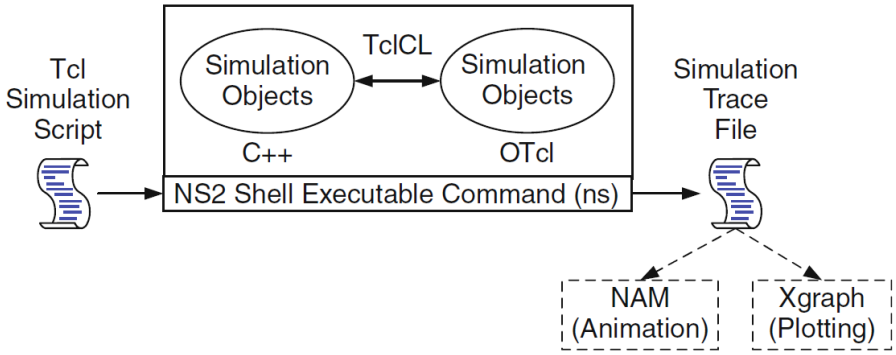


Figura 4.1: Arquitectura de ns-2

4.2.2 ns-3

ns-3[2] (Network Simulator 3) [9]es un simulador de redes de código abierto basado en la metodología de simulación por eventos discretos. El simulador ns-3 surgió a partir de la idea de rediseñar ns-2 (Sección 4.2.1) aunque luego se encontró que no valía la pena mantener la compatibilidad hacia atrás con ns-2 debido a que la mayoría de los modelos más útiles estaban implementados en bifurcaciones o partes separadas que eran generalmente incompatibles unas con las otras. Por este motivo, se decidió que el nuevo simulador iba a ser escrito desde cero, usando el lenguaje de programación C++.

ns-3 permite a los investigadores estudiar los protocolos de Internet y sistemas a gran escala en un ambiente controlado. La meta del proyecto ns-3 es desarrollar un entorno de simulación abierto, el cual se convierta en el medio preferido para realizar la investigación y estudio de las redes[9].

El proyecto ns-3 está comprometido a construir una base sólida de simulación que esté bien documentada, que sea fácil de usar y depurar, además, que responda a la necesidad de cumplir con el flujo de trabajo de la simulación completa. Esto lo realiza partiendo desde la configuración de la simulación a una colección de registros obtenidos y su análisis.

La infraestructura de ns-3 permite el estudio y desarrollo de modelos de simulación de alto desempeño tanto en redes basadas en IP, como en las no IP. Sin embargo, la gran mayoría de los usuarios han tendido a centrase en simulaciones de escenarios IP inalámbricos, los cuales involucran modelos como WiFi, WiMAX, o LTE (Long Term Evolution) para capas 1 y 2 del modelo OSI y una variedad de protocolos de enrutamiento dinámicos como OLSR y AODV.

4.2.3 JiST/SWANS

JiST (Java in Simulation Time) [10]es un motor de alto desempeño para simulaciones de eventos discretos el cual se ejecuta sobre una máquina virtual de Java estándar. Permite a los programadores simular diferentes tipos de escenarios generales de manera eficiente

[2] http://www.nsnam.org

y transparente. Es una plataforma muy eficiente y altamente optimizada en cuanto al consumo de memoria y tiempo de ejecución de la simulación.

SWANS (Scalable Wireless Adhoc Network Simulator) [11], basado en la plataforma JiST, fue creado debido a que las herramientas para la simulación de redes de la época no cubrían las necesidades de los investigadores. SWANS está organizado como un componente de software independiente que permite formar redes inalámbricas o armar redes de sensores, permitiendo simular redes con un gran número de nodos. Busca aprovecharse del diseño de JiST para lograr simulaciones de alto rendimiento, ahorrar memoria y ejecutar aplicaciones de red estándar de Java sobre las redes simuladas.

4.2.4 OMNeT++

OMNeT++[3](Objective Modular Network Testbed in C++) [12] es un entorno de simulación de código abierto basado en la metodología de simulación por eventos discretos. En lugar de ser un simulador especializado en algún área de aplicación en específico, fue diseñado para ser lo más general posible. Es un framework el cual, en lugar de proveer directamente los componentes para simular redes de computadores o de otros dominios, proporciona el entorno y las herramientas básicas para escribir tales simulaciones. Por ejemplo, el soporte para los modelos de movilidad y redes es provisto por el Mobility Framework y el INET Framework, respectivamente. Estos son desarrollados de manera independiente a OMNeT++ y tienen sus propios ciclos de lanzamiento.

El kernel y las bibliotecas de OMNeT++ están escritas en C++. Las bibliotecas de simulación proveen mecanismos para manejar mensajes, soportar la configuración y ensamblado de módulos, generar números aleatorios con varias distribuciones, crear colas, recolectar estadísticas, entre otros.

Las simulaciones en OMNeT++pueden ser ejecutadas bajo varias interfaces de usuario. La interfaz gráfica de usuario para animaciones es altamente útil para realizar demostraciones y depuración, mientras que la interfaz de línea de comando es la mejor opción para ejecuciones en lote. El simulador, así como sus interfaces de usuario y herramientas, son muy portables y han sido adoptados para los sistemas operativos más comunes (Unix, Windows y MacOS).

OMNeT++soporta simulaciones distribuidas y paralelas. Puede usar varios mecanismos para la comunicación entre las partes de una simulación distribuida y paralela como MPI (MessagePassing Interface) o los denominados *pipes*. Los algoritmos de simulación en paralelo pueden ser extendidos o incluso nuevos algoritmos pueden ser añadidos fácilmente. Los modelos no necesitan ninguna instrumentación especial para ejecutarse en paralelo, sólo es cuestión de configuración. OMNeT++ puede ser utilizado para hacer presentaciones de algoritmos en paralelo en salones de clase, ya que la simulación puede ser ejecutada en paralelo incluso bajo el GUI (Graphical User Interface), obteniendo un *feedback* detallado acerca de lo que está ocurriendo.

4.3 Simuladores de Tráfico

En las simulaciones de redes vehiculares, estos simuladores son utilizados para generar las trazas de movimiento de los vehículos. En las redes vehiculares el movimiento de los vehículos afecta la comunicación y la comunicación entre los vehículos afecta el

[3] http://www.omnetpp.org

comportamiento de estos. Estos simuladores son de importancia ya que aportan realismo a la simulación al implementar modelos de movilidad vehicular realistas que han sido estudiados y mejorados a lo largo del tiempo.

4.3.1 SUMO

SUMO[4] (Simulation of UrbanMObility)[13]es una plataforma de simulación de tráfico de nivel microscópico, multimodal, y código abierto que emula el flujo del tráfico de forma continua en el espacio y discreta en el tiempo. Los modelos microscópicos son aquellos que simulan el movimiento de cada vehículo sobre las vías, donde su desplazamiento es determinado tanto por las capacidades físicas del vehículo para moverse como por el comportamiento del conductor para controlarlo. Gracias a esto es posible simular una demanda de tráfico que consista de un conjunto de vehículos individuales, con sus propias rutas y que se mueven a través de una determinada red de carreteras. Es posible que cada conductor intente utilizar el camino más corto a través de la red, pero cuando esto sucede, alguna de las carreteras (sobre todo las avenidas principales) se tienden a congestionar, lo cual reduce el beneficio de su uso. La solución a este problema en ingeniería de tráfico se conoce como *userassignment*. SUMO utiliza DUA (Dynamic UserAssignment) para tratar con escenarios de este estilo[14].

SUMO no es solo un simulador de tráfico, es más bien un conjunto de aplicaciones que ayudan a preparar y realizar la simulación de tráfico. Las redes de carreteras en SUMO pueden ser generadas por una aplicación llamada *netgen* o generadas importando un mapa de carreteras. La aplicación *netconvert* permite leer la red de carretera de otros simuladores de tráfico y también permite importar formatos comunes como *Shapefile* u *OpenStreetMap*.

4.3.2 MOVE

MOVE (MObility model generator for VEhicular networks) [15]es una herramienta desarrollada en Java para la generación con rapidez de modelos de movilidad realistas para simulaciones de redes vehiculares. MOVE está desarrollado sobre el simulador SUMO (descrito en la Sección 4.3.1). La salida de esta herramienta es un modelo de movilidad realista que puede ser utilizado inmediatamente por el simulado ns-2. El GUI de MOVE (Figura 4.2) está desarrollado para que los usuarios puedan rápidamente generar escenarios de simulación realistas sin tener que escribir configuraciones en scripts y sin tener que aprender los detalles del simulador.

[4] http://sumo.sourceforge.net

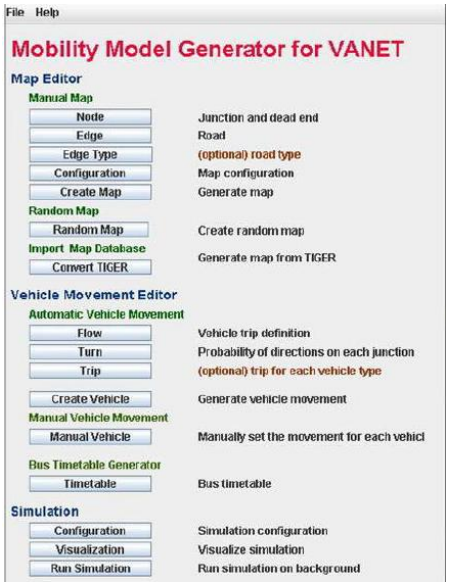


Figura 4.2: Interfaz Gráfica de Usuario de MOVE

MOVE [15] está compuesto por un editor de mapas y un editor de movimiento vehicular. El editor de mapas crea la red de carreteras. El editor de movimiento vehicular permite al usuario especificar la ruta que tomará cada vehículo. Existen tres maneras de definir los patrones de movimiento de los vehículos: (1) se pueden crear manualmente, (2) se pueden generar automáticamente y (3) se pueden especificar basándose en una tabla de tiempos de autobús para simular el transporte público.

4.3.3 CityMob

CityMob[5][16] es un generador de patrones de movilidad para VANETs compatible con ns-2 (Sección 4.2.1), desarrollado por la Universidad Politécnica de Valencia. Permite crear escenarios urbanos con facilidad, implementando tres modelos de movilidad. Los caminos se crean para ambos sentidos, con canales en ambas direcciones y los nodos se mueven a velocidades aleatorias dentro de un rango definido por el usuario. A continuación se describen los tres modelos de movilidad implementados en CityMob:

- Modelo Simple: Patrones de movilidad verticales y horizontales sin cambio de dirección o semáforos.
- Modelo Manhattan: Los nodos se mueven de manera horizontal o vertical en un mapa de tipo Manhattan. Todas las calles son en ambos sentidos con un carril en cada dirección. La dirección de los nodos se selecciona aleatoriamente y no se repiten dos movimientos consecutivos.
- Modelo Downtown: Este modelo se asemeja al modelo Manhattan con la diferencia de que la densidad del tráfico no se distribuye uniformemente. En este modelo hay áreas con una densidad mayor (*downtown*). Hay un parámetro (llamado p) que se puede utilizar para establecer la probabilidad de que la posición inicial de un nodo sea dentro del *downtown* y la probabilidad de que los nodos que están fuera del *downtown* entren a dicha área.

En CityMob la distancia entre las calles es configurable. Está limitada por el tamaño del mapa y debe haber un número mínimo de intercepciones para permitir a los nodos

[5]http://www.grc.upv.es/Software/citymob.html

cambiar de dirección. Los usuarios pueden configurar el número de nodos simulados y el número de nodos que están dañados. Cada nodo tomará una posición inicial aleatoria, aunque en el modelo *downtown* la probabilidad de iniciar dentro del área *downtown* es mayor debido a que existe mayor densidad. La velocidad puede variar según el tamaño del mapa. Los nodos se moverán con una velocidad por debajo de la velocidad máxima definida por el usuario.

4.3.4 VanetMobiSim

VanetMobiSim (Vehicular Ad Hoc Networks Mobility Simulator) [17]es un conjunto de extensiones de CanuMobiSim[6], un framework usado para la simulación genérica de movilidad por el CANU[7] (Communication in Ad Hoc Networks for Ubiquitous Computing) en la Universidad de Stuttgart, Alemania. CanuMobiSim es una plataforma independiente basada en Java.Provee una arquitectura de movilidad eficiente y fácilmente extensible. El framework incluye modelos de movilidad, convertidor de datos geográficos en varios formatos y un entorno gráfico. Genera trazas de movilidad para los siguientes simuladores de red: ns-2, GloMoSim[8] (Global Mobile Information System Simulation Library), y QualNet[9].

Al ser de propósito general, CanuMobiSim no cuenta con un alto nivel de detalle que permita representar escenarios específicos como la simulación de ambientes vehiculares[18]. Es por ello que VanetMobiSim extiende a CanuMobiSim para soportar la movilidad vehicular con un alto grado de realismo. Estas extensiones consisten principalmente de un modelo de topología vial usando estructuras de datos compatibles con GDF (Geographic Data Files) [17]y un conjunto de modelos de movilidad orientados al ambiente vehicular. El modelo topológico está compuesto de elementos espaciales, sus atributos y las relaciones que unen esos elementos espaciales para describir las áreas vehiculares, es decir, las calles y carreteras.

4.4 Simuladores Integrados

Los simuladores integrados son simuladores que se encargan de ambos aspectos de la simulación de redes vehiculares: la simulación del movimiento de los vehículos y la simulación de la comunicación entre los vehículos. Algunas aplicaciones de redes vehiculares, como las aplicaciones de seguridad, requieren la interacción en tiempo real entre el movimiento de los vehículos y la comunicación de estos. Los simuladores integrados permiten la comunicación en tiempo real de ambos componentes, por lo que son más adecuados para el estudio de este tipo de aplicaciones.

4.4.1 TraNS

TraNS[10] (Traffic and Network Simulation Enviroment) [19] es una herramienta de código abierto para la simulación de redes VANET que integra dos simuladores de código abierto: el simulador de redes ns-2 (descrito en la Sección 4.2.1) y el simulador de tráfico SUMO (descrito en la Sección 4.3.1).TraNS permite que el simulador de redes utilice modelos de movilidad realistas provistos por el simulador de tráfico y además permite que

[6] http://vanet.eurecom.fr
[7] http://xurl.es/i630f
[8] http://pcl.cs.ucla.edu/projects/glomosim
[9] http://www.scalable-networks.com
[10] http://lca.epfl.ch/projects/trans

el simulador de redes influya en el comportamiento del simulador de tráfico basándose en la comunicación entre los vehículos.

TraNS tiene dos modos de operación: el modo *network-centric* y el modo *application-centric*. El modo *network-centric* (Figura 4.3) es usado para evaluar protocolos de comunicación VANET que no influyen en la movilidad de los nodos, como por ejemplo, intercambio de contenido de usuario como música o información de viaje. En este modo las trazas de movimiento son generadas por el simulador de tráfico antes de iniciar la simulación y son adaptadas a un formato entendible por el simulador de redes por medio de un *parser*. El modo *application-centric* (Figura 4.4) es usado para evaluar aplicaciones VANET que si influyen en la movilidad de los nodos, como por ejemplo, aplicaciones de seguridad para evitar colisiones entre vehículos. En este modo, ambos simuladores deben correr simultáneamente y la comunicación entre ambos se lleva a cabo a través de una interfaz conocida como TraCI.

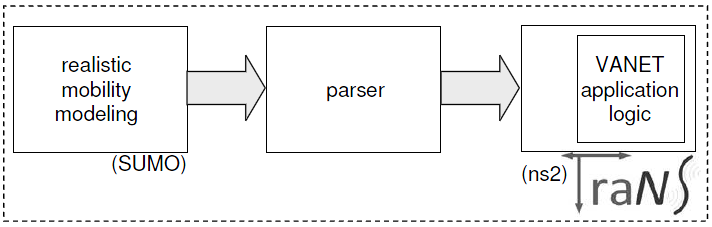


Figura 4.3: Arquitectura de TraNS (Modo Network-Centric)

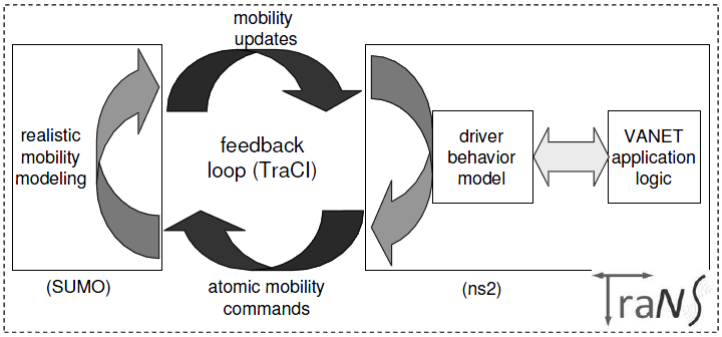


Figura 4.4: Arquitectura de TraNS (Modo Application-Centric)

En ambos modos de operación, la comunicación entre los simuladores se realiza a través de una conexión TCP/IP dedicada de tal manera que la simulación se puede realizar usando dos hosts distintos siempre y cuando TraNS esté instalado en ambos hosts.

4.4.2 GrooveNet

GrooveNet[11][20] es un simulador híbrido que permite la comunicación entre vehículos simulados y vehículos reales. Para simular movilidad en caminos reales, GrooveNet permite cargar mapas reales de los Estados Unidos importando archivos TIGER/Line disponibles al público de manera gratuita por el US Census Bureau.

GrooveNet soporta varias interfaces de red para comunicación V2V y V2I como: una interfaz DSRC 5.9GHz, interfaces IEEE 802.11a/b/g e interfaces celulares EVDO

[11]http://www.seas.upenn.edu/rahulm/Research/GrooveNet

(Enhanced Voice-Data Only). La comunicación se puede establecer por TCP o UDP. Los vehículos reales se comunican entre sí por las interfaces DSRC o 802.11.

GrooveNet soporta varios modelos de movilidad:(1) el modelo *car-following* en el cual los vehículos no exceden la velocidad del vehículo que está al frente, (2) El modelo *Street Speed*, en el cual los vehículos se mueven con una velocidad dentro de un rango del límite de velocidad para el camino y (3) el modelo *Fixed Mobility*, en el cual los vehículos no se mueven.Este último modelo es usado para las simulaciones que no requieren movimiento dinámico de los nodos.

GrooveNet provee varios modos de operación tales como: *drive mode*, *simulation mode, playback mode, hybrid simulation mode* y *test generation mode*. Adicionalmente, el simulador tiene la capacidad de comunicarse con los vehículos reales para obtener información extra, ejecutar un escenario específico, mezclar entre dos modos, reproducir los archivos *logs* generados durante cualquiera de los modos de operación. La habilidad única de GrooveNet de integrar los vehículos simulados con los vehículos reales le permite que funcione tanto como un banco de pruebas como un simulador.

4.4.3 MobiREAL

MobiREAL[12][21] es un simulador desarrollado para la plataforma Windows que provee una nueva metodología para modelar y simular el movimiento de nodos y para la evaluación de aplicaciones MANET. Está compuesto en dos partes: (1) el simulador de comportamiento, el cual simula el comportamiento de los nodos móviles y (2) el simulador de redes, el cual simula el intercambio de datos entre los nodos móviles. El simulador de comportamiento y el simulador de redes son dos programas independientes que se intercambian los datos necesarios a través de un canal TCP.

MobiREAL adopta un modelo probabilístico para describir el comportamiento de los nodos inalámbricos. El modelo propuesto permite describir como los nodos pueden cambiar sus destinos, rutas y velocidades basados en su posición, sus alrededores e información obtenida de aplicaciones. MobiREAL simula MANETs usando el soporte de movilidad del GTNetS [13] (Georgia Tech Network Simulator). El animador permite visualizar dinámicamente el movimiento de los nodos, estados de conectividad y transmisión de paquetes.

4.4.4 NCTUns

NCTUns [14] (NationalChiaoTungUniversity Network Simulator) [22]es un simulador y emulador extensible de alta fidelidad capaz de simular varios protocolos de redes cableadas e inalámbricas. Está basado en una novedosa metodología de simulación, por lo que provee ventajas únicas en comparación con otros simuladores. Puede ser usado como un emulador ya que integra la simulación y la emulación permitiendo así que los dispositivos reales interactúen con dispositivos simulados. NCTUns usa la pila de protocolo TCP/IP para generar resultados de alta fidelidad. Puede correr aplicaciones reales de Unix en nodos simulados sin necesitar modificaciones.

[12] http://www.mobireal.net
[13]http://www.ece.gatech.edu/research/labs/MANIACS/GTNetS
[14]http://nsl.csie.nctu.edu.tw/nctuns.html

Entre las redes soportadas por NCTUns se encuentran: redes Ethernet, redes inalámbricas 802.11a/b, 802.16 WiMAX, 802.11p/1609 WAVE, etc. La arquitectura de NCTUns permite el soporte de simulaciones en paralelo.

NCTUns provee una interfaz gráfica que permite al usuario editar de manera visual todos los parámetros necesarios para desarrollar los casos de simulación, crear la topología y visualizar los resultados por medio de gráficos o animaciones.

4.5 Motivo de Elección del Simulador NCTUns para el Estudio

Luego de estudiar la arquitectura y la metodología de simulación de varios simuladores, se decidió realizar los estudios de análisis de desempeño sobre el simulador NCTUns debido a las características únicas que este posee con respecto a los otros simuladores a raíz de la novedosa tecnología de simulación utilizada en su implementación. Otro motivo para la elección del simulador NCTUns es lo fuertemente acoplados que están los componentes de simulación de tráfico y de simulación de redes, facilitando así el estudio de redes vehiculares.

5. NCTUns

NCTUns [23] es un simulador y emulador de redes que corre bajo el sistema operativo Linux. Permite la simulación de varios dispositivos y protocolos usados en redes cableadas o inalámbricas. Su diseño está basado en la combinación dedosmetodologías: la metodología de simulación por eventos discretos y una metodología de simulación novedosa conocida como “kernel re-entering”, inventada por el autor del simulador, el profesor S.Y. Wang mientras realizaba sus estudios de doctorado en la Universidad de Harvard.

El uso de la nueva metodología de simulaciónpermite queNCTUns tenga algunas ventajassobre otros simuladores similares. Entre estas ventajas está la posibilidad de que los nodos simulados ejecuten aplicaciones reales, es decir, aplicaciones desarrolladas para ser utilizadasen computadores de la vida real y no aplicaciones desarrolladas para ser utilizadas en un ambiente de simulación a través de un API (Application Programming Interface),como suele ser el caso de otros simuladores. Otra ventaja que ofrece el uso de esta herramienta es la fidelidad de las simulaciones que se realizan, NCTUns utiliza la misma pila de protocolos de Linux para llevar a cabo las simulaciones.

NCTUns [23]provee una plataforma basada en módulos. Los módulos son implementaciones de las capas de la pila de protocolos, como por ejemplo el módulo MAC de Ethernet. Los desarrolladores de protocolos pueden probar sus protocolos en NCTUns desarrollando nuevos módulos o modificando los ya existentes. Al desarrollar y combinar módulos de interés, se pueden crear dispositivos especiales con comportamientos que se adapten a la necesidad de los desarrolladores.

NCTUns [24] combina las capacidades provistas por un simulador de redes y las provistas por un simulador de tráfico, permitiendo que esta herramienta sea utilizada para el diseño de protocolos para redes de comunicación de sistemas ITS (Intelligent Transportation Systems).La plataforma de simulación de NCTUns es altamente integrada, soporta completamente las interacciones entre una red de carreteras y una red de comunicaciones sin necesidad de uso de ningún tipo de software que realice la conexión entre la simulación de redes y la simulación de tráfico.

La arquitectura distribuida de NCTUns permite que las simulaciones sean ejecutadas de manera remota y concurrente. Los componentes del simulador se comunican a través del protocolo TCP/IP.Varias máquinas de simulación pueden prestar sus servicios a uno o más programas de usuarios de NCTUns bajo esta arquitectura.

NCTUns puede ser utilizado como emulador, permitiendo el intercambio de paquetes entre nodos simulados y elementos reales, como routers o computadores, conectados físicamente a la máquina de simulación. NCTUns [25]también permite un enfoque de emulación distribuido en el cual las redes emuladas son divididas en partes más pequeñas para que cada parte sea emulada en una máquina diferente, aprovechando la capacidad de cada máquina al máximo.

5.1 Antecedentes

El predecesor de NCTUns [23] es el Harvard Network Simulator[15], desarrollado por el profesor S.Y. Wang en el año 1999. Eventualmente se descubre que este simulador tiene varias limitaciones e inconvenientes que necesitaban ser superados. Por esta razón, luego de unirse a la NCTU (National Chiao Tung University), el profesor S.Y. Wang, junto a un grupo de estudiantes,desarrollan el simulador NCTUns (National Chiao Tung University Network Simulator). Para superar las limitaciones que existían en su predecesor, el profesor S.Y. Wang inventó una manera de combinar la metodología de simulación por eventos discretos y la metodología kernel re-entering.

Inicialmente, NCTUns fue desarrollado para correr sobre el sistema operativo FreeBSD, pero, puesto que el sistema operativo Linux fue ganandomás popularidad, los desarrolladores de NCTUns adoptaron ésta última plataforma, abandonando FreeBSD. Aunque oficialmente NCTUns ha sido desarrollado para la distribución Fedora, algunos usuarios avanzados de Linux han logrado llevarlo a otras distribuciones como Debian o Ubuntu exitosamente. Esto gracias al hecho de que todas las distribuciones de Linux utilizan el mismo kernel y sólo varían en configuraciones y aplicaciones por defecto.

NCTUns 1.0, la primera versión del simulador, fue lanzada a la comunidad el primero de noviembre del 2002 como un programa de código abierto para la simulación de redes. A partir de la versión 4.0 (lanzada el 25 de julio del 2007)[24], NCTUns comenzó a combinar sus capacidades de simulación de redes con algunas capacidades de simulación de tráfico como construcción de redes de carreteras y conducción automática de vehículos. La versión 5.0 (lanzada en el año 2008) [22],además de otras capacidades, incluye el soporte para el estándar IEEE 802.11p.

La última versión de código abierto de NCTUns es la versión 6.0. En el año 2011, debido al éxito que fue acumulando el simulador desde su primer lanzamiento, el profesor Wang decide comercializar NCTUns y funda la empresa Estinet[16], la cual ofrece la versión comercial del simulador NCTUns bajo el nombre Estinet 8.0.

5.2 Metodología de Simulación

NCTUns [26]está basado en la metodología de simulación kernel re-entering. Las interfaces de tipo túnel disponibles en el sistema operativo Linux son un elemento clave en la metodología kernel re-entering. Una interfaz de tipo túnel es una pseudo interfaz de red que no tiene ninguna red física conectada a ella. Las funciones de una interfaz de tipo túnel son las mismas que las funciones de una interfaz Ethernet.

El reto para los desarrolladores ha sido poder combinar la metodología kernel re-entering con la metodología de simulación por eventos discretos. NCTUns realiza modificaciones al kernel de Linux para que provea servicios al motor de simulación (descrito en la Sección 5.3.2) y para que los objetos simulados puedan comunicarse con el motor de simulación. Los objetos simulados en NCTUns [27]no están contenidos en un solo programa, están distribuidos en programas independientes que corren concurrentemente en una máquina Unix, como generadores de tráfico, el motor de simulación, el kernel de Unix, etc.

[15] http://www.eecs.harvard.edu/networking/simulator.html
[16] http://www.estinet.com

En laFigura 5.1, tomada de[23], se puede apreciar el principio de la tecnología *kernel re-entering*. En una simulación, cuando un emisor TCP envía un paquete, este paquete es enviado al kernel y pasa a través de la pila de protocolos del kernel como lo haría cualquier paquete Ethernet. Puesto que la interfaz tipo túnel 1 ha sido configurada como el dispositivo de salida del paquete, el paquete será insertado en la cola de salida del túnel 1. El motor de simulación detectará el evento inmediatamente y emitirá una llamada *read* al sistema para obtener el paquete.

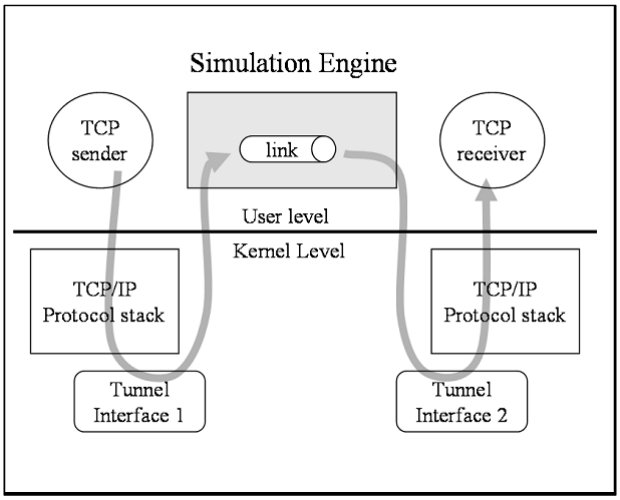


Figura 5.1: Metodología de Simulación “kernel re-entering”

Luego de simular el tiempo de propagación y el tiempo de transmisión, el motor de simulación pondrá el paquete en la cola de entrada de la interfaz tipo túnel 2 a través de una llamada *writ*e al sistema. El paquete pasará nuevamente a través de la pila de protocolos TCP/IP. Luego el paquete será colocado en la cola de recepción del socket creado por el receptor. Finalmente el receptor usará una llamada al sistema para sacar el paquete del kernel.

El ejemplo de la Figura 5.1permite apreciar como el paquete enviado por el emisor pasa a través del kernel dos veces. Es por esto que la metodología fue llamada *kernel re-entering methodology* o metodología de reentrada al kernel. Entrando al kernel varias veces es como se crea la ilusión de que un paquete pasa a través de varios nodos, cuando en realidad el paquete siempre permanece en la misma máquina y pasa varias veces a través de la misma pila de protocolos.

5.3 Arquitectura

Para soportar simulaciones remotas y concurrentes, NCTUns utiliza una arquitectura distribuida en la cual los distintos componentes se pueden dividir de acuerdo a su funcionalidad. NCTUns también utiliza una arquitectura de sistema abierto, permitiendo a los usuarios fácilmente agregar y/o modificar módulos de protocolos al simulador.

5.3.1 Interfaz Gráfica de Usuario

La interfaz gráfica de usuario o GUIprovista por el simulador [28]permite al usuario editar de manera visual todos los parámetros necesarios para desarrollar los casos de simulación. A través de la interfaz gráfica los usuarios pueden:

- Dibujar la topología de la red.
- Configurar los protocolos a ser utilizados por los nodos.
- Trazar gráficos del rendimiento de la red.
- Reproducir la animación de una traza de transferencia de paquetes.
- Especificar las ubicaciones iníciales y el recorrido de los nodos móviles.
- Dibujar la red de carreteras en el caso de simulaciones de redes vehiculares.
- Importar una red de carreteras existente por medio de archivos S*hapefile*.
- Otras funcionalidades.

La GUI de NCTUns (Figura 5.2)[28]utiliza sockets TCP/IP para comunicarse con los otros componentes. Por ejemplo, la GUI puede entregar un trabajo de simulación a una máquina de simulación remota para que ejecute dicha simulación y luego los archivos generados sean devueltos al programa GUI y mostrados al usuario. Finalmente, el usuario podrá trazar curvas de desempeño, examinar los datos registrados o reproducir la animación de la transferencia de paquetes.

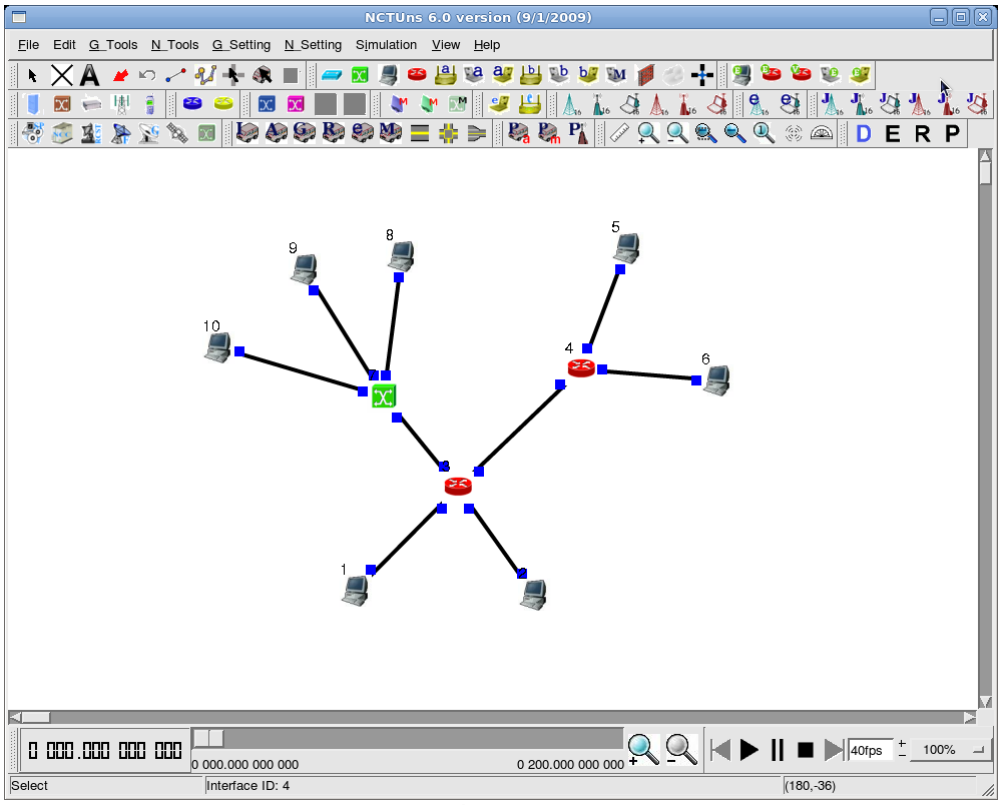


Figura 5.2: Interfaz Gráfica de Usuario de NCTUns

Mientras una simulación está siendo ejecutada en una máquina remota, el usuario puede consultar o establecer los valores de un objeto de la simulación en cualquier momento. Por ejemplo, el usuario puede consultar la tabla de enrutamiento de un router. El usuario puede también desconectarse de la simulación actual para seguir trabajando en otra simulación mientras la anterior está siendo ejecutada en un servidor de simulación[28].

NCTUns [28]tiene cuatro modos de operación. Un usuario de la interfaz debe cambiar de modo en los momentos debidos para que el editor de topología funcione correctamente. Los modos de operación son los siguientes:

- *Draw Topology.*
- *Edit Property.*
- *Run Simulation.*
- *Play Back.*

En el modo *Draw Topology* los usuarios pueden agregar y eliminar nuevos nodos/enlaces para construir la topología de la red. En el modo *Edit Property,* un usuario puede editar las propiedades de los nodos, como por ejemplo los módulos de protocolo que un nodo en específico tendrá durante la simulación o las aplicaciones que deben ser ejecutadas por los nodos durante la simulación. El modo *Run Simulation* permite a los usuarios comenzar o detener la simulación. En este modo no se pueden modificar las configuraciones de la simulación. Una vez terminada la simulación, el modo *Play Back* permite reproducir o detener la animación del intercambio de paquete entre los nodos y del movimiento de los nodos móviles.

5.3.2 Motor de Simulación

El motor de simulación de NCTUns [29] es un programa que funciona como un pequeño sistema operativo. A través de un API, este programa provee los servicios básicos de simulación a los módulos de protocolos. Entre los servicios que provee el motor de simulación están:

- El mantenimiento de un reloj virtual.
- La administración de los temporizadores.
- La programación de eventos (*Event Scheduling*).
- El Registro de variables.

El motor de simulación debe ser compilado junto con los módulos de protocolo (descritos en la Sección 5.3.3) para formar un programa de usuario el cual es llamado servidor de simulación. Cuando el servidor de simulación es ejecutado para realizar una simulación, éste toma los archivos generados por el GUI como entrada, realiza la simulación y genera los logs de transferencia de datos y paquetes como salida. En una máquina de simulación solo puede correr un servidor de simulación a la vez ya que este utiliza muchos recursos del kernel.

5.3.3 Módulos de Protocolo

Un módulo de protocolo [28] es una implementación de una capa de la pila de protocolos escrita en C++.NCTUns provee varios módulos de protocolos por defecto. Los usuarios pueden agregar nuevos módulos de protocolos o reemplazar algunos ya existentes por sus propios módulos. La manera más recomendable de desarrollar módulos en NCTUns [23] es modificando el código de los módulos que el simulador trae por defecto.

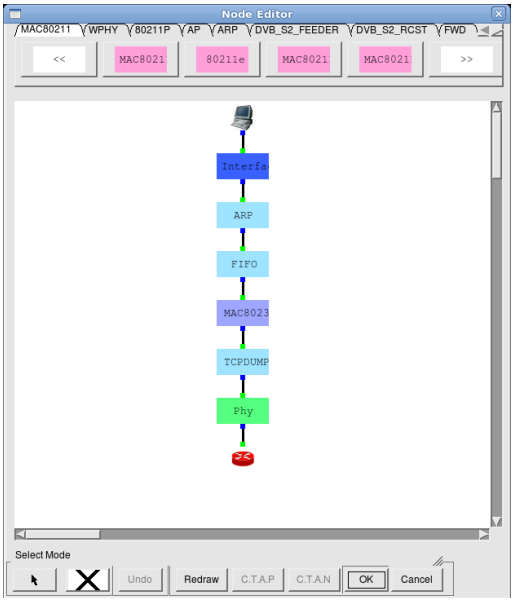


Figura 5.3: Ventana de Edición de Módulos de Protocolo de un Nodo

En la Figura 5.3se puede apreciar la ventana de edición de los módulos de protocolo de un nodo de tipo host que representa un computador, se puede apreciar que el conjunto de módulos del nodo forman una pila de protocolos. A través de esta ventana, el GUI permite a los usuarios visualizar los módulos pertenecientes al nodo en cuestión y además agregar nuevos módulos al nodo o intercambiar un módulo del nodo por otro módulo.

5.3.4 Despachador de Trabajo

El despachador de trabajo [29]es un programa que permite realizar simulaciones concurrentes en múltiples máquinas de simulación. Una máquina de simulación es una máquina donde: primero, fue modificado el kernel para adaptarse a la metodología de simulación, segundo, donde han sido compilados el motor de simulación y los módulos de protocolo y por último en la cual existe un programa llamado coordinador (descrito en la Sección 5.3.5) en ejecución. La instalación de NCTUns completa todos estos requisitos en el computador que es instalado para convertirlo en una máquina de simulación. En la Figura 5.4tomada de [27]se puede ver la arquitectura distribuida de NCTUns. Cuando un usuario envía un trabajo de simulación al despachador de trabajo a través del GUI, el despachador selecciona una máquina de simulación disponible para ejecutar dicho trabajo. Si no hay una máquina de simulación disponible, el despachador puede encolar el trabajo de simulación hasta que se libere una.

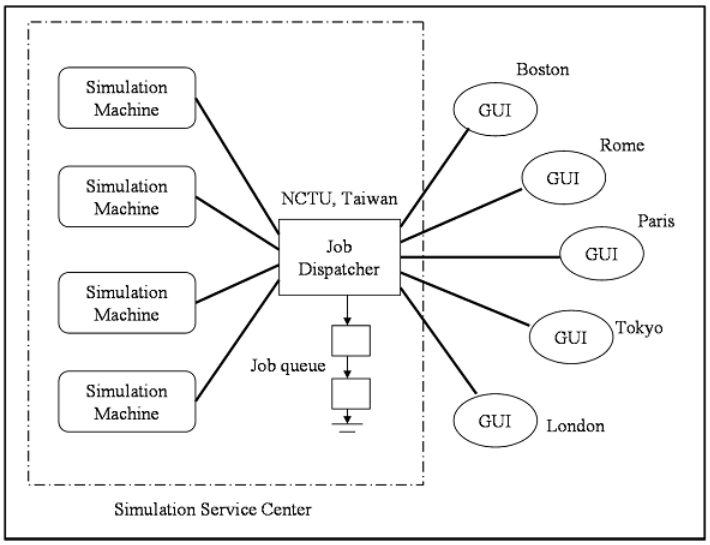


Figura 5.4: Arquitectura Distribuida de NCTUns

5.3.5 Coordinador

El coordinador [28]es un programa ejecutado en cada máquina en la cual reside un servidor de simulación. El coordinador debe ser ejecutado y mantenido en ejecución. Su objetivo es informar al despachador de trabajo si la máquina está ocupada o no. Cuando es ejecutado, se registra inmediatamente con el despachador, luego se encarga de notificarle al despachador de trabajo cuando su estado cambia de ocupado a ocioso y viceversa.

Cuando el coordinador recibe un trabajo del despachador, ejecuta un servidor de simulación para realizar la simulación especificada. Durante la simulación, el coordinador se encarga también de iniciar o terminar algunas aplicaciones que hayan sido especificadas en algunos nodos de la simulación para generar tráfico en la red simulada. El coordinador conoce los IDs de los procesos de estas aplicaciones generadoras de tráfico, por lo tanto es él quien registra los procesos con el kernel. Todas las llamadas al sistema que estén relacionadas con tiempo serán basadas en tiempo virtual en lugar de tiempo real.

Cuando el servidor de simulación está en ejecución, el coordinador se comunica con el despachador y laGUI. Por ejemplo, periódicamente el servidor de simulación envía el tiempo virtual de la red simulada al coordinador, el coordinador luego envía esta información al GUI permitiendo al usuario conocer el progreso de la simulación. El intercambio de mensajes entre el servidor de simulación y el GUI se realiza a través del coordinador, por ejemplo cuando un usuario consulta la tabla de enrutamiento de un router a través de la interfaz gráfica, esto se hace posible con un mensaje al coordinador.

5.3.6 Modificaciones al Kernel de Linux

NCTUns[29] modifica el kernel de Linux para que un servidor de simulación pueda correr correctamente utilizando la metodología de *kernel re-entering*. Por ejemplo, durante una simulación los temporizadores de las conexiones TCP usados en la simulación deben ser activados por tiempo virtual en lugar de por tiempo real. Lo mismo debe suceder con todos los servicios relacionados con tiempo que sean requeridos por aplicaciones que corran en nodos simulados. Por ejemplo, cuando el kernel recibe la llamada al sistema *sleep(5)* hecha por un proceso de una aplicación que corre en un nodo simulado, el kernel

debería suspender la ejecución del proceso por 5 segundos en tiempo de simulación en lugar de tiempo real.

Otro problema que se presenta al implementar la metodología de *kernel re-entering* es que el kernel debe realizar el cambio de puertos UDP/TCP entre un número de puerto especificado por el usuario de la simulación y el puerto que será utilizado en el kernel realmente. Este cambio de puertos debe ser realizado para permitir que varias aplicaciones dentro de la simulación puedan usar el mismo número de puerto en diferentes nodos. Por ejemplo, si en la simulación existen dos nodos que están corriendo un servidor web, ambos van a querer utilizar el puerto bien conocido número 80. Es necesario que el kernel realice un cambio de puertos a la hora de ejecutar estos procesos y que dicho cambio sea invisible para la simulación, de lo contrario no se pudieran realizar simulaciones en las que ocurran este tipo de hechos.

5.3.7 Demonios y Aplicaciones Reales

NCTUns [27]utiliza demonios de enrutamiento como por ejemplo routed o gated que corren a nivel de usuario directamente para llenar las tablas de enrutamiento de los *routers* en una red simulada. NCTUns también permite que los nodos simulados ejecuten aplicaciones a nivel de usuario para la generación de tráfico o para la configuración y monitoreo de la red. Por ejemplo, un nodo puede correr la aplicación *tcpdump* en una red simulada para capturar los paquetes que pasan a través de un enlace. Como se mencionó anteriormente, esta es una de las características que hace al simulador único en comparación con los demás.

5.4 Desarrollo de Módulos

NCTUns [23] provee una plataforma basada en módulos. Un módulo (como se menciona en la Sección 5.3.3) corresponde a una capa en una pila de protocolos. Por ejemplo, el módulo ARP implementa el protocolo ARP (Address Resolution Protocol) de un nodo, mientras que el módulo FIFO (First In First Out) implementa la planificación y manejo de buffer tipo FIFO. Los módulos se enlazan entre sí para formar una pila de protocolos usada por un dispositivo de red. Un desarrollador puede insertar, eliminar o reemplazar un módulo a una pila de protocolos existentes. A través de estas operaciones, el desarrollador puede controlar y cambiar el comportamiento de un dispositivo de red.

Para agregar un nuevo módulo al motor de simulación se requieren 3 acciones: (1) registro del módulo, (2) registro de parámetros de inicialización y (3) registro de variables *Get/Set.*

5.4.1 Registro del Módulo

Todos los módulos deben ser registrados con el motor de simulación antes de que NCTUns pueda usarlos para generar resultados de simulación[23]. Se deben llevar a cabo dos pasos para registrar un módulo con el motor de simulación. Primero, se debe agregar la declaración REG_MODULE (nombre, tipo) a la función main() del archivo nctuns.cc, donde nombre es el nombre del módulo y tipo es el nombre de la clase C++ del módulo correspondiente. El segundo paso es agregar la declaración MODULE_GENERATOR (nombre) al archivo C++ que implementa el módulo. El argumento nombre es el nombre del módulo que se desea agregar.

5.4.2 Registro de Parámetros de Inicialización

Un módulo pudiera tener varios parámetros cuyos valores deben ser inicializados al comienzo de la simulación [23], por ejemplo, un módulo FIFO normalmente tiene un parámetro para especificar el tamaño máximo de cola permitido. Estos tipos de parámetros deben ser registrados explícitamente con el motor de simulación para que sus valores puedan ser inicializados. El registro de los parámetros se realiza usando el macro *vBind* (nombre_exportado, dirección_variable) donde nombre_exportado es el nombre bajo el cual será exportado el parámetro y dirección_variable es una referencia a la dirección de la variable a ser exportada. Cuando la simulación inicie, el motor de simulación tomará el valor especificado en un archivo (archivo *.tcl*) generado a partir de las opciones introducidas por el usuario en el GUI.

5.4.3 Registro de Variables Get/Set

Algunas veces es necesario observar el estado de una variable, un nodo o un protocolo en tiempo de ejecución. Por ejemplo, un usuario puede estar interesado en observar como varía la longitud de cola de una cola FIFO durante la simulación. Los desarrolladores de módulos[23] pueden registrar algunas variables con el motor de simulación para que estas puedan ser exportadas y accedidas en tiempo de ejecución. El motor de simulación provee el macro EXPORT(nombre_variable, modo_permiso) para permitir esta funcionalidad. El parámetro nombre_variable es el nombre de la variable a ser exportada. El parámetro modo_permiso es el modo de permiso de acceso para la variable. El modo de permiso puede ser solo lectura (READ_ONLY), solo escritura (WRITE_ONLY) o ambos.

5.5 Gestión de Recursos de Radio

NCTUns [30] soporta varios modelos de canal.Entre los modelos de canal soportados por NCTUns se encuentran: (1) un modelo simplificado en el cual solo se especifican el rango de transmisión y el rango de interferencia para una interfaz de red. (2) un modelo “two-rayground reflection”, (3) un modelo “Rayleigh Fading”.

Entre los protocolos MAC inalámbricos soportados por NCTUns se encuentran: (1) CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) con RTS/CTS (Request to Send y Clear to Send) usados para redes IEEE 802.11b, (2) Multiple Frequency Time Division Muiltiplexed Access (MF-TDMA) para redes celulares, y (3) protocolos MAC multicanal y multiradio para reducir el efecto de la interferencia en redes mesh inalámbricas. La Figura 5.5muestra la ventana de especificación de la capa física y modelo de canal para los nodos con interfaces inalámbricas.

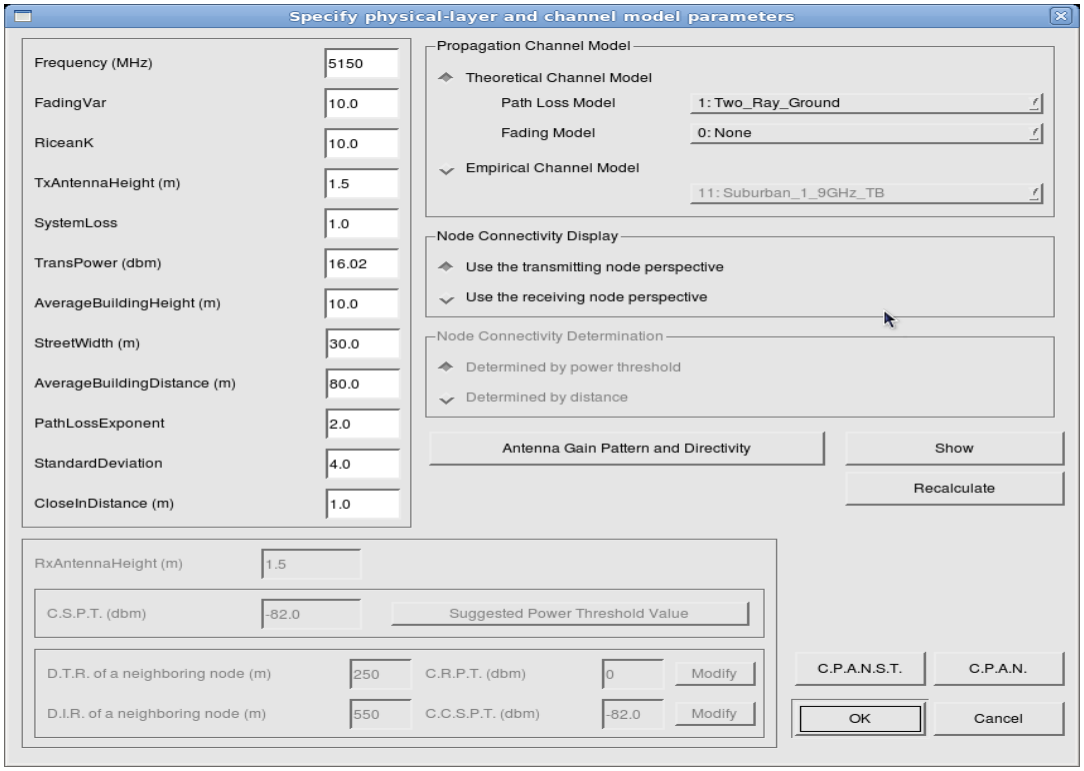


Figura 5.5: Ventana de Especificación de la Capa Física y Modelo de Canal

5.6 Movimiento de Nodos Móviles

NCTUns provee varias maneras de establecer el movimiento de los nodos móviles inalámbricos:

- Generar *waypoints* de manera aleatoria: La GUI de NCTUns contiene una herramienta que permite generar *waypoints* de manera aleatoria para todos los nodos móviles en una simulación, estos *waypoints* se pueden generar uno por cada click para todos los nodos o se puede establecer el tiempo (en segundos de simulación) para el cual se deben generar aleatoriamente los *waypoints* de todos los nodos móviles inalámbricos.
- Importar nodos y movimiento desde un archivo: La GUI provee una opción para importar nodos móviles y sus movimientos desde un archivo con extensión .mdt.En este archivo se describe el tamaño del espacio de simulación, el número total de nodos móviles y se especifica el movimiento de cada nodo móvil.
- Importar Movimientos desde un archivo: Esta opción se utiliza para importar el movimiento de un solo nodo desde un archivo con extensión .mpt.En este archivo se especifica por cada línea, las coordenadas (x,y), el tiempo de llegada y la velocidad del nodo.

5.7 Soporte para Redes Vehiculares

Aunque inicialmente NCTUns no fue diseñado para proveer soporte para redes vehiculares, a partir de la versión 4.0 [24] el simulador NCTUns combinó sus capacidades de simulación de redes con capacidades de simulación de tráfico, permitiendo la construcción de redes de carreteras y vehículos con movimiento automático.

5.7.1 Uso del GUI para la Construcción de Redes de Carreteras

Una manera de construir las redes de carreteras en NCTUns es trazándola manualmente a través del GUI. La Figura 5.6muestra los 3 objetos de caminos que pueden ser utilizados en NCTUns para la construcción de la red de carreteras.El objeto de la izquierda es un segmento de camino, el cual puede tener múltiples canales en ambos sentidos. El objeto del medio representa un cruce de caminos, el cual puede o no contener un semáforo según se especifique en la simulación. Finalmente el objeto a la derecha de la imagen es un unidor de caminos, su función es unir 2 caminos cuyos números de canales sean diferentes.

Figura 5.6: Objetos de Caminos Usados en NCTUns

En la Figura 5.7se puede apreciar una red de carreteras construida a través de la interfaz gráfica de NCTUns. Los nodos provistos por NCTUns que representan vehículos con interfaces inalámbricas deben ser colocados sobre estas carreteras de manera que funcionen correctamente.

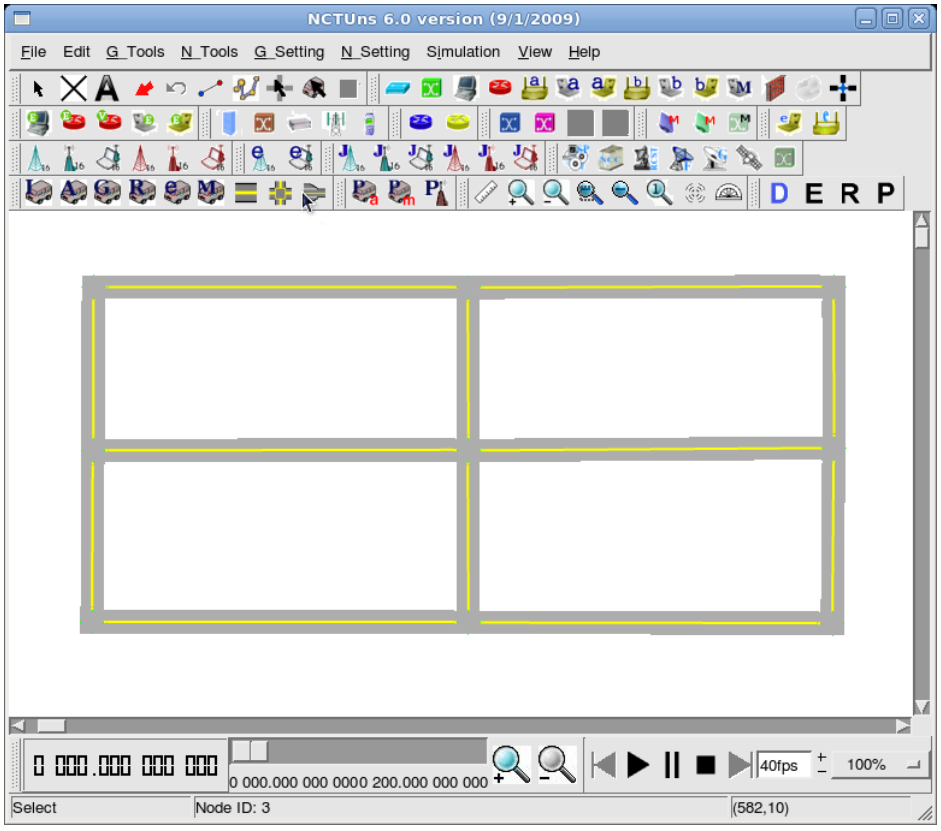


Figura 5.7: Red de Carreteras Construida Usando el GUI de NCTUns

5.7.2 Uso de Shapefiles para la Construcción de Redes de Carreteras

NCTUns permite importar un mapa del mundo real al GUI. Los mapas soportados tienen el formato *shapefile*. El formato *shapefile* es un formato de archivo de datos espaciales

desarrollado por la compañía Esri[17], la cual crea software para sistemas de información geográfica. Los *shapefiles*no almacenan información topología, usan un formato vectorial en el cual los caminos son representados usando líneas.Por lo tanto, al importar un mapa a la simulación, se debe especificar el número de canales que tendrán los caminos en ambas direcciones. En la Figura 5.8se puede apreciar una simulación a la cual se importó un mapa con formato *shapefile*.

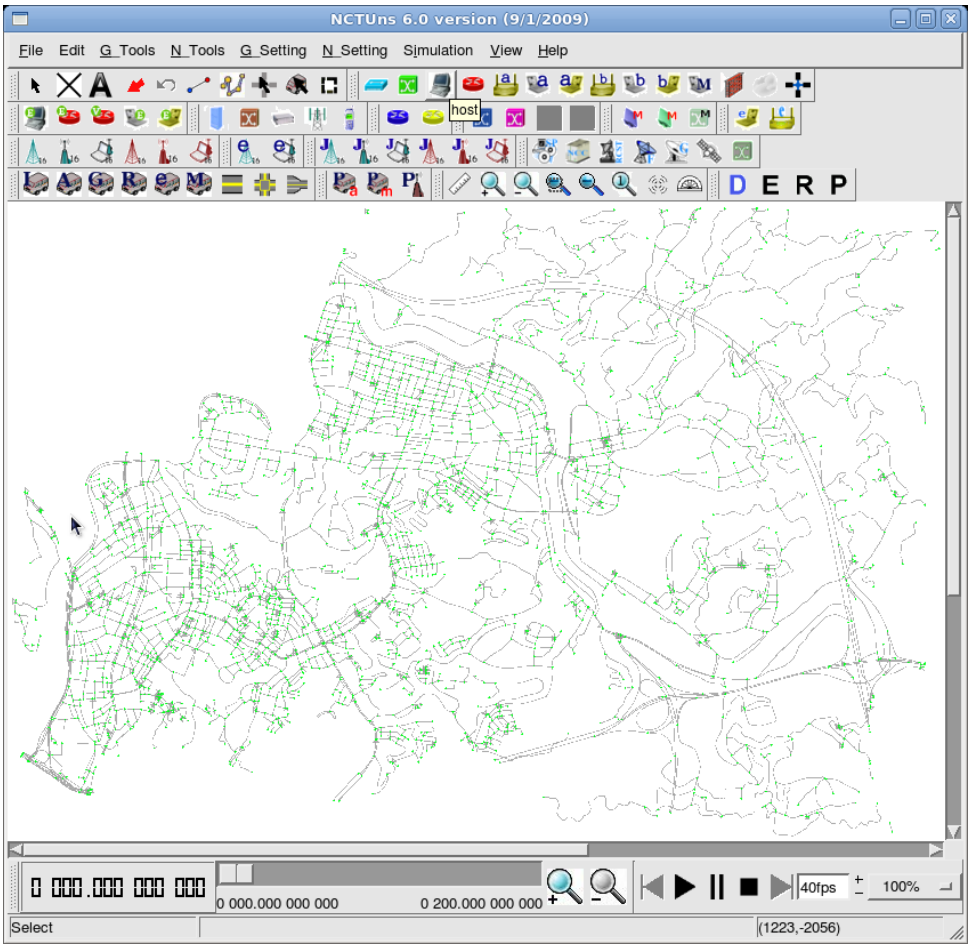


Figura 5.8: Red de Carreteras Construida Importando Shapefile

5.7.3 Vehículos Soportados

NCTUns [28] provee seis tipos de vehículos con diferentes interfaces inalámbricas para la simulación de redes vehiculares. En la Figura 5.9se pueden apreciar los diferentes tipos de vehículos soportados por NCTUns:

- Vehículo I: Vehículo con interfaz 802.11b modo infraestructura.
- Vehículo A: Vehículo con interfaz 802.11b modo ad hoc.
- Vehículo G: Vehículo con un radio GPRS.
- Vehículo R: Vehículo con interfaz satélite RCST.
- Vehículo e: Vehículo con interfaz 802.16eWiMAX móvil.
- Vehículo M: Vehículo con todos los tipos de Interfaz anteriores.

Los vehículos pueden ser desplegados en la red de carretera uno por uno o se puede utilizar una herramienta provista por el GUI para desplegar una cantidad de vehículos

[17] http://www.esri.com

especificada por el usuario, con una separación de metros entre cada vehículo también especificada por el usuario.

Figura 5.9: Vehículos Soportados en NCTUns

5.7.4 Movimiento de Vehículos

Existen 2 enfoques para el control de movimiento de los vehículos en NCTUns[24]. El primero es el enfoque preestablecido, el segundo es el enfoque de autopiloto. En el enfoque preestablecido el usuario debe establecer el camino y la velocidad de cada vehículo antes del comienzo de la simulación. Durante la simulación, cada vehículo se moverá a través de su camino preestablecido a la velocidad preestablecida en la red de carreteras.

En el enfoque autopiloto, el componente encargado de controlar el movimiento de los vehículos en una simulación es llamado *car agent*. En una simulación, el motor de simulación ordena la ejecución de un programa car agent por cada vehículo. Bajo este enfoque, el usuario no debe especificar ni el camino ni la velocidad exacta de cada vehículo, sino debe especificar los parámetros de movimiento del vehículo (velocidad inicial, velocidad máxima, aceleración máxima, desaceleración máxima, etc.) al *car agent*. El *car agent* controlará automáticamente el movimiento del vehículo durante la simulación basado en los parámetros especificados. Cada vehículo determinará dinámicamente su camino y su velocidad según el tráfico que lo rodee y las condiciones del camino.

5.8 Soporte para la Emulación

NCTUns [22]integra la simulación y emulación. Debido al hecho de que en una simulación los nodos pueden correr aplicaciones reales y utilizan la pila de protocolos de Linux, las aplicaciones que corren en nodos reales pueden establecer conexiones TCP/UDP con aplicaciones que corren en nodos simulados en la red emulada.

Para convertir el simulador NCTUns en un emulador,se deben realizar tres operaciones[31]. Primero, el tiempo de simulación debe ser cambiado al tiempo real, con el fin de sincronizar el reloj virtual con el tiempo real. Segundo, se deben agregar entradas de enrutamiento a la tabla de enrutamiento de los dispositivos “reales” para que sus paquetes puedan ser transmitidos a la máquina de simulación. Finalmente, las cabeceras de los paquetes capturados deben ser traducidas para que los paquetes puedan ser intercambiados entre los dispositivos de la red simulada y los dispositivos reales.

6. Trabajos Relacionados

En este capítulo se presentan una serie de trabajos relacionados con el análisis de desempeño de simuladores de redes vehiculares. También, se discuten algunas contribuciones sobre análisis de desempeño realizados con el simulador NCTUns. Finalmente, se presentan algunos trabajos que utilizan herramientas de simulación para el análisis y/o diseño de algoritmos de enrutamiento y modelos de movilidad.

6.1 Evaluación de Desempeño de Simuladores de Redes Vehiculares

Existen varios trabajos con el objetivo de estudiar y comparar las distintas herramientas que permiten la simulación de redes vehiculares. Uno de estos se presenta en el artículo “A Survey and ComparativeStudy of Simulatorsfor Vehicular Ad Hoc Networks (VANETs)” [32]. En este trabajo se estudian distintos simuladores de código abierto que pueden ser usados en las redes vehiculares. Los autores de la investigación dividieron los simuladores en 3 categorías: (1) simuladores de tráfico, (2) los simuladores de red, y (3) los simuladores de redes vehiculares.

Los autores de [32] evaluaron el realismo y la efectividad de las trazas de movimiento generadas por los simuladores de tráfico. Para eso, realizaron una simulación del protocolo WMD (WarningMessageDissemination) utilizando trazas reales y trazas generadas por 3 simuladores: VanetMobiSim, CityMob, SUMO. Estas trazas fueron luego utilizadas en el simulador ns-2. La investigación mostró que las trazas generadas por el simulador VanetMobiSim son las que más se asemejan a la realidad.

En el caso de los simuladores de red se estudiaron: ns-2, GloMoSim, JiST/SWANS y SNS. Se realizó una comparación de los simuladores tomando en cuenta varias características. En la Tabla 5.1[32], se pueden observar los resultados. Una debilidad encontrada en todos los simuladores es el poco soporte para redes vehiculares, como por ejemplo la falta de modelos de movilidad o del soporte para el protocolo 802.11p (a excepción del simulador ns-2.33).

	ns-2	GloMoSim	JiST/SWANS	SNS
Software				
Portability	✓	✓	✓	✓
Freeware	✓	✓	✓	✓
Opensource	✓	✓	✓	✓
Available examples	✓	✓	✓	✓
Continuous development	✓	×	✓	×
Large networks	×	✓	✓	✓
Console	✓	✓	✓	✓
GUI	✓	✓	✓	✓
Scalability	Poor	High	High	High
Ease of setup	Easy	Moderate	Hard	Easy
Ease of use	Hard	Hard	Hard	Hard
VANET				
802.11p	Only for ns-2.33	×	×	×
Obstacles	×	×	×	×
Vehicular traffic flow model	×	×	×	×

Tabla 6.1: Comparación de los Simuladores de Red Estudiados

En el casode los simuladores de redes vehiculares se estudiaron las herramientas TraNS, MobiREAL, NCTUns y GrooveNet. En este estudio se comparan los simuladores bajo parámetros como modelos de movilidad y protocolos soportados para redes vehiculares, facilidad de uso, construcción de red de carreteras, entre otros. En el estudio se concluye que a pesar de la capacidad que tienen estos simuladores de integrar la simulación de

tráfico con la simulación de redes, todavía se requieren mayores contribuciones para que puedan ser aceptados completamente para las investigaciones de redes vehiculares.

En el trabajo de investigación titulado “Propuesta de un Conjunto de Benchmarks para Evaluar el Desempeño de Simuladores de red en el Área de Redes Vehiculares” [33], se desarrollan un conjunto de benchmarks para determinar las capacidades y limitaciones en el área de redes vehiculares de los simuladores ns-3, OMNeT++ y JiST/SWANS. En el trabajo se concluye queOMNeT++ y JiST/SWANS son muy similares, al nivel de consumo de memoria y tiempo de ejecución. Al nivel de protocolos de enrutamiento, DYMO (Dynamic MANET On-demand) y AODV fueron los que arrojaron mejores resultados, siendo DYMO superior a AODV. BATMAN (Better Approach to Mobile AdhocNetworking) mostró un desempeñomuy pobre.Acerca del simuladorJiST/SWANS se concluye que es un simulador eficiente en cuanto a utilización de memoria y tiempo de ejecución pero bastante limitado a nivel de modelos para la simulación de redes VANETs. En el caso de ns-3 se concluye que es un simulador bastante completo a nivel de modelos para simular redes vehiculares y que también maneja bien el uso de la memoria y el tiempo de ejecución con los modelos de OLSR y DSDV.

En el trabajo también se demuestra que los protocolos de enrutamiento para MANETs no son adecuados para ser utilizados en redes vehiculares, ya sea porque les toma tiempo establecer una rutao porque cuando estos protocolos fueron diseñados no estaban pensados para operar en redes con alto dinamismo como lo son las redes vehiculares y no pueden tomar a tiempo los múltiples y rápidos cambios en la topología.

6.2 Evaluación de Desempeño del Simulador NCTUns

Se han realizado varios estudios con el objetivo de evaluar el desempeño del simulador NCTUns. Un ejemplo es el estudio realizado en “NCTUns Tool for Wireless Vehicular Communication Network Researches” [31], donde se evalúa el desempeño de NCTUns con respecto al tiempo de simulación y el uso de memoria por parte de los componentes del simulador. En las simulaciones realizadas, se varíandos parámetros: el número de carreteras en una red de carreteras tipo Manhattan y el número de vehículos desplegados. Por ejemplo, se corrieron algunas simulaciones con 200 vehículos donde se varió la cantidad de carretera en 385, 721, 1057, 1339 y 1729.Se concluyó que tanto el tiempo de simulación como la utilización de memoria subían ligeramente al aumentar el número de carreteras, teniendo poco impacto en el desempeño en la simulación. En otras simulaciones se dejó constante el número de carreteras yse varió el número de vehículos en 250, 350, 450, 550, 650, 750 y 850. De los resultados obtenidos se concluyó que tanto el tiempo de simulación como el uso de memoria por parte del motor de simulación aumentan en relación a la cantidad de vehículos desplegados. Los autores concluyeron que el tiempo simulado es 63 veces más lento que el tiempo real.

Otro trabajo en el que se busca evaluar el desempeño del simulador y emulador NCTUns es “NCTUns Distributed Network Emulator”[25]. En este trabajo se realiza una evaluación de desempeño con el enfoque de emulación distribuida de NCTUns. Se plantea un caso de emulación complejo el cual se dividió en 6 partes para ser emuladas por 6 máquinas de emulación distintas. Los autores concluyen quese puede lograr una aceleración (3.9 veces más rápido)utilizando 6 máquinas que trabajen en conjunto para un mismo escenario de emulación.

6.3 Estudios deRedes VehicularesUsando Herramientas de Simulación

Una gran parte de estudios de redes vehiculares utilizan simuladores, por motivos de reducción de costos y de la facilidad de reproducir los casos de estudios. En el trabajo "AnalyzingRouting Protocol Performanceversus Bitrate in Vehicular Networks"[34]se presenta un estudio realizado a 3 protocolos de enrutamiento: AODV, DYMO y BATMAN. Las simulaciones se desarrollaron basándose en un escenario de una ciudad real utilizando la herramienta SUMO para generar las trazas vehiculares. Las trazas generadas fueron importadas al simulador OMNeT++, se utilizó el modelo de propagación Open Free Space y se utilizó IEEE 802.11a debido a la ausencia de IEEE 802.11p en el ambiente de simulación. Se utilizaron 4 métricas para la evaluación de desempeño: el PDR, OWD promedio, el número de saltos promedio y el NRL. El estudio concluye identificando DYMO como el mejor protocolo de los 3 para redes vehiculares según las métricas tomadas en cuenta. También se percibió que los protocolos no escalaban bien para redes vehiculares ya que el PDR estaba muy bajo.

En el trabajo "Performance Evaluation of TCP and UDP Protocols in VANET Scenariosusing NCTUns-6.0 SimulationTool" [35]se presenta un estudio cuyo objetivo es evaluar el desempeño de los protocolos TCP y UDP en redes vehiculares utilizando el simulador NCTUns. Se utiliza el protocolo AODV para el enrutamiento y las métricas consideradas son: el throughput, el número de paquetes descartados, entre otras. En este estudio se proponen tres escenarios de movilidad utilizando el modelo Manhattan: (1) un escenario de 2x2 con 20 nodos, (2) un escenario de 3x3 con 30 nodos, y (3) un escenario de 5x5 con 50 nodos. El trabajo permitió observar que,como es de esperarse el throughput es mayor para el protocolo UDP que para el protocolo TCP en los 3 escenarios, mientras que la tasa de paquetes descartados es considerablemente menor para el protocolo TCP.

Otro trabajo que utiliza herramientas de simulación para estudiar las redes vehiculares es el trabajo titulado "Performance Evaluación of Manhattan Downtown Scenarios for Vehicular Ad Hoc Networks with CityMob and NCTUns"[36]. En este trabajo se utiliza el simulador de tráfico CityMob para proponer los escenarios de simulación y se utiliza el simulador NCTUns para llevar a cabo las simulaciones. Se tomaron en cuenta parámetros como el retardo medio, la pérdida de paquetes y el *throughput* en función de la velocidad de los vehículos y los modelos de canal de radio.

En el trabajo "Performance Evaluation of AODV and ADV Protocols in VANET Scenarios" [37], se evalúa el desempeño de los protocolos de enrutamiento AODV y ADV en las redes vehicularse utilizando NCTUns-6.0. Se toman 3 métricas para evaluar el desempeño: el número de paquetes descartados, el throughput y el tiempo real de simulación. En el trabajo se plantean 2 tipos de escenarios: (1) escenarios de autopistas, los cuales son caminos de varios canales y con una velocidad máxima alta y (2) escenarios de ciudad, escenarios tipo Manhattan con mayor número de nodos y en los cuales la velocidad máxima es menor. El trabajo concluyó con que ADV superóa AODV en la mayoría de los casos. Los casos de simulación en donde se utilizó el protocolo ADV tuvieron un mejor throughput y unatasa de paquetes descartados entre 80% y 90% menor en comparación a los casos de AODV. El tiempo real de simulación también fue menor para las simulaciones que utilizaban el protocolo ADV.

7. Marco Metodológico

Para cumplir los objetivos trazados en el Capítulo 2, es necesario hacer uso de una metodología de desarrollo de software que permita realizar las tareas de desarrollo de una manera estructurada y planificada. A continuación se presenta la especificación de la metodología utilizada y otros detalles importantes que fueron tomados en cuenta para el desarrollo e implementación del conjunto de benchmarks.

7.1 Adaptación de la Metodología de Desarrollo

Las metodologías de desarrollo ágiles plantean un esquema de trabajo que se divide en 4 fases: Análisis, Diseño, Codificación y Pruebas. Un conjunto de los modelos tradicionales proponen que dicho esquema se ejecute de forma lineal o secuencial para la implementación de la aplicación.

La adaptación del esquema de trabajo escogido se basa en el modelo de desarrollo ágil, dividiendo los requerimientos de la aplicación por módulos, y desarrollando un módulo por iteración. Por lo tanto, en cada iteración se realizan las 4 fases del método tradicional. A continuación se describen los aspectos que se tomaron en cuenta en cada una de las fases de desarrollo.

7.1.1 Planificación

En esta fase se definen los requerimientos y se establecen las prioridades y el tiempo que tomará implementar satisfactoriamente dichos requerimientos. Estos requerimientos luego serán implementados a través de iteraciones que permitan ir agregando funcionalidades o características requeridas hasta obtener el producto deseado.

7.1.2 Diseño

El diseño es la fase en la cual construye la estructura lógica de la solución. Durante la fase de diseño se crean las especificaciones de los componentes de software a desarrollar para cumplir los requerimientos planteados. Eldiseño puede variar según los resultados de la iteración anterior y según los resultados arrojados durante la fase de pruebas.

7.1.3 Codificación

La fase de codificación es la fase en la cual se realiza la implementación de las soluciones diseñadas. El producto de esta fase es el código fuente en el lenguaje escogido para el desarrollo.

7.1.4 Pruebas

Cada iteración culmina con la fase de pruebas, durante esta fase se realizan pruebas para determinar si los requerimientos fueron cumplidos de manera satisfactoria.

Debido al desarrollo secuencial de los requerimientos la mayoría de las veces los resultados de una prueba son el punto de partida de la fase de análisis que se requerirá para la obtención de otro objetivo.

7.1.5 Tecnologías a Utilizar

- C++:Lenguaje de programación híbrido que soporta la programaciónestructurada y orientado a objetos. Uno de los compiladores libres de C++ es el de GNU, el compilador g++.
- NCTUns: Simulador de redes cableadas e inalámbricas que provee características únicas. Usa la pila de protocolo TCP/IP para generar resultados de alta fidelidad. Puede correr aplicaciones reales de Unix en nodos simulados sin necesitar modificaciones.
- ns2 Trace Tookit: Una herramienta diseñada para la visualización del movimiento de nodos en trazas con el formato ns-2. Es útil para visualizar trazas que han sido generadas a partir de otras herramientas como VanetMobiSim o SUMO.
- Wireshark[18]: Analizador de protocolos utilizado para solucionar problemas en redes de comunicaciones, ampliamente empleado como herramienta didáctica para la educación. Posee una interfaz gráfica y muchas opciones de organización y filtrado de información. Permite ver todo el tráfico que pasa a través de una red Ethernet, aunque es compatible con algunas otras tecnologías (e.g., WiFi, Point-to-Point), estableciendo laconfiguración de las interfaces de red en modo promiscuo.
- Tcpdump[19]: Herramienta en línea de comandos cuya utilidad principal es analizar el tráfico que circula por la red, muy similar a Wireshark sólo que sin interfazgráfica.

[18]http://www.wireshark.org/
[19]http://www.tcpdump.org/

8. Marco Aplicativo

Como se señala en el Capítulo 7, este trabajo utiliza la metodología ágil de desarrollo de software conocida comoXP. El presente capítulo describe en detalle cada una de las iteraciones llevadas a cabo para la implementación de la solución utilizando dicha metodología en la cual cada iteración cuenta con una fase de Diseño, Codificación y Prueba.

8.1 Análisis General

Para cubrir los objetivos definidos en el Capítulo 2, se realizó un análisis general a través del cual se pudo determinar de manera global los principales requerimientos de laaplicación. A continuación se presentala lista de requerimientos planteada para generar tanto la implementación delbenchmark circular, en el cual el movimiento de los vehículos está confinado a una carretera circular como del benchmark de la ciudad, en el cual el movimiento de los vehículos corresponde a un movimiento sobre mapas y carreteras reales:

- Diseño de una herramienta para la generación de tráfico UDP por parte de los vehículos y RSUs en ambos benchmarks.
- Diseño de una herramienta para permitir el movimiento de los vehículos en una carretera circular como es requerido enel benchmark circular.
- Diseño de una herramienta para la traducción de trazas ns-2 a un formato reconocible por NCTUns para permitir elmovimiento de los vehículos como es requerido en el benchmark de la ciudad.
- Modificación de la implementación de los protocolos de enrutamiento para el reporte de datos necesarios para las estadísticas.
- Generación de reportes de estadísticas acerca del consumo de CPU, de la memoria y del tiempo de ejecución en las simulaciones.
- Generación de reportes o archivos de salida donde se almacene toda la información y los resultados obtenidos de cada simulación.
- Elaboración de un mecanismo para la configuración y ejecución de los casos de simulación de manera automática, sin el uso del GUI.

8.2 Desarrollo de la Aplicación

El desarrollo de la aplicación se llevó a cabo mediante iteraciones, cada iteración con su propio objetivo. La iteración 1 se basa en el desarrollo de la herramienta de generación de tráfico UDP. Las iteraciones 2 y 3 corresponden al desarrollo de los modelos de movilidad circular y de la ciudad, respectivamente. Las iteraciones 4 y 5 constan en las modificaciones que debieron realizarse a ciertos componentes del simulador para adaptarlos a los requerimientos planteados. La iteración 6 constituyeel desarrollo de una herramienta para el cálculo de los resultados de las simulaciones. Por último, la iteración 7 describe el desarrollo de un mecanismo para la configuración y ejecución de las simulaciones sin necesidad de hacer uso del GUI provisto por NCTUns.

A continuación se especifican de manera detallada cada una de las iteraciones que fueron necesarias y las fases implementadas en ellas según el modelo de programación XP para el desarrollo de la aplicación.

8.2.1 Iteración 1: Generación de Tráfico UDP

Para obtener datos en cuanto al desempeño del simulador en el área de redes vehiculares, además del movimiento de los nodos, es necesario que exista una comunicación constante entre los vehículos y los RSUs involucrados en la simulación.Por esta razón fue necesario desarrollar una aplicación que se encargue de la generación de tráfico UDPen los vehículos durante la simulación.

- **Fase de Diseño:** Para el envío y recepción de tráfico UDP por parte de los vehículos y los RSUs se valióde la ventaja que ofrece NCTUns al permitir que los nodos simulados corran aplicaciones reales (como se menciona en la Sección 5.3.7). Puesto que el simulador NCTUns corre únicamente sobre la plataforma Linux, el programa de generación de tráfico utilizado por los nodos que representan los vehículos y los RSUs fue desarrollado para dicha plataforma.

  Los vehículos y los RSUs ambos deben enviar paquetes UDP, la única restricción es que los RSUs solo pueden enviarpaquetes de datos a los vehículos a diferencia de los vehículos los cuales pueden enviarpaquetes tanto a los RSUs como a otros vehículos.

  Adicionalmente, fue necesaria la creación de un modelo propio para la aplicación UDP. En este modelo se envían datagramas UDP a destinos de forma aleatoria pero a una tasa de bits constante. Se crearon las aplicaciones *UDPforCarApp* para los vehículos y *UDPforRSUApp* para los RSUs. En *UDPforCarApp* los vehículos envían datagramas UDP con una cierta frecuencia cuyo valor se especifica como parámetro de la simulación.El destino se escoge al azar,en forma probabilística se escoge si enviar el datagrama a un vehículo o a un RSU y luego se escoge al vehículo o RSU en cuestión. Para *UPDforRSUApp* no existe la posibilidad de envío entre RSUs, por lo tanto cada vez que un RSU desea enviar un datagrama UDP se escoge un vehículo destino de forma aleatoria.

- **Fase de Codificación:** El desarrollo de las aplicaciones UDP se realizó haciendo uso de la comunicación entre procesos a nivel de sockets que provee labiblioteca Sockets de C++.Se definió el puerto 9999para ser utilizado por las aplicaciones UDP para el intercambio de datagramas. En la Figura 8.1se puede apreciar de manera general el algoritmo utilizado para el envío de datagramaspor parte de los vehículos y los RSUs.

MIENTRAS *tiempo actual < tiempo total a simular*
Bloquearse hasta recibir un datagrama o hasta se venza el tiempo de espera
SI *se venció el tiempo de espera* ***ENTONCES***
Seleccionar un nuevo nodo destino
Crear un nuevo datagrama
Enviar el datagrama
Calcular un nuevo tiempo de espera
SINO *//El nodo recibió un datagrama*
Calcular diferencia entre el timestamp del datagrama yel tiempo actual
Obtener datos como el delay y el TTL del datagrama recibido
//El tiempo de espera no se ha vencido por lo tanto sigue vigente
FINSI
FIN MIENTRAS

Figura 8.1: Algoritmo Utilizado porla Aplicación Generadora de Tráfico UDP

La función *Select*,presentada en laFigura 8.2fue clave para el desarrollo de la aplicación ya que esta permite que un nodo se "bloquee" hasta recibir un paquete de otro nodo o hasta que se venza el tiempo de espera estipulado hastaelenvío del siguiente paquete. Una ventaja de las modificaciones que la instalación de NCTUns realiza al kernel de Linux es que no es necesario especificar que los tiempos de espera deben ser basados en tiempo virtual en lugar de tiempo real, ya que el simulador se encarga de ello durante la simulación. Es decir, la aplicación desarrollada también se puede utilizar en la vida real en cualquier computador que corra sobre la plataforma Linux para la generación de tráfico UDP.

```
int select(
  int nfds, fd_set *readfds, fd_set *writefds,
  fd_set *exceptfds, struct timeval *timeout
)
```

Figura 8.2: Función Select

- **Fase de Pruebas:**Para probar la herramienta de generación de tráfico UDP, se corrieron simulaciones en las cuales los vehículos y RSUs ejecutaban la aplicación de generación de tráfico UDP desarrollada variando los valores de los parámetros, como la longitud del paquete o el tiempo de espera entre el envío de un paquete y el siguiente.

  Para efectos de las pruebas, la herramienta de generación de tráfico imprimía por pantalla(como se puede observar en laFigura 8.3)información relevante durante la simulación al momento de recepción o envío de un paquete como el tiempo en el cual se envióo recibió un paquete o el tiempo de espera que debe transcurrir antes del envíodel siguiente paquete por parte de un vehículo o RSU.

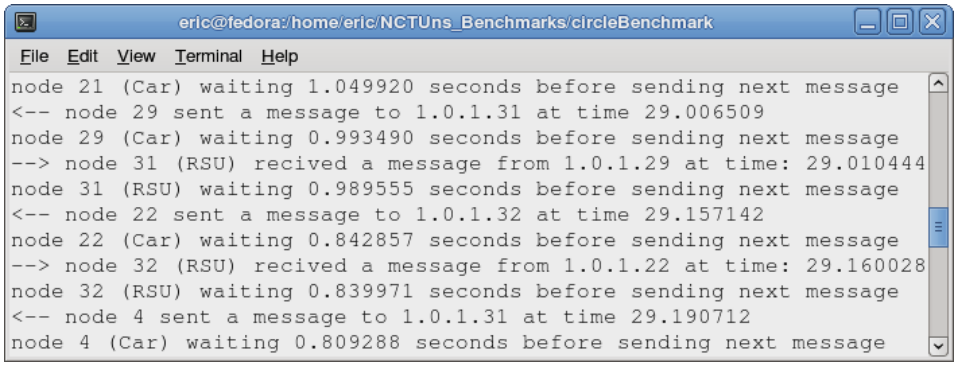


Figura 8.3: Mensajes de Envío y Recepción de Datos

Con la herramienta *tcpdump* se capturaron los paquetes enviados entre los nodos durante la simulacióny posteriormenteutilizando la herramienta *wiresharks*e pudo comprobar,como se puede ver en laFigura 8.4Figura 8.4,el contenido de los paquetes enviados durante la simulación para verificar datos como: las direcciones IP, los puertos y datos de aplicacióncontenidos en dichos paquetes.

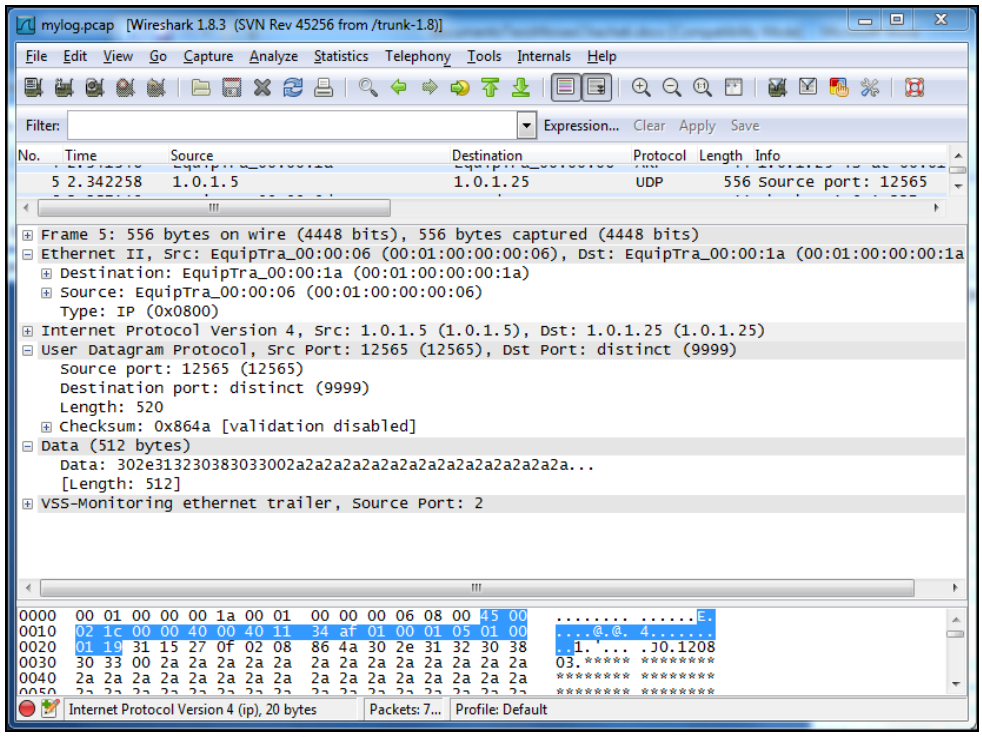


Figura 8.4:Verificación de Paquete UDP Utilizando Wireshark

8.2.2 Iteración 2: Movilidad en el Benchmark Circular

Muchos de los simuladores que permiten la simulación de redes móviles contienen modelos de movilidad ya implementados. El modelo de movilidad circular (tipo redoma) es un modelo sencillo que permite a los nodos moverse de una manera coherente y apegada a la realidad, además permite reproducir simulaciones de manera fácil cuando solo se necesita variar el número de nodos involucrados en la simulación. Otra ventaja de este modelo es que no presenta mayores problemas al momento de aumentar la velocidad de los nodos.

El simulador NCTUns no provee implementación del modelo de movilidad circular. Sin embargo provee un API, a través del cual se pueden desarrollar los modelos de movilidad deseados para las simulaciones.

- **Fase de Diseño:** En el modelo de movilidad circular hay una serie de parámetros que pueden ser modificados antes de la ejecución de la simulación y así obtener casos de pruebas distintos. Esos parámetros son el número total de vehículos, el radio del círculo interno y la separación entre los vehículos en segundos.

  Los parámetros son tomados en cuenta según el modo de operación escogido, existen 3 modos de operación.Para el primer modo se tiene que especificar el número total de vehículos que se desean en la simulación y la separación entre ellos. Entonces el programa calcula el radio del círculo necesario para soportar el número de vehículos dado, con la separación entre ellos especificada. Para el segundo modo se debe indicar la distancia entre los vehículos y el radio del círculo para que el programa calcule el número de vehículos totales bajo esos parámetros. Y por último, para el tercer modo el usuario debe especificar el número total de vehículos y el radio del círculo, siendo no significativa la separación entre los vehículos.

  En resumen se tiene que la topología circular puede contener más de un canal con un ancho dado llamado *laneWidth* y cada canal soporta una velocidad indicada por parámetro. Los canales pueden tener sentidos distintos, esto se logra a través de las velocidades, si la velocidad es positiva los vehículos circulan en sentido horario y si es negativa irán en sentido anti-horario. En laFigura 8.5 se observa la topología circular para los benchmarks.

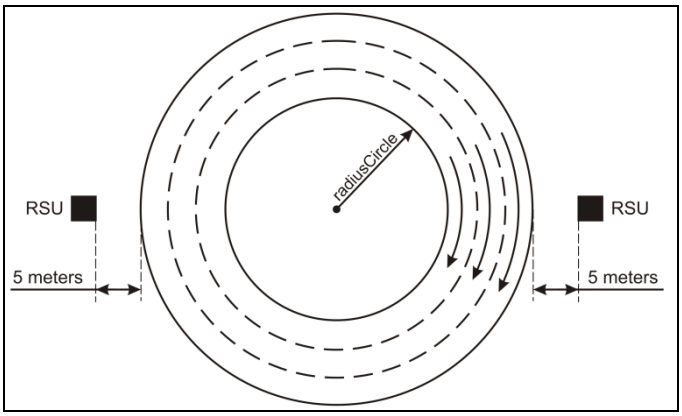


Figura 8.5: Topología Circular

  El movimiento de los vehículos se lleva a cabo por un nodo que corre la aplicación *circleController*. Esta aplicación se encarga de calcular y actualizar las posiciones de todos los vehículos durante la simulación basándose en los parámetros mencionados anteriormente como el ancho y la velocidad de los canales, el radio del círculo, entre otros.

- **Fase de Codificación:** El desarrollo de la aplicación *circleController*se realizó utilizando el API *tactic_api* provisto por NCTUns para el desarrollo de aplicaciones para redes móviles y redes tácticas. Este API provee funciones de gran valor para la implementación de modelos de movilidad, una de estas funciones es la función

*usleepAndReleaseCPU()*mostrada en laFigura 8.6. Estafunciónes similar a la función *usleep* y es utilizada con el mismo propósito,la diferencia es que ésta indica al motor de simulación que puede liberar el procesador mientras se cumple el tiempo especificado a través del parámetro*usecond* para que el motor de simulación pueda aprovechar el procesador para otras aplicaciones ejecutadas por otros nodos durante la simulación.

```
void usleepAndReleaseCPU(
  int myTCPsockfd, int mynid, int usecond, int pre_schedule_select
)
```

Figura 8.6: Función usleepAndReleaseCPU

Otra función indispensable para el desarrollo de la aplicación es la función *createTCPSocketForCommunicationWithSimulationEngine()* (Figura 8.7). A través de esta función se establece el enlace de comunicación entre la aplicación que corre en los nodos y el motor de simulación para poder utilizar otras funciones de importancia como la función usleepAndReleaseCPU.

```
int createTCPSocketForCommunicationWithSimulationEngine (
  int mynid, int gid1, int gid2, int gid3, int gid4, int gid5,
  int gid6, int gid7, int gid8, int humanControlled, int &UDPsocketFd2,
  int proc_type, int moreMsgFollowing
)
```

Figura 8.7: Función createTCPSocketForCommunicationWithSimulationEngine

En la Figura 8.8se muestra un segmento de código de la aplicación *circleController*. En él se puede apreciar el principal objetivo de la aplicación el cual es calcular y actualizar la posición de todos los nodos. Esto se hace posible mediante la función*setCurrentWaypoint()* presentada en la línea 10 de la Figura 8.8.En la línea 2 del segmento de código también se puede apreciar el uso de la función *usleepAndReleaseCPU* mencionada anteriormente, en este caso el tiempo de espera equivale al valor del parámetro *updateInterval* en microsegundos. Básicamente el nodo que corre la aplicación *circleController* se encarga de actualizar las posiciones de los nodos por intervalos de tiempo según el valor establecido antes de la simulación a través del parámetro *updateInterval*.

```
1  while (1) {
2    usleepAndReleaseCPU(myTCPsockfd, mynid, updateInterval*1000000, 1);
3    for(int carIndex=0;carIndex<totalCars;carIndex++){
4      cars[carIndex].curAngle += cars[carIndex].deltaAngle;
5
6      if (cars[carIndex].curAngle>=360.0) cars[carIndex].curAngle-= 360.0;
7
8      nextX=cxcy+cars[carIndex].laneRadius*cos(cars[carIndex].curAngle/360.0*2.0*M_PI);
9      nextY=cxcy+cars[carIndex].laneRadius*sin(cars[carIndex].curAngle/360.0*2.0*M_PI);
10     n =setCurrentWaypoint(myTCPsockfd,carIndex+1,nextX,nextY,nextZ,1);
11   }
12 }
```

Figura 8.8: Segmento de Código del circleController

- **Fase de Pruebas:**Para probar la implementación de la movilidad circular se recurrió al GUI que ofrece NCTUns.Como se puede apreciar en laFigura 8.9, el GUI permite reproducir el movimiento de los nodos luego de la ejecución de la

simulación. También se procedió a verificar las posiciones de los nodos almacenadasen un archivo plano de texto mientras se fueron generando durante la simulación como se aprecia en laFigura 8.10. Se hicieron diversas pruebas a través de simulaciones variando el número de canales, el radio o el modo.

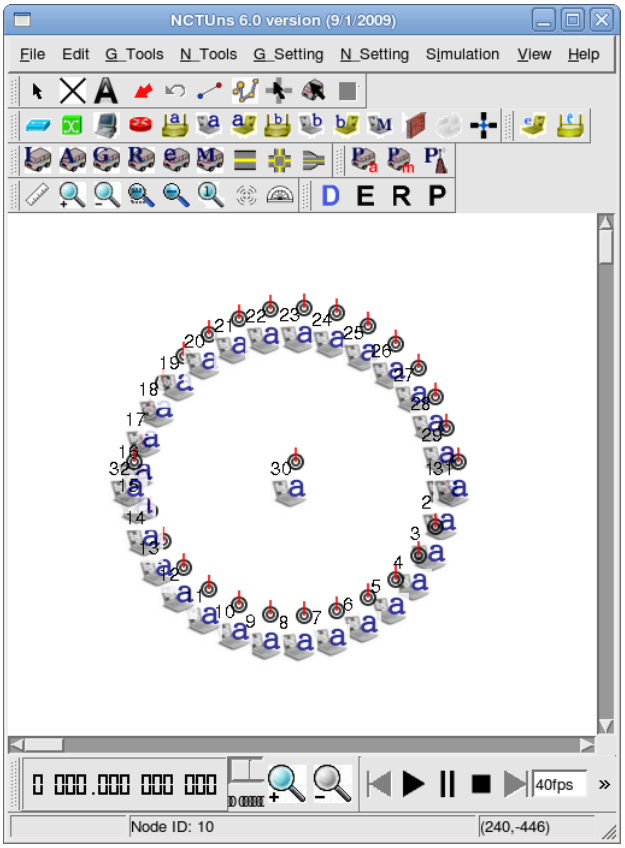


Figura 8.9: Prueba del Benchmark Circular Utilizando el GUI de NCTUns

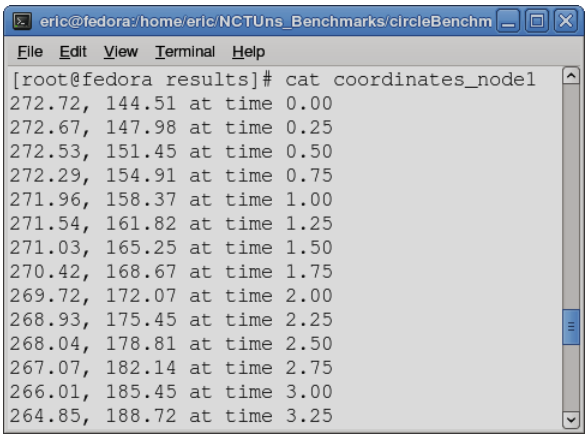


Figura 8.10: Prueba del Benchmark Circular Utilizando Archivo de Texto

8.2.3 Iteración 3: Movilidad en el Benchmark de la Ciudad

- **Fase de Diseño:** Uno de los tópicos más importantes en la simulación de redes vehiculares es la movilidad de los nodos. Es importante la utilización de un modelo de movilidad realista para que los resultados de la simulación reflejen el comportamiento real de una red vehicular. Por ejemplo, la movilidad de un vehículo está típicamente limitada a las calles que están separadas por edificios, árboles u otros objetos. Un modelo de movilidad realista con suficiente nivel de detalle es

crítico para obtener resultados precisos. Son estas las razones que motivaron a la implementación en el simulador NCTUns del benchmark de la ciudad en donde la movilidad de los nodos simulados se realiza sobre topologías de redes de carreteras reales.

La solución fue diseñada pensando en utilizar mecanismos sencillos que permitan obtener y reproducir la movilidad de los nodos sobre la red de carreteras deseada. La mayoría de los simuladores permiten la implementación de este modelo de movilidad a partir de archivos de trazas de movilidad ns-2, los cuales describen el movimiento de los nodos en espacio y tiempo haciendo referencia también a la velocidad con la cual se desplazan. Sin embargo NCTUns contiene su propio formato de trazas que pueden ser importados en la simulación.

Es de interés utilizar los archivos de trazas de movilidad ns-2 ya que estos garantizan la posibilidad de crear simulaciones de redes vehiculares obteniendo una movilidad bastante realista y brindando la capacidad de reproducir el mismo experimento varias veces y bajo las mismas condiciones en distintos simuladores, facilitando la comparación entre estos.

Otro motivo para la utilización de trazas de movilidad ns-2 es que hace posible la utilización de los simuladores de tráfico vehicular SUMO y VanetMobiSim, necesarios para la creación de los archivos de trazas de movilidad ns-2. La importancia que tienen estos simuladores de tráfico es que son capaces de importar mapas digitales (OSM para SUMO y TIGER/Line para VanetMobiSim) con lo cual se pueden crear simulaciones de tráfico vehicular sobre escenarios reales. Cada uno de estos simuladores de tráfico tienen sus características y limitaciones, ambos tienen la capacidad de generar trazas de movilidad adecuadas a lo que se requiera, por esta razón para generar trazas de movilidad se utilizaron ambas herramientas aprovechando así las ventajas que cada una ofrece.

La solución consiste en diseñar una aplicación que permita la traducción de trazas de movilidad ns-2 al formato utilizado por el simulador NCTUns para el movimiento de los nodos.

Una vez que se obtiene una traza de movilidadadecuada, es importante que ésta sea leída de manera correcta por el simulador y que el movimiento de los nodos en la simulación corresponda con lo que sugiere el archivo de traza de movilidad ns-2. En la fase de codificación se explicará de manera detallada la forma que se desarrolló la aplicación para la traducción de trazas con el formato ns-2 a trazas con el formato aceptado por NCTUns.

- **Fase de Codificación:**La aplicación encargada de la traducción de las trazas se desarrolló en el lenguaje C++. Esta aplicación comienza con la lectura del archivo de trazas ns-2 para almacenar los datos en las estructuras presentadas en laFigura 8.11. Seguidamente, se invoca la función presentada en la Figura 8.12la cual se encarga de recorrer todos los movimientos almacenados en las estructuras mencionadas anteriormente para ir interpretando los movimientos a la manera esperada por el simulador NCTUns.

```
typedef struct ns2Entry{
  double time;
  double posX;
  double posY;
  double speed;
}ns2Move;

typedef struct ns2Node{
  double initialX;
  double initialY;
  vector <ns2Move> movements;
}ns2Nodes;

typedef struct ns2Data{
  int totalNodes;
  ns2Nodes node[255];
}NS2;
```

Figura 8.11: Estructuras de Datos Para el Almacenamiento de las Trazas ns-2

```
void translateNS2TraceToNCTUNSTrace(NS2 *sNS2, NCTUns *sNCTUns)
```

Figura 8.12: Función translateNS2TracetoNCTUNSTrace

Una vez que los movimientos han sido traducidos al formato NCTUns, se realiza la escritura del archivo de trazas que será importado durante la simulación para implementar la movilidad en el benchmark de la ciudad. En la Figura 8.13se puede apreciar el procedimiento encargado de realizar dicha escritura.

```
void writeNCTUnsTraceFile(FILE *mdt, NCTUns *sNCTUns,int totalCars){
  fprintf(mdt,"# The size of the working area.\n");
  fprintf(mdt,"%f %f 1000\n",sNCTUns->maxX,sNCTUns->maxY);
  fprintf(mdt,"# Number of mobile nodes\n");
  fprintf(mdt,"%d\n",totalCars+totalRSUs);
  for(int i=0;i<totalCars;i++){
    if(i!=0) fprintf(mdt,"\n");
    fprintf(mdt,"NODE_BEGIN ID_NODE_80211%c_ADHOC\n",opModeCar);
    fprintf(mdt,"# Node %d\n# initial location x(m) y(m) "
        "arrival_time(s) pause_time(s) speed(m/s)\n",i+1);
    fprintf(mdt," %f %f %f %.1f %f\n",sNCTUns->node[i].movements[0].posX,
      sNCTUns->node[i].movements[0].posY, sNCTUns->node[i].movements[0].time,
      sNCTUns->node[i].movements[0].pauseTime,sNCTUns->node[i].movements[0].speed);
    fprintf(mdt,"# PATH_BEGIN number_of_turning_points\n");
    if(sNCTUns->node[i].movements.size()>0)
      fprintf(mdt," PATH_BEGIN %d\n",sNCTUns->node[i].movements.size());
    else
      fprintf(mdt," PATH_BEGIN %d\n",0);
      fprintf(mdt,"# x(m) y(m) arrival_time(s) pause_time(s) speed(m/s)\n");
    for(int j=0;j<sNCTUns->node[i].movements.size();j++){
      fprintf(mdt," %f %f %f %.1f %f\n",sNCTUns->node[i].movements[j].posX,
        sNCTUns->node[i].movements[j].posY, sNCTUns->node[i].movements[j].time,
        sNCTUns->node[i].movements[j].pauseTime,sNCTUns->node[i].movements[j].speed);
    }
    fprintf(mdt," PATH_END\n");
    fprintf(mdt,"NODE_END\n");
  }
}
```

Figura 8.13: Función writeNCTUnsTraceFile

- **Fase de Pruebas:** Las pruebas en esta iteración tienen que ver con la comparación entre la movilidad descrita por los archivos de trazas y la movilidad reproducida en los nodos simulados. Es importante verificar que existe una correspondencia entre el movimiento que realizan los nodos simulados y el movimiento que indica el archivo de trazas de movilidad ns-2. Para lograr esto, se

puede evaluar que la posición de los nodos simulados en el tiempo corresponda con lo que indica el archivo de trazas.

Otra prueba realizada consistió en verificar y comparar de manera visual los movimientos de la traza ns-2 a través de la herramienta *ns2 Trace Toolkit*(Figura 8.14) y los movimientos de la traza NCTUns a través del GUI de NCTUns (Figura 8.15)

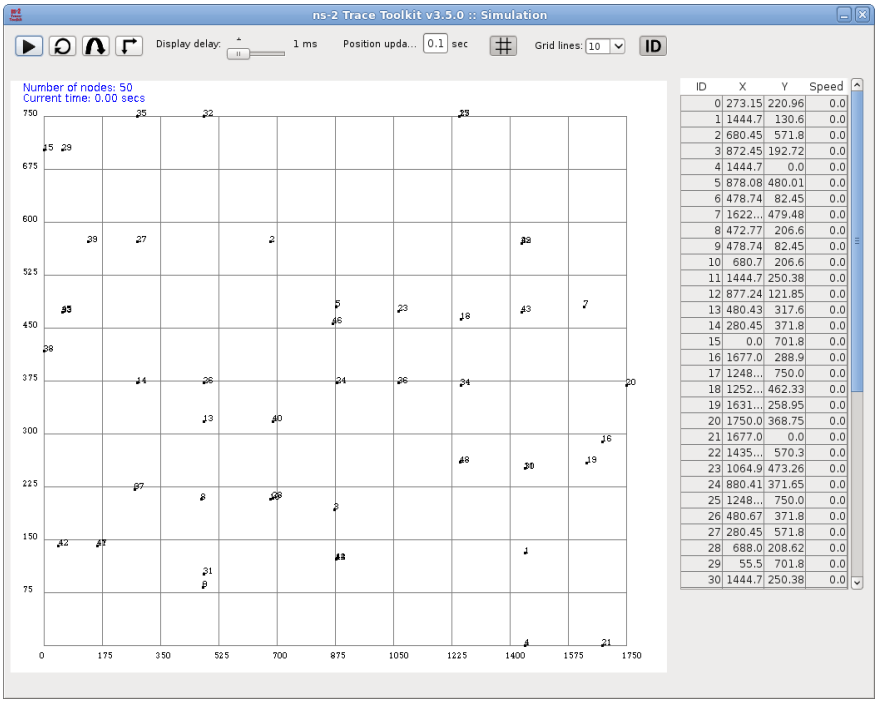


Figura 8.14: Comprobación del Benchmark de la Ciudad Utilizando la Herramienta ns-2 Trace Toolkit

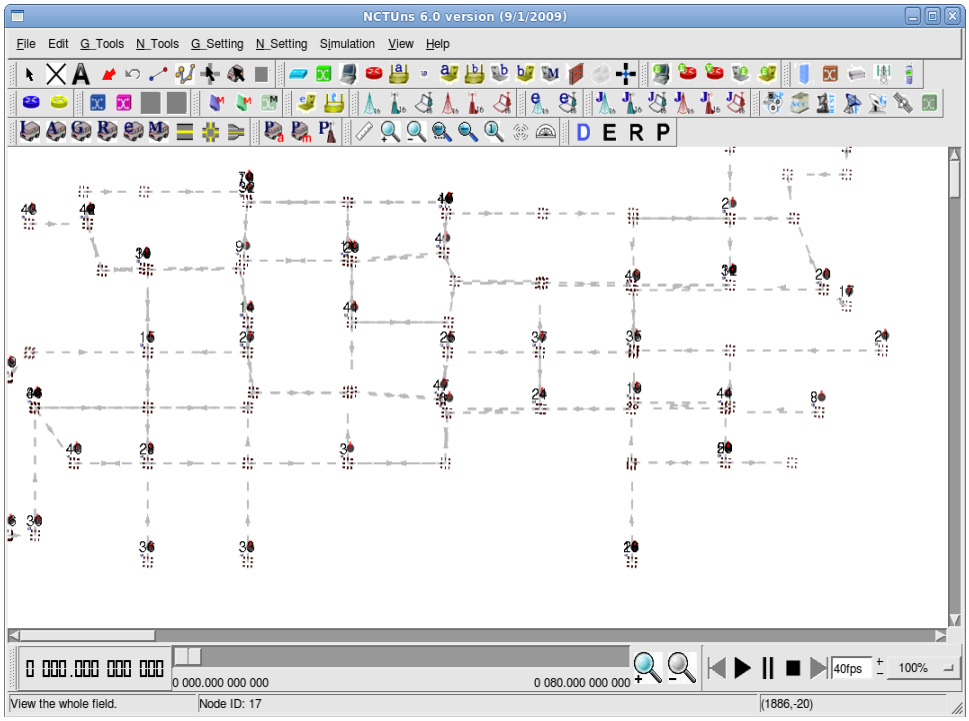


Figura 8.15: Comprobación del Benchmark de la Ciudad Utilizando el GUI de NCTUns

8.2.4 Iteración 4: Modificación de la Implementación de los Protocolos de Enrutamiento para Redes Móviles de NCTUns

El simulador NCTUns provee implementaciones de los protocolos de enrutamiento AODV, ADV y DSDV. Sin embargo para poder registrar datos como el número total de mensajes enviados por un protocolo de enrutamiento, es necesario modificar la implementación de los protocolos para agregar esta funcionalidad.

- **Fase de Diseño:** Los módulos de protocolo (Sección 5.3.3) en NCTUns son programas escritos en C++ que implementanla funcionalidad de los protocolos de la vida real. Para añadir a estos módulos la capacidad de conteo del número total de mensajes enviados por el protocolo de enrutamiento es necesario agregar ciertas variables y funciones a los programas *aodv.cc*, *advd.cc* y *dsdv.cc* contenidos en el simulador NCTUns.

  Los datos finales deben ser escritos en un archivo de texto plano para luego ser analizados por la aplicación encargada de generar los resultados (esta aplicación se describe en la Sección 8.2.6).

- **Fase de Codificación:** La fase de codificación consistió en agregar ciertas líneas de código a los programas que definen los protocolos de enrutamiento a ser evaluados en los benchmarks. En la Figura 8.16se puede apreciar un segmento de código delmódulo AODV.El conjunto de lineas desde la linea 13 a la 17 corresponde a la declaraciónde variables que debieron ser incorporadas para actuar como contadores del numero de paquetes enviados por el protocolo AODV en este caso.

```
1  class AODV : public NslObject {
2                                    // default
3    int  HELLO_INTERVAL;            //1000
4    int  ALLOWED_HELLO_LOSS ;       //2
5    int  ACTIVE_ROUTE_TIMEOUT;      //3000
6    int  DELETE_PERIOD;             //3000
7    int  NET_DIAMETER;              //15
8    int  NODE_TRAVERSAL_TIME;       //40 ms
9    int  RREQ_RETRIES;              //5
10   int  RREQ_RATELIMIT;            //10/per sec
11   int  RERR_RATELIMIT;            //10/per sec
12
13   static int numHelloRREP;  //0
14   static int numRREP;      //0
15   static int numRREQ;      //0
16   static int numRERR;      //0
17   int SHOW_MESSAGES;       //0
```

Figura 8.16: Segmento de Código Agregado al Módulo AODV

Una vez incorporados los contadores fue necesario ubicar en el código del módulo todos aquellos segmentos en los cuales se trate el envío o enrutamiento de un paquete correspondiente al protocolo AODV. En la Figura 8.17 se tiene un ejemplo de un segmento de código cuya función es el envío de un paquete *Hello* del protocolo AODV, como se puede apreciar, la línea 7 fue agregada con el fin de

incrementar el valor del contador de paquetes *Hello* enviados por el protocolo. De esta misma manera se procedió a modificar los otros protocolos de enrutamiento para redes móviles soportados por NCTUns.

```
BASE_OBJTYPE(type);
    type = POINTER_TO_MEMBER(AODV, sendHello);
    SendHello_timer.setCallOutObj(this, type);
    // send hello every HELLO_INTERVAL
    SendHello_timer.start((hello_interval_ + tmprandom), 0);

    numHelloRREP++;
    if (SHOW_MESSAGES)
        printf("node %d sent RREP Message (HELLO)\n",ipv4addr_to_nodeid(*mip));

    return (put(pkt, sendtarget_));
}
```

Figura 8.17: Segmento de Código del Módulo AODV

Los datos calculados por los módulos de protocolo pueden ser obtenidos en cualquier momento de la simulación gracias al mecanismo provisto por NCTUns para la comunicación de los módulos con otros componentes. Este mecanismo se fundamenta en el uso de la función *command()* dentro de la cual los desarrolladores pueden definir su propia sintaxis para obtener valores de variables definidas dentro del módulo. En la Figura 8.18:se puede ver la función *command()*desarrollada, correspondiente al módulo AODV.

```
int AODV::command(int argc, const char *argv[]) {
  char tmpstr[100];
  if ( argc == 2 ) {
    char *workdir = getenv ("NCTUNS_WORKDIR");
    if ( !strcmp(argv[0], "Get")&&(argc>=2)) {
  /* Obtener el numero de mensajes enviados por el protocolo AODV */
     if (!strcmp(argv[1], "numAODVMsg")) {
       printf("numHelloRREP: %d\n",numHelloRREP);
       printf("numRREP: %d\n",numRREP);
       printf("numRREQ: %d\n",numRREQ);
       printf("numRERR: %d\n",numRERR);
       sprintf(tmpstr,"echo %d %d %d %d >> %s/AODVstats", numHelloRREP,
       numRREP,numRREQ,numRERR,workdir);
       system(tmpstr);
       return(1);
     }
    }
  }
  return(NslObject::command(argc, argv));
}
```

Figura 8.18: Función Command del Módulo AODV

- **Fase de Pruebas:** La fase de pruebas de esta iteración consistió en realizar distintas simulaciones que permitieran verificar que los módulos de protocolos correspondientes a los protocolos AODV, ADV y DSDV generaban los datos necesarios para el posterior cálculo de estadísticas como el Normalized Routing Load.

8.2.5 Iteración 5: Generación de Datos para Estadísticas

La aplicación descrita en la Sección 8.2.1 se encarga de la generación de tráfico UDP en las redes simuladas. Este tráfico es generado con el fin de obtener estadísticas para evaluar el simulador NCTUns en ambos benchmarks. En algunos casos no es sencillo obtener valores necesarios para poder calcular las estadísticas, como es el caso correspondiente al número de saltos.Para calcular el número de saltos promedio es necesario conocer el valor del campo TTL (Time to Live)de la cabecera IP de cada paquete recibido por cada nodo.

- **Fase de Diseño:** En NCTUns los mensajes envidados entre los nodos en las simulaciones pasan a través de la pila de protocolos TCP/IP de Linux como se menciona en la sección 5.2. Para obtener el valor del campo TTL de los paquetes recibidos es necesario modificar el módulo *Interface*ya que, como se puede apreciar en la Figura 8.19, este es el último módulo a través del cual pasa un paquete antes de entrar al kernel.Es decir este es el último módulo que tiene acceso al campo TTL antes de la recepción de los datos por parte de la aplicación.

Figura 8.19: Módulos de un Nodo Móvil con Interfaz 802.11b

Adicionalmente se requiere agregar una marca de tiempo o *timestamp*a los paquetes enviados por la aplicación UDP descrita en la Sección 8.2.1 para que al momento de recepción de los paquetes se pueda hacer el cálculo del *One Way Delay*mediante las diferencias de tiempo de recepción y tiempo de emisión de cada paquete.

- **Fase de Codificación:** La fase de codificación consistió en dos partes:

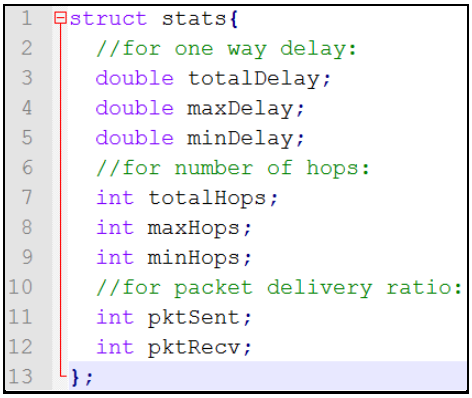
La primera parte consistió en la adaptación al módulo*Interface* para realizar modificación al segmento de datos de los paquetes recibidos para incluir el valor del campo TTL. En la Figura 8.20se puede apreciar un segmento de código del módulo *Interface*. La instrucción presentada en la línea 14 permite obtener el valor del campo TTL, luego la instrucción presentada en la línea 15 permite escribir dicho valor en el segmento de datos del paquete recibido. Se utiliza el valor 36 ya que los bytes anteriores corresponden a la cabecera IP (20 bytes), la cabecera UDP (8 bytes) y 8 bytes restantes corresponden al *timestamp* colocado ahí por la aplicación UDP del nodo emisor para de esta manera hacer posible que la aplicación generadora de tráfico que recibe el paquete tenga acceso a dicho valor para el posterior cálculo de estadísticas como el número de saltos promedio de los paquetes enviados durante la simulación.

Al culminarse el tiempo de simulación los datos obtenidos por cada nodo son escritos en un archivo de texto plano para su posterior uso por parte de la aplicación que calcula los resultados. Esta aplicación se encuentra descrita en la Sección 8.2.6.

```
1  int interface::recv(ePacket_ *pkt) {
2
3    Packet      *pkt_;
4    char       *p = 0 ;
5    int      n;
6
7    assert(pkt&&(pkt_=(Packet *)pkt->DataInfo_));
8    p = pkt_->pkt_sget();
9
10   u_char ttl;
11   iphdr = (struct ip *)pkt_->pkt_sget();
12   if (iphdr->ip_p == IPPROTO_UDP) {
13       //printf ("Packet is UDP\n");
14       GET_IP_TTL(ttl, iphdr);
15       p[36]=ttl;
16   }
```

Figura 8.20: Segmento de Código del Módulo Interface

La segunda parte consistió en la modificación a las aplicaciones *UDPForCarApp* y *UDPforRSUApp* para el registro de datos como el número máximo de saltos de los paquetes recibidos, el retraso, el número de paquetes enviados, entre otros. En laFigura 8.21se puede apreciar la estructura utilizada para almacenar dichos datos. Al culminarse el tiempo de simulación, los datos obtenidos por cada nodo son escritos en un archivo de texto plano para su posterior uso por parte de la aplicación que calcula los resultados. Esta aplicación se encuentra descrita en la Sección8.2.6.



```
struct stats{
  //for one way delay:
  double totalDelay;
  double maxDelay;
  double minDelay;
  //for number of hops:
  int totalHops;
  int maxHops;
  int minHops;
  //for packet delivery ratio:
  int pktSent;
  int pktRecv;
};
```

Figura 8.21: Estructura para Almacenamiento de Datos para Estadísticas

- **Fase de Pruebas:**Las pruebas unitarias para evaluar el correcto funcionamiento de la generación de datos para las estadísticas consistieron en la ejecución de simulaciones controladas con pocas cantidades de nodos y en las cuales se pudiera verificar por medio de herramientas como *tcpdump* y *wireshark* que el número de saltos o el número de paquetes enviados y recibidos durante la simulación coincidía con los valores reales.

8.2.6 Iteración 6: Recopilación de Datos y Presentación de Resultados

La solución planteada para la generación de tráfico UDP por los nodos involucrados en la simulación, específicamente lasaplicaciones*UDPforCarApp*y *UDPforRSUApp (*descritas en la Sección 8.2.1*)* implica que los resultados son generados individualmente por cada nodo. Sin embargo es necesario obtener resultados a un nivel macro. Por esta razón fue necesario el desarrollo de una aplicación para recopilar los datos arrojados por cada nodo individualmente y realizar un cálculo de los resultados de manera global.

- **Fase de Diseño:** Los resultados generados por cada nodo son escritos en un archivo de texto plano por cada nodo. Se diseñó la aplicación *getResults* para analizar el contenido de estos archivos y realizar un cálculo general de estadísticas como el One Way Delay o el Packet Delivery Ratio entre otras puntualizadas en el planteamiento del problema (Sección2.1).

  La aplicacióndiseñada también debe analizar el contenido del archivo generado por el módulo de protocolo modificado (Sección 8.2.4) para igualmente mostrar estadísticas sobre el protocolo de enrutamiento como el Normalized Routing Load.

- **Fase de Codificación:** Esta fase consistió en el desarrollo de la aplicación *getResults* la cual fue desarrollada utilizando el lenguaje C++. Esta aplicación consta básicamente de un conjunto de funciones para la lectura de los archivos de datos escritos individualmente por cada vehículo y RSU al finalizar la simulación, y otras funciones para mostrar los resultados de las simulaciones de manera ordenada y sistemática tanto en la consola de comandos como en un archivo.

  En la Figura 8.22 se puede apreciar un segmento del código de la aplicación *getResults.*En él se puede apreciar como la aplicación se encarga de leer los archivos escritos por los vehículos y los RSU durante la simulación para posteriormente calcular algunas estadísticas como el delay mínimo y máximo o el número de paquetes recibidos y enviados por el nodo.

```
for(int i=1;i<=totalCars+totalRSUs+1;i++)
{
  fscanf(results,"%d %f %f %f %d  %d  %d  %d  %d", &nodeID
        , &nodeTotalDelay, &nodeMaxDelay, &nodeMinDelay, &nodeTotalHops
        , &nodeMaxHops, &nodeMinHops, &nodePktSent, &nodePktRecv);

    totalDelay+=nodeTotalDelay;

    if (nodeMaxDelay>maxDelay) maxDelay=nodeMaxDelay;

    if (nodeMinDelay<minDelay) minDelay=nodeMinDelay;

    totalHops+=nodeTotalHops;

    if (nodeMaxHops>maxHops) maxHops=nodeMaxHops;

    if (nodeMinHops<minHops) minHops=nodeMinHops;

    pktRecv+=nodePktRecv;

    pktSent+=nodePktSent;

}
```

Figura 8.22: Segmento de Código de la Aplicación getResults

- **Fase de Pruebas:** Las pruebas de esta iteración consistieron en realizar el cálculode manera manual de las estadísticas a partir de los datos en los archivos generados por los nodos y por el módulo de enrutamiento para luego comparar los resultados con los datos calculados y exhibidos por la aplicación *getResults*. En la Figura 8.23se puede apreciar la manera en la que los resultados de la simulación son presentados por la aplicación al concluir la simulación.

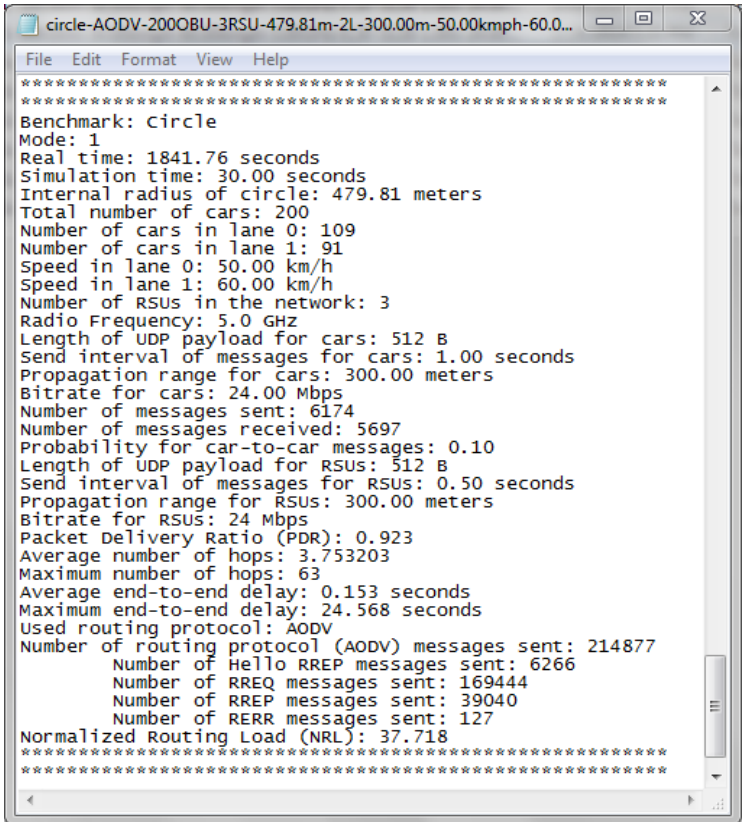


Figura 8.23: Visualización de los Resultados de una Simulación

8.2.7 Iteración 7: Mecanismo para la Configuración y Ejecución de Simulaciones sin el Uso del GUI de NCTUns

Para poder realizar una evaluación completa a un simulador es necesario realizar un alto número de simulaciones.El simulador NCTUns cuenta con un GUI a través del cual se pueden configurar todos los parámetros necesarios para las simulaciones. Sin embargo, cuando se desea realizar un conjunto de simulaciones en las cuales se desea especificar un gran conjunto de parámetros, hacerlo a través del GUI se convierte en una tarea repetitiva, aburrida y propensaa errores. Por esta razón se hizo necesario el desarrollo de un mecanismo para configurar y ejecutar las simulaciones sin que sea obligatorio el uso del GUI.

- **Fase de Diseño:** NCTUns provee una manera de ejecutar un caso de simulación sin tener que hacer uso del GUI. Sin embargo no provee una manera de crear estos casos de simulación sin hacer el uso del GUI. Para poder crear los casos de simulación sin uso del GUI es necesario conocer todos los archivos que son generados por el GUI cuando se crea un caso de simulación para poder desarrollar una aplicación que genere estos archivos de forma automática a partir de un conjunto de parámetros especificados por el usuario.

  Las aplicaciones *circleBenchmark.cpp* y *cityBenchmark.cpp*fueron desarrolladas con el fin de crear los casos de simulación para los benchmarks circular y de la ciudad, respectivamente. Estas aplicaciones generan todos los archivos necesarios para creación de los casos de simulación basándose en una lista de parámetros especificados en los archivos *circleBenchmark.h* y *cityBenchmark.h,* según sea el caso. En la Figura 8.24 se presenta un segmento del archivo circleBenchmark.h, donde se puede apreciar algunos de los parámetros utilizados para crear los casos de simulación del benchmark circular.

```
1  int mode = 1;
2  // Duration of the simulation, also known as simulation time (seconds)
3  double simTimeLimit = 30.0;
4  // Interval between update of position of cars (milliseconds)
5  double updateInterval = 250.0;
6  // Lane width (meters)
7  double laneWidth = 2.6;
8  // Number of Lanes
9  int numLanes = 2;
10 // Number of cars (for modes 1 and 3 only)
11 // Limitation: totalCars + totalRSUs <= 254
12 int totalCars = 250;
13 // Number of Road Side Units (RSUs)
14 // Limitation: totalCars + totalRSUs <= 254
15 int totalRSUs = 0;
16 // Distance between the streets and the RSUs
17 double streetRSUDistance = 5.0;
18 // Space around the working area in the GUI (meters)
19 double border = 10.0;
20 // Internal radius (in meters) of circular road (for modes 2 and 3 only)
21 double radiusCircle = 100.0 - 1.3;
22 // Speed of each lane (km/h). Set to negative values for vehicles
23 // that rotate counterclockwise
24 double speed[] = {50.0, 60.0};
25 // Distance (in seconds) between cars (for modes 1 and 2 only).
26 // The distance in meters will depend on the speed of the lane
27 double carDistance = 2.0;
28 // UDP payload length for cars (bytes)
29 int messageLengthCar = 512;
30 // UDP payload length for RSUs (bytes)
31 int messageLengthRSU = 512;
```

Figura 8.24: Segmento de Archivo circleBenchmark.h

Uno de los archivos de mayor importancia para la especificación de un caso de simulación en NCTUns es el archivo de descripción de la red cuya extensión es *.tcl*. En la Figura 8.25se muestra un segmento de un archivo de configuración *.tcl*. En este archivo se especifica la conectividad, los tipos y la configuración de las pilas de protocolo de los nodos además de otros parámetros como el tiempo de simulación.

```
Create Node 1 as MOBILE with name = MOBILE1
  Define port 1
    Module Interface : Node1_Interface_LINK_1
      Set Node1_Interface_LINK_1.guitag_ordinary = yes
      Set Node1_Interface_LINK_1.ip = 1.0.1.1
      Set Node1_Interface_LINK_1.netmask = 255.255.255.0

    Module AODV : Node1_AODV_LINK_1
      Set Node1_AODV_LINK_1.guitag_ordinary = yes
      Set Node1_AODV_LINK_1.HELLO_INTERVAL = 1000
      Set Node1_AODV_LINK_1.ALLOWED_HELLO_LOSS = 2
      Set Node1_AODV_LINK_1.ACTIVE_ROUTE_TIMEOUT = 3000
      Set Node1_AODV_LINK_1.DELETE_PERIOD = 3000
      Set Node1_AODV_LINK_1.NET_DIAMETER = 15
      Set Node1_AODV_LINK_1.NODE_TRAVERSAL_TIME = 40
      Set Node1_AODV_LINK_1.RREQ_RETRIES = 5
      Set Node1_AODV_LINK_1.RREQ_RATELIMIT = 10
      Set Node1_AODV_LINK_1.RERR_RATELIMIT = 10

    Module ARP : Node1_ARP_LINK_1
      Set Node1_ARP_LINK_1.guitag_ordinary = yes
      Set Node1_ARP_LINK_1.arpMode = RunARP
      Set Node1_ARP_LINK_1.flushInterval = 3000
      Set Node1_ARP_LINK_1.ArpTableFileName = Demo25_AODV.arp
```

Figura 8.25: Segmento del Archivo de Especificación de Caso de Simulación

Otro archivo de gran importancia para el desarrollo de los casos de simulación correspondientes a los benchmarks es el archivo de descripción de aplicaciones de tráfico cuya extensión es *.tfc*. En este archivo se especifican que aplicaciones a nivel de usuario serán utilizadas por los nodos durante la simulación. Este archivo es utilizado para indicar que cada nodo debe correr la aplicación generadora de tráfico (descrita en la Sección 8.2.1) o para indicar que el nodo encargado del movimiento en el benchmark circular debe correr la aplicación *circleController* (descrita en la Sección 8.2.2) entre otras aplicaciones que deben ser ejecutadas por los nodos durante la simulación.

Adicionalmente fue necesario el desarrollo de un script para unir los procesos de desarrollo de un caso de simulación, ejecución de un caso de simulación y los procesos de cálculo y presentación de los resultados. El script *runNCTUns.sh*fue desarrollado para que una vez especificados los parámetros de la simulación, se pueda correr la simulación haciendo invisible para el usuario el hecho de que por cada caso de simulación se crean archivos de simulación para luego correr la simulación especificada en dichos archivos y que finalmente otra aplicación se encargue del cálculo y la presentación de los resultados.

En la fase de codificación se explica cómo fueron desarrolladas las aplicaciones *circleBenchmark.cpp* y *cityBenchmark.cpp* y el script *runNCTUns* para el desarrollo y ejecución de casos de simulación sin el uso del GUI.

- **Fase de Codificación:** La fase de codificación consta de 3 partes:

  La primera parte está enfocadaen el desarrollo de la aplicación *circleBenchmark.cpp*. Como se puede apreciar por la extensión del archivo (*.cpp*), esta aplicación fue desarrollada utilizando el lenguaje de programación C++. El objetivo de esta aplicación consiste básicamente en la validación de los parámetros de entrada y la generación de todos los archivos necesarios para la descripción de un caso de simulación correspondiente al benchmark circular.

  LaFigura 8.26 muestra un segmento del código de la aplicación, donde se puede apreciar que principalmente la aplicación se basa en la escritura de archivos de texto plano que describen un área específica del caso de simulación basándose en los parámetros definidos en *circleBenchmark.h*.

```
for(int carIndex=carsPlaced;carIndex<=carsLeft;carIndex++)
{
    startAngle= 360.0/numCarsLane[lane]*aux;
    x0=cxcy+r*cos(startAngle*M_PI/180.0);
    y0=cxcy+r*sin(startAngle*M_PI/180.0);
    placeNode(mdt,carIndex,x0,y0,opModeCar);
    fprintf(tfc, "$node_(%d) %f %f UDPForCarApp %d %d %d %f %f %f %d %d %d\n",
        carIndex,0.0,realSimTimeLimit,totalCars,totalRSUs,messageLengthCar,
        sendIntervalCar,car2CarMsgPercent,simTimeLimit,showIncomingPackets,
        showOutgoingPackets,showTimeofNextMessage);
    fprintf(ndt,"%d 802.11(%c)\n",carIndex,opModeCar);
    fprintf(sce,"$node_(%d) set %f %f 0.0 0.0 0.000000 0.000000\n",carIndex,x0,y0);
    fprintf(srt,"#node%d\n",carIndex);
    fprintf(srt,"route add -net 1.%d.1.0/24 tun%d\n\n",carIndex,carIndex);

    carsPlaced++;
    aux++;
}
```

Figura 8.26: Segmento de Código de la Aplicación circleBenchmark

La segunda parte consta en el desarrollo de la aplicación *cityBenchmark*. Esta aplicación es muy similar a la aplicación *circleBenchmark* ya que ambas se encargan de desarrollar los archivos necesarios para la descripción de un caso de simulación. Sin embargo existen parámetros,como por ejemplo el radio del círculo, que solo tienen validez para un solo benchmark ya sea el benchmark de la ciudad o el benchmark circular. Una de las diferencias en las aplicaciones es que el benchmark de la ciudad no requiere la creación de un nodo dedicado al movimiento de los vehículos. Otra diferencia es que en el benchmark de la ciudad, no se necesita calcular la posición de los vehículos para definirla en el archivo correspondiente ya que ésta es determinada por los archivos de trazas que especificarán el movimiento de los vehículos.

La tercera parte consiste en el desarrollo del script *runNCTUns.sh* el cual tiene como función unir los procesos de creación del caso de simulación, ejecución del benchmark y ejecución del programa *getResults* (descrito en la Sección 8.2.6) que calcula y muestra los resultados de la simulación realizada. Este scriptconsta de un conjunto de comandos ejecutados de manera secuencial que comienzan con la compilación de los códigos *circleBenchmark.cpp*, *cityBenchmark.cpp* y *getResults.cpp* para la actualización de los parámetros especificados en los archivos *circleBenchmark.h* y *cityBenchmark.h.* Luego la ejecución de las

aplicaciones *circleBenchmark*o*cityBenchmark*según el caso para la generación de los archivos necesarios para la descripción del caso de simulación. Seguidamente se realiza la ejecución de la simulación y finalmente la ejecución de la aplicación *getResults* para generación de los resultados de la simulación.

- **Fase de Pruebas:** Las pruebas de esta iteración consistieron en modificar los parámetros de la simulación para crear casos de simulación a través de las aplicaciones desarrolladas y por otro lado crear los mismos casos de simulación a través del GUI. Seguidamente se compararon los archivos generados por el GUI con los archivos generados por las aplicaciones. No deberían existir diferencias entre estos conjuntos de archivos ya que ambos describen el mismo caso de simulación.

  La prueba del script se realizó de manera similar, los comandos fueron ejecutados uno por uno manualmente en la consola de comandos y se comparó el resultado con la ejecución del script que ordena la ejecución de dichos comandos de manera secuencial.

9. Pruebas y Análisis de los Resultados

En este capítulo se presentan las pruebas realizadas y los resultados obtenidos con los benchmarks propuestos para evaluar el desempeño del simulador NCTUns en redes vehiculares.

9.1 Resultados de las Pruebas Realizadas en el Benchmark Circular

El objetivo de estas pruebas es evaluar el comportamiento de NCTUns en un escenario con una carretera circular en la cual se forma una red vehicular y la cual puede o no incluir RSUs. Los valores por defecto para los parámetros más relevantes para este benchmark se muestran en la Tabla 9.1.

Parámetro	Valor por Defecto
Tiempo de duración de la simulación	30 segundos
Número de vehículos	200 vehículos
Radio del círculo interno de la carretera circular	479.81m
Distancia entre vehículos	2 segundos
Número de canales	2 canales
Anchura de los canales	2.6m
Velocidad de los vehículos en cada canal	50 km/h y 60 km/h
Estándar WiFi	802.11a
Radio frecuencia	5 GHz
Rango de propagación de la señal	300m
Tamaño del payload de UDP	512 bytes
Bitrate	24 Mbps
Intervalo de envío de mensajes de los vehículos	1.0 segundos
Intervalo de envío de mensajes de los RSUs	0.5 segundos
Probabilidad de envío de mensajes vehículo-a-vehículo	50%
Protocolos de enrutamiento	AODV, ADV y DSDV

Tabla 9.1: Valores por Defecto de los Parámetros de las Simulaciones Correspondientes al Benchmark Circular

En las Secciones 9.1.1, 9.1.2 y 9.1.3 se describen pruebas realizadas enlas cuales no se incluyeronRSUs en lassimulaciones. Las Secciones 9.1.4 y9.1.5 describen pruebas que síincluyeron RSUs en las simulaciones.

9.1.1 Variación del Tamaño del Payload UDP de los Mensajes Enviados por los Vehículos

El parámetro a variar en esta prueba es el tamaño del *payload* UDP de los mensajes enviados por los vehículos a otros vehículos ya que en esta prueba no fueron incluidos RSUs.Las simulaciones se realizaron utilizando los siguientes valores: 10, 50, 100, 250, 500 y 1000 bytes para la carga útil UDP. Se utilizaron los protocolos de enrutamiento: AODV, ADV y DSDV.

En la Figura 9.1 se observa como la variación del tamaño de la carga útil UDP no tiene efecto sobre el número de saltos promedio. Al comparar los protocolos de enrutamiento se observa que DSDV tiene valores inferiores a aquellos exhibidos por los protocolos AODV y ADV. Sin embargo, al observar los valores presentados en la Figura 9.2, correspondientes al PDR, se nota que el PDR tiene valores inferiores para el protocolo DSDV en comparación con los protocolos AODV y ADV. Esto significa que el número de saltos promedio es inferior para el protocolo DSDV debido a que solo es capaz de realizar

enrutamiento satisfactoriamente para aquellos mensajes que están a cuando mucho 2 saltos de distancia.

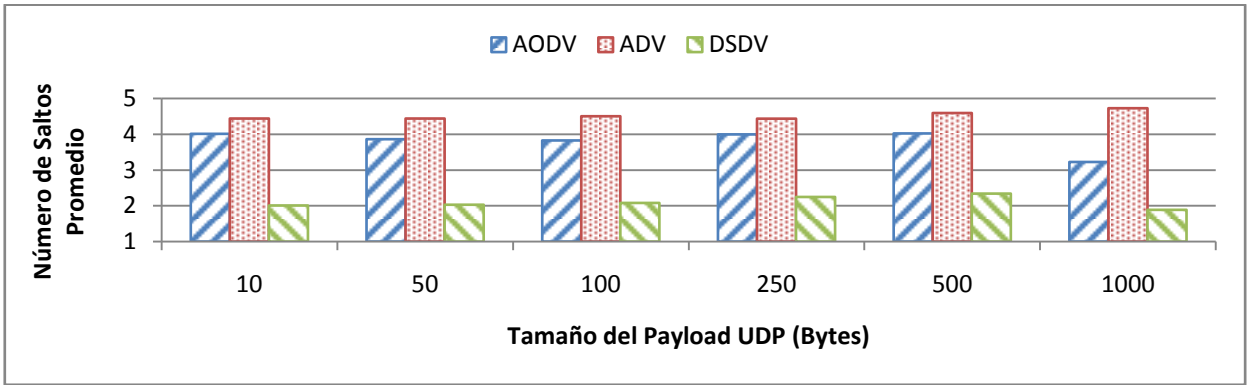


Figura 9.1: Número de Saltos Promedio para el Benchmark Circular en Función del Tamaño del Payload UDP

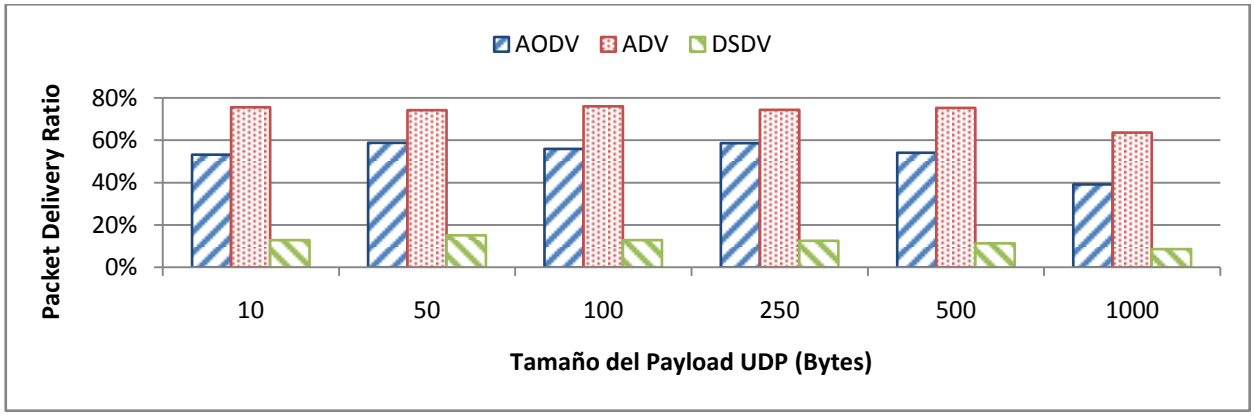


Figura 9.2: Packet Delivery Ratio para el Benchmark Circular en Función del Tamaño del Payload UDP

En la Figura 9.3 se puede observar que los valores correspondientes al End-to-End Delay se mantienen constante para los distintos tamaños del payload UDP con la excepción de tamaño de paquete de 1000 bytes, esto se debe al aumento en los tiempos de serialización y en el uso del medio por parte de los nodos para transmitir paquetes de mayor tamaño. Al comparar los protocolos entre sí,se observa que ADV tiene valores superiores a los protocolos AODV y DSDV.

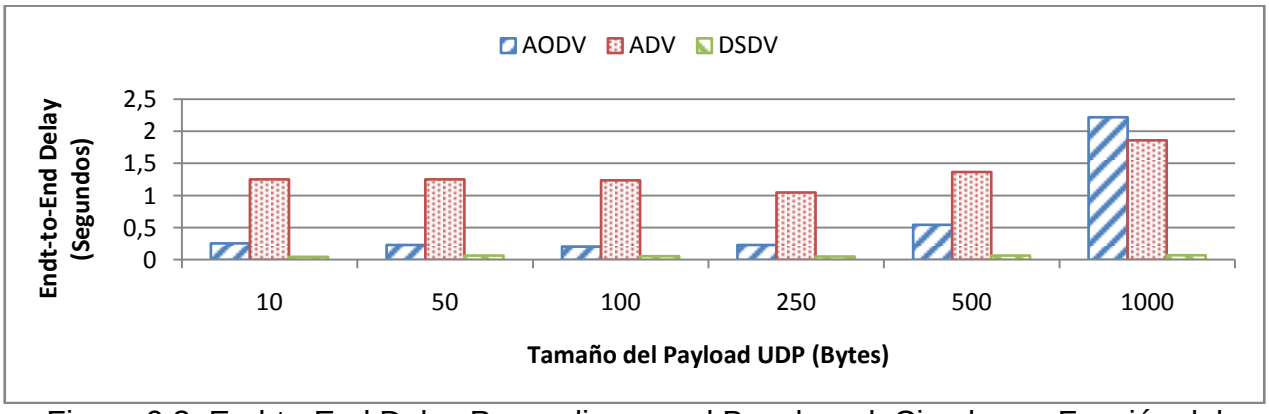


Figura 9.3: End-to-End Delay Promedio para el Benchmark Circular en Función del Tamaño del Payload UDP

La Figura 9.4muestra los valores correspondientes al NRL. Como es de esperarse, los valores se mantienen relativamente constantes al variar el tamaño del payload UDP. Lo mismo se observa en la Figura 9.5 correspondiente al número de mensajes enviados por el protocolo de enrutamiento. Al comparar los protocolos de enrutamiento se encuentra que el protocolo DSDV requiere el envío de un gran número de mensajes de control, en algunos casos hasta 7 veces más que los protocolos ADV y AODV para mantener las rutas, esto se debe a que DSDV es un protocolo proactivo por lo queno intenta descubrir la ruta al momento de enviar un mensaje sino que siempre intenta mantener las rutas actualizadasa todos los destinos.Dada la alta movilidad de los nodos en la red planteada, las rutas cambian con mucha frecuencia por lo cual este protocolo parece no adaptarse a las necesidades de las redes vehiculares. Finalmente, en la Figura 9.6 se puede observar que el tiempo requerido para efectuar las simulaciones tampoco varía en función del tamaño del payload UDP. Al comparar con la Figura 9.5 se puede señalar que los protocolos que envían un mayor número de mensajes requieren más tiempo para culminar las simulaciones.

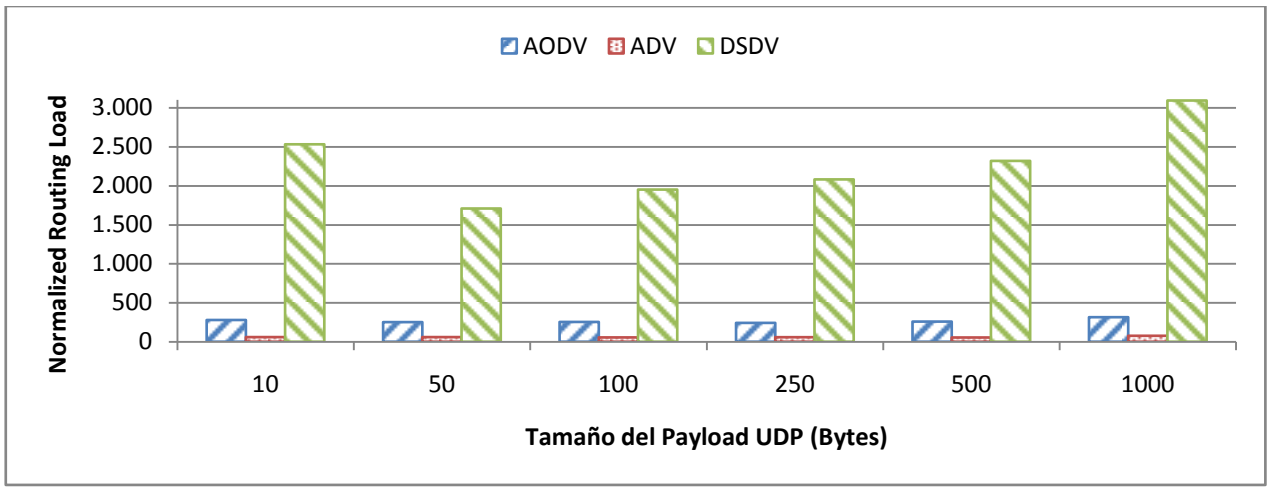


Figura 9.4: Normalized Routing Load para el Benchmark Circular en Función del Tamaño del Payload UDP

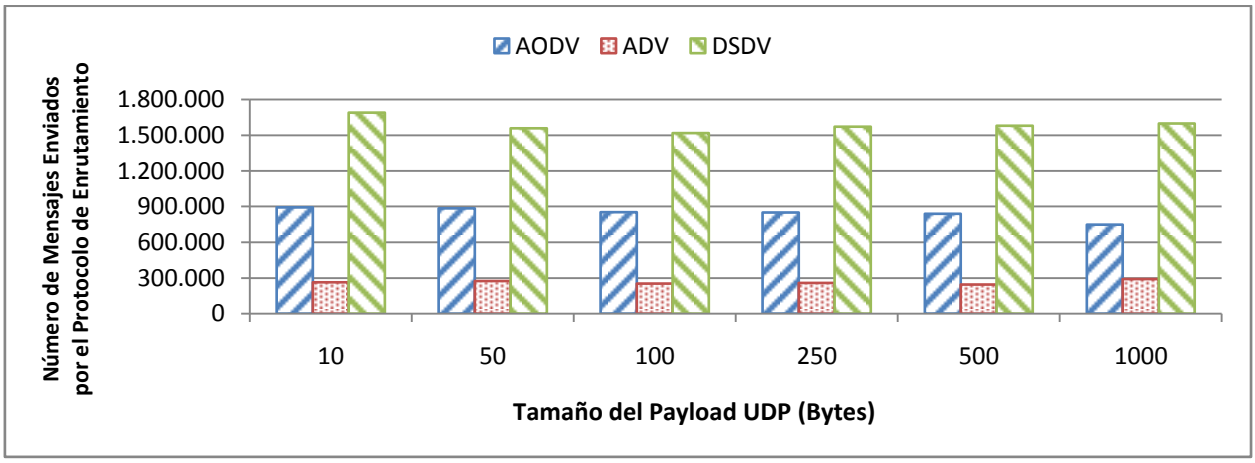


Figura 9.5: Número de Mensajes Enviados por Protocolo de Enrutamiento para el Benchmark Circular en Función del Tamaño del Payload UDP

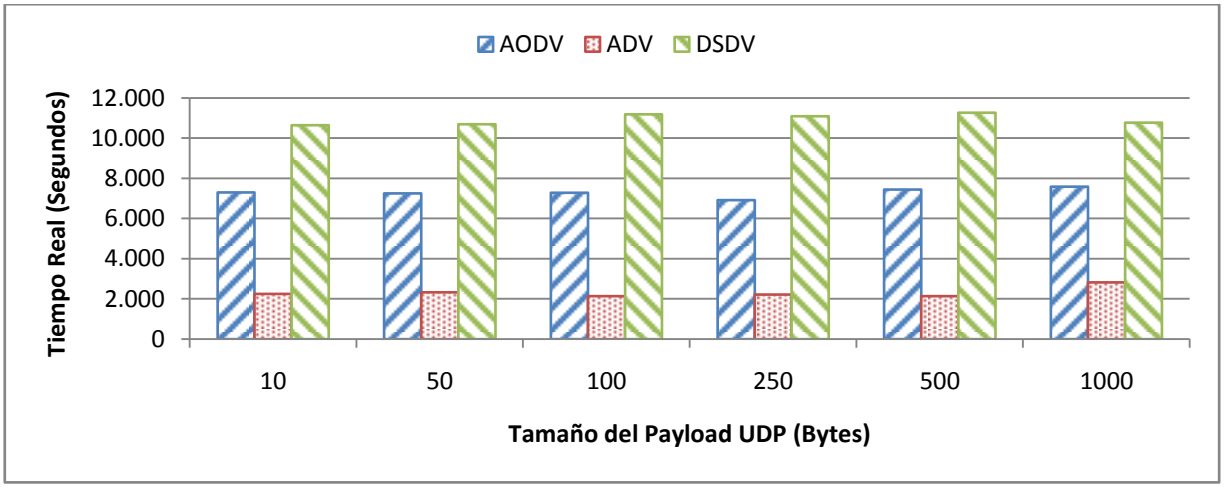


Figura 9.6: Tiempo Real de la Simulación para el Benchmark Circular en Funcióndel Tamaño del Payload UDP

9.1.2 Variación del Rango de Propagación de las Señales Emitidas por los Vehículos

Esta prueba consiste en observar los efectos que tiene la variación del rango de propagación de las señales emitidas por los vehículos en la red vehicular planteada por el benchmark circular. Los valores utilizados para el rango de propagación en las simulaciones son: 40m, 50m, 60m, 70m, 80m, 90m, 100m, 200m, 300m, 400m y 500m. En esta prueba no se incluyen RSUs, los valores de los otros parámetros se encuentran definidos en la Tabla 9.1.

En la Figura 9.7 se observa quelos valores de PDR van en aumento a medida que el rango de propagación crece, hasta llegar a un valordel rango de propagación alrededor de 100m, donde el valor del PDR disminuye a medida que el rango de propagación aumenta.

El comportamiento mostrado por los protocolos es esperado ya que para valores superiores de rango de propagación más nodos comparten el espacio a través del cual transmiten, lo cual aumenta el número de colisiones disminuyendo así el número de mensajes recibidos satisfactoriamente. Por otro lado, para valores de rango de propagación inferiores,la probabilidad de que un nodo conozca la ruta al destino es menor, requiriendo el envío de un mayor número de mensajes de control por parte de los protocolos de enrutamiento reactivos. Otro efecto ocasionado por un rango de propagación inferior es que el número de saltos requerido para enviar cada mensaje es mayor como se puede observar en la Figura 9.8. En dicha figura, se observa claramente como el número de saltos disminuye mientras el rango de propagación aumenta, esto se debe a que mientras mayor sea el rango de propagación mayor es el número de vehículos que cada nodo alcanza en un salto.

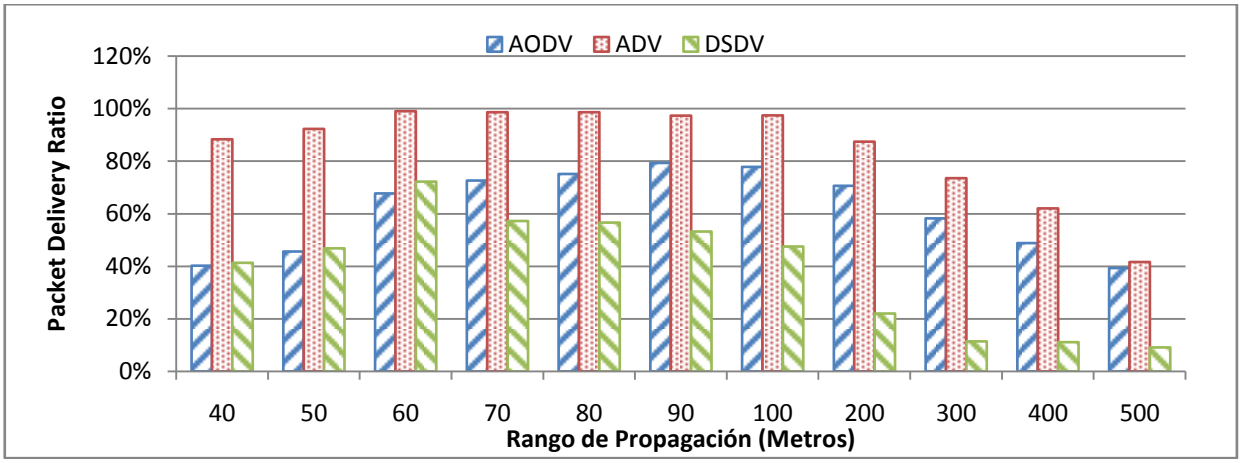


Figura 9.7: Packet Delivery Ratio para el Benchmark Circular en Función del Rango de Propagación

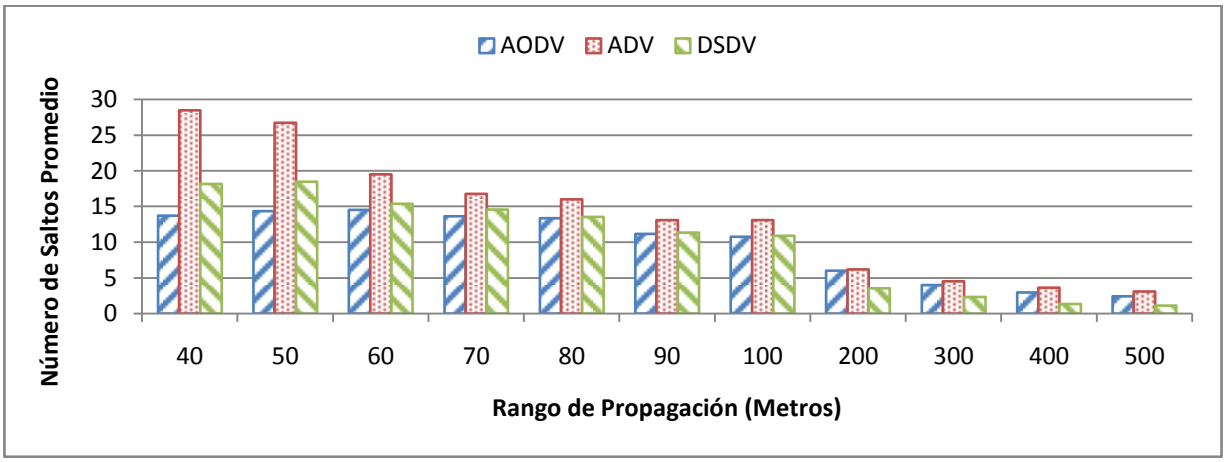


Figura 9.8: Número de Saltos para el Benchmark Circular en Función del Rango de Propagación

En la Figura 9.9 se observa el efecto del valor del rango de propagación sobre el *end-to-end delay*.Se puede notar que en el caso de los protocolos ADV y AODV se va produciendo un aumento en el retardo a medida que aumenta el rango de propagación a partir de 100m en adelante, esto se debe a que, como se mencionó anteriormente, al aumentar el rango de propagación aumenta el número de colisiones y existe más competencia entrevehículos por el medio, ya que el espacio es compartido por un mayor número de nodos a medida que aumenta el rango de propagación. En el caso del protocolo DSDV no existe aumento en el end-to-end delay a medida que aumenta el rango de propagación, al observar la Figura 9.7, se puede notar que el PDR presentado por el protocolo DSDV es de alrededor de 20% para rangos de propagación superiores a los 100m. Esto significa que los valores presentados en la Figura 9.9 correspondientes al end-to-end delay para el protocolo DSDV a partir de valores de rango de propagación superiores a los 100m son solo del 20% de los paquetes que se intentó enviar durante la simulación.

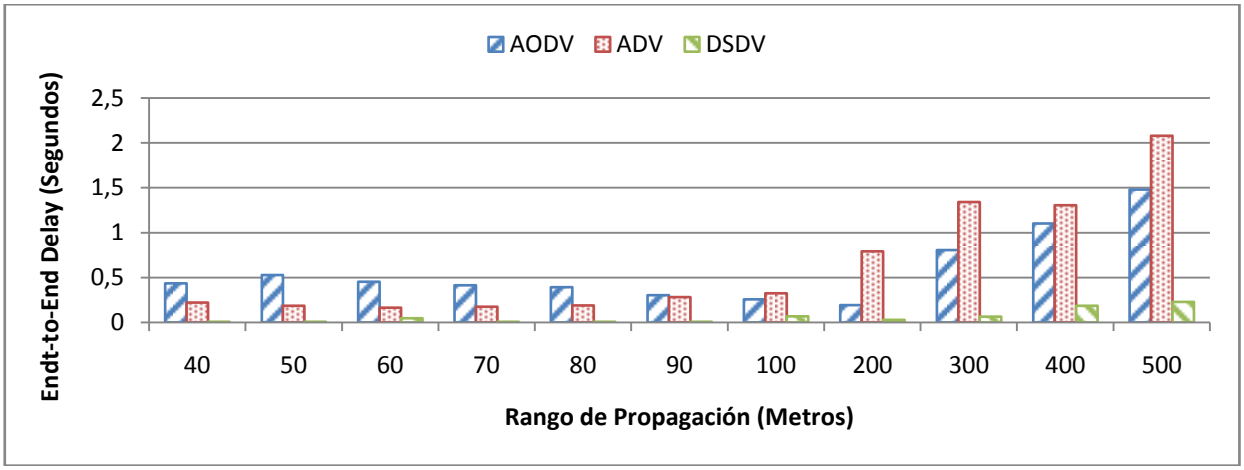


Figura 9.9: End-to-End Delay para el Benchmark Circular en Función del Rango de Propagación

LaFigura 9.10muestra los efectos del valor del rango de propagación en el NRL.Como se puede apreciar, el NRL se mantiene constante para los protocolos AODV y ADV, sin embargo para el protocolo DSDV el NRL aumenta significativamente a partir de valores de rango de propagación mayores a los 100m. Al observar la Figura 9.7 se nota que el protocolo DSDV sufre una disminución del PDR a partir de rangos de propagación mayores a 100m lo cual significa que el número de paquetes recibidos por los vehículos disminuye considerablemente, llegando este a ser inferior al 20% de los paquetes enviados. Esto provoca un aumento en el NRL a pesar de que DSDV es un protocolo proactivo.

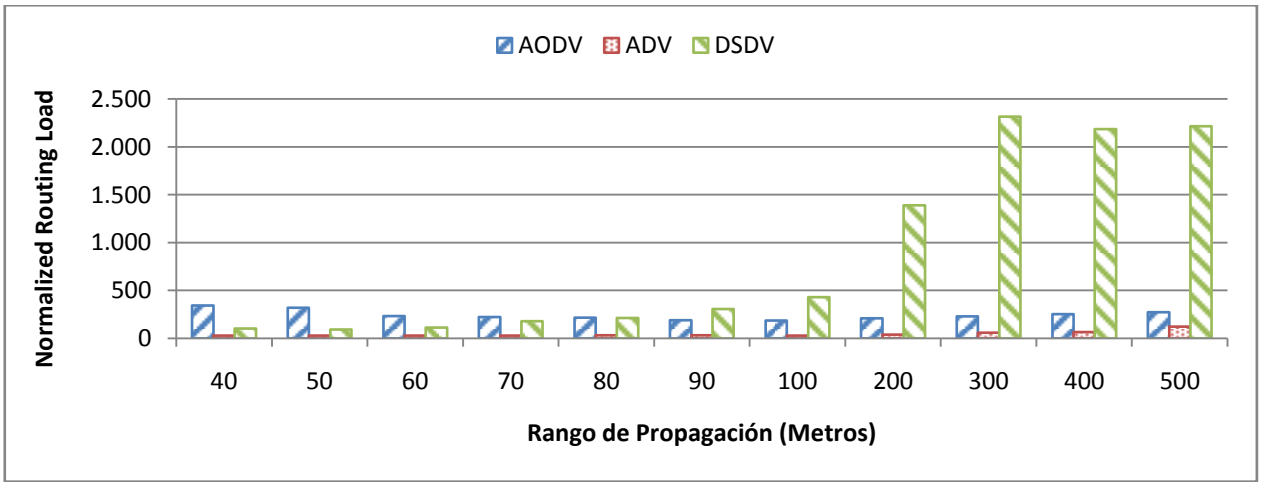


Figura 9.10: Normalized Routing Load para el Benchmark Circular en Función del Rango de Propagación

En la Figura 9.11se puede apreciar el número de mensajes enviados por los distintos protocolos de enrutamiento. De manera general el protocolo ADV presenta valores constantes a rededor de 200000 mensajes de control mientras que el protocolo AODV presenta valores que se mantienen alrededor de los 800000 mensajes de control para los distintos valores de rango de propagación. El protocolo DSDV muestra valores superiores a los protocolos ADV y AODV cuando el rango de propagación es superior a los 100msimilar al comportamiento en las otras pruebas, sin embargo para valores de rango de propagación menores a 100m el número de mensajes enviados por el protocolo DSDV

va en aumento desde los 200000 mensajes hasta 800000 mensajes de control enviados durante la simulación.

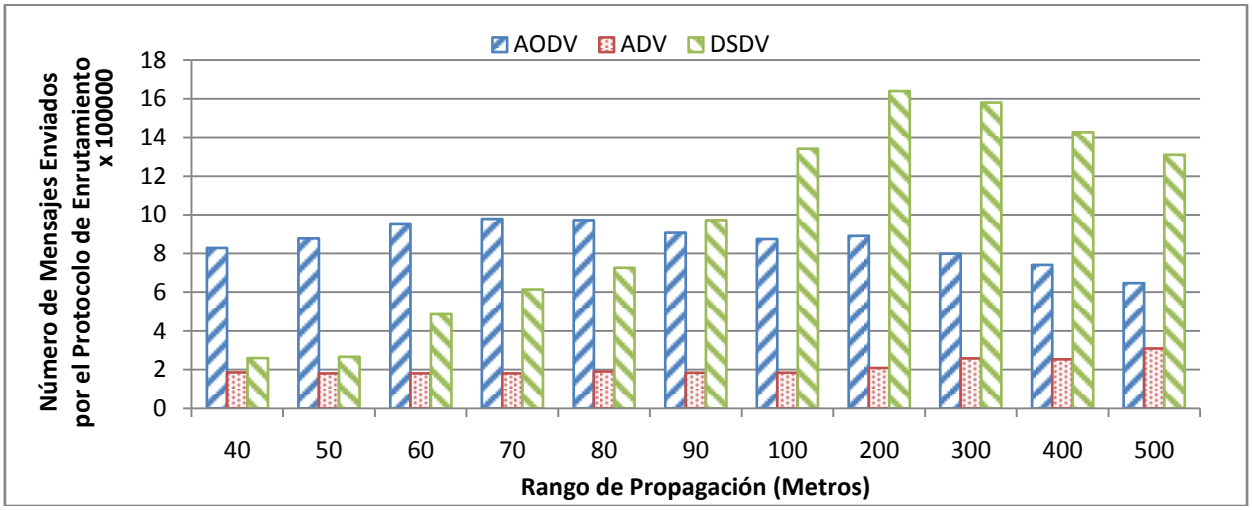


Figura 9.11: Número de Mensajes Enviados por el Protocolo de Enrutamiento parael Benchmark Circular en Función del Rango de Propagación

Por último, en la Figura 9.12 se puede observar como el tiempo de simulación aumenta a medida que aumenta el rango de propagación, esto se debe a que al aumentar el rango de propagación cada mensaje enviado es procesado por más nodos ya que el medio es compartido entre más nodos y también al hecho de que ocurren más colisiones.

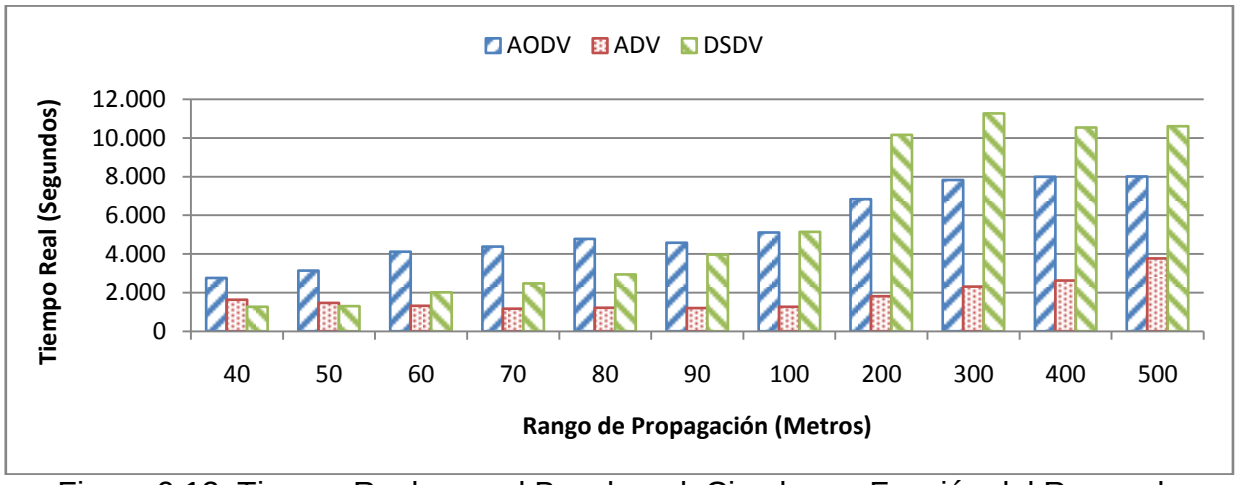


Figura 9.12: Tiempo Real para el Benchmark Circular en Función del Rango de Propagación

9.1.3 Variación del Número de Vehículos Incluidos en la Simulación

El objetivo de esta prueba es evaluar el efecto que tiene la cantidad de vehículos sobre el desempeño del simulador y el desempeño de los protocolos de enrutamiento. Para estas simulaciones se utilizaron los valores por defecto presentados en la Tabla 9.1 y solo se variaron el número de vehículos y el radio interno de la carretera circular. Los valores para el número de vehículos son: 50, 100, 150, 200 y 250. Los valores en metros para el radio interno de la carretera circular son: 118.09, 238.66, 359.23, 479.81 y 600.38. Los valores del radio del círculo interno fueron calculados automáticamente por la aplicación *circleBenchmark*(Sección 8.2.2) para garantizar una distancia entre vehículos de 2 segundos.

Uno de los efectos que tiene el aumento de vehículos sobre el desempeño de la red es el aumento del número de saltos como se puede observar en la Figura 9.13. Este aumento se debe a que al haber más vehículos en la simulación y al mantener la misma distancia entre vehículos, se trabaja con una red circular de mayor radio y por ende aumenta la probabilidad de que el vehículo receptor este a una mayor distancia del vehículo emisor.

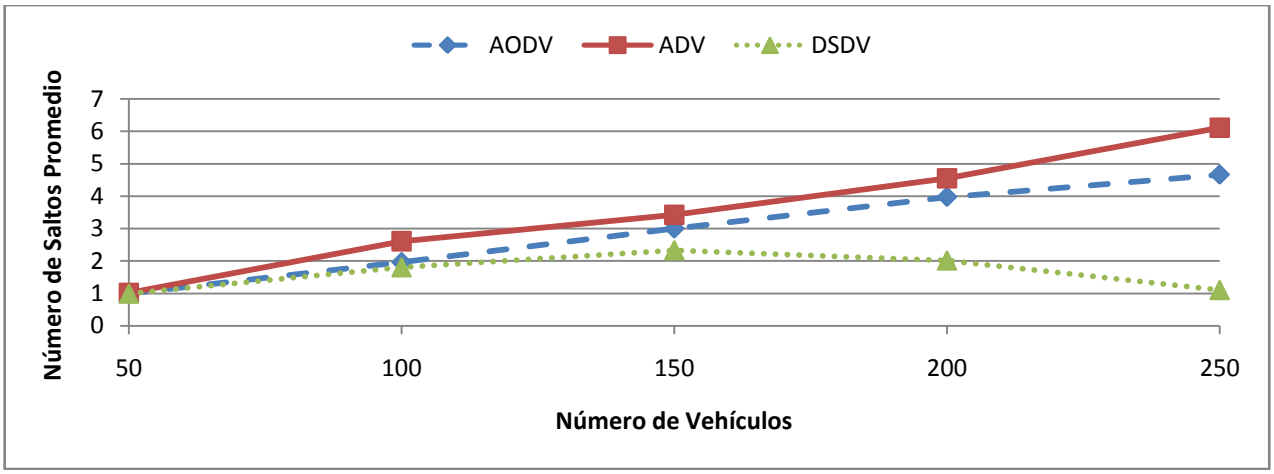


Figura 9.13: Número de Saltos Promedio para el Benchmark Circular en Función del Número de Vehículos

En laFigura 9.14se puede observar como el aumento de vehículos produce un efecto negativo sobre la red vehicular disminuyendo el PDR a medida que el número de vehículos aumenta. Esto se debe al aumento del número de saltos y que existe mayor competencia por el medio mientras mayor es el número de vehículos. Otro efecto que tiene el número de vehículos en la red es el aumento del end-to-end delay como se observa en la Figura 9.15. El aumento del end-to-end delay se debe al aumento del número de saltos, ya que el mensaje debe ser procesado en una cantidad mayor de nodos y ademásse debe a que los protocolos de enrutamiento deben enviar un mayor número de mensajes de control como se puede apreciar en la Figura 9.16

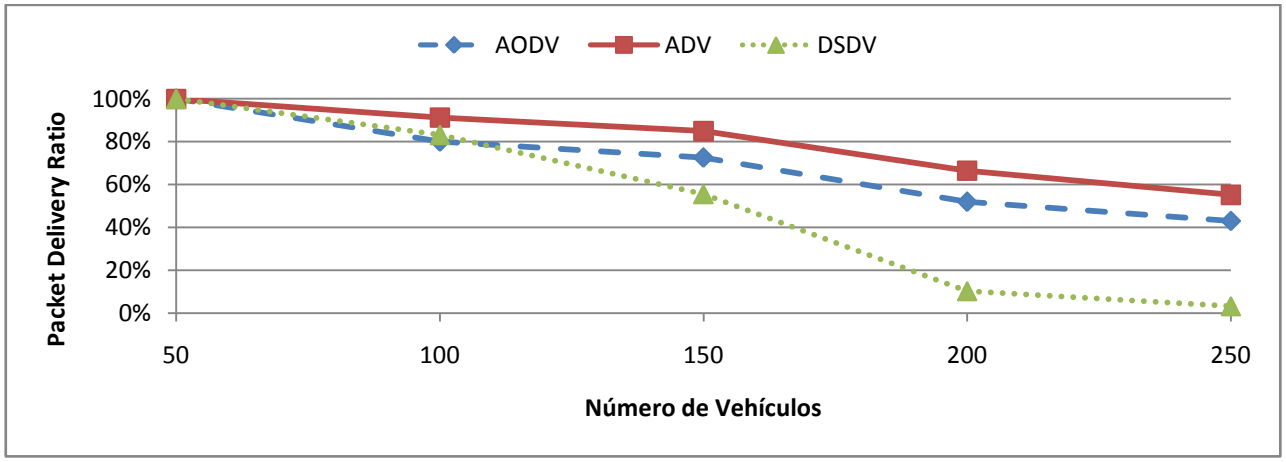


Figura 9.14: Packet Delivery Ratio para el Benchmark Circular en Función del Número de Vehículos

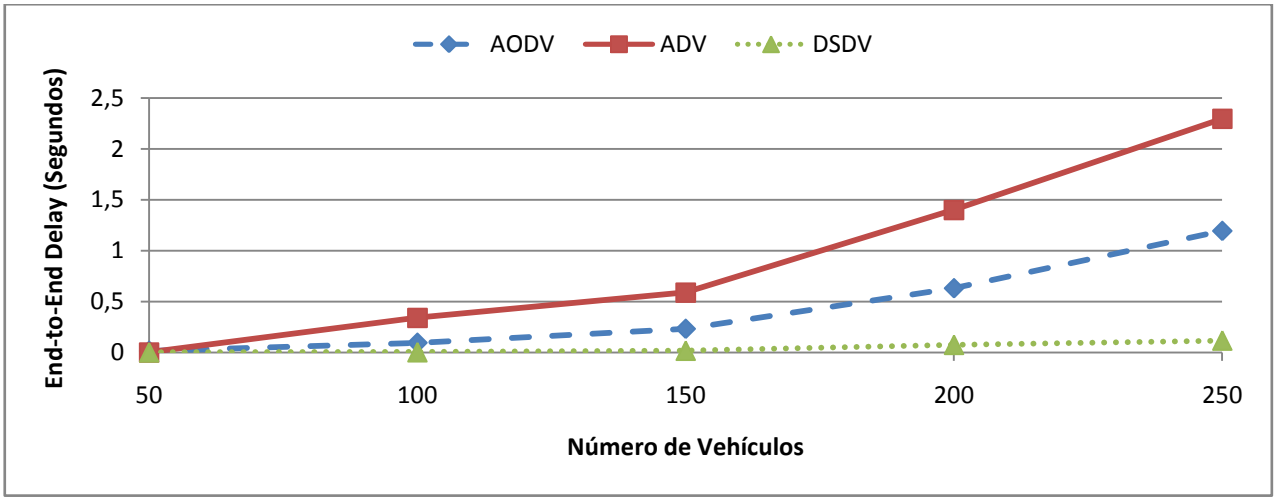


Figura 9.15: End-to-End Delay para el Benchmark Circular en Función del Número de Vehículos

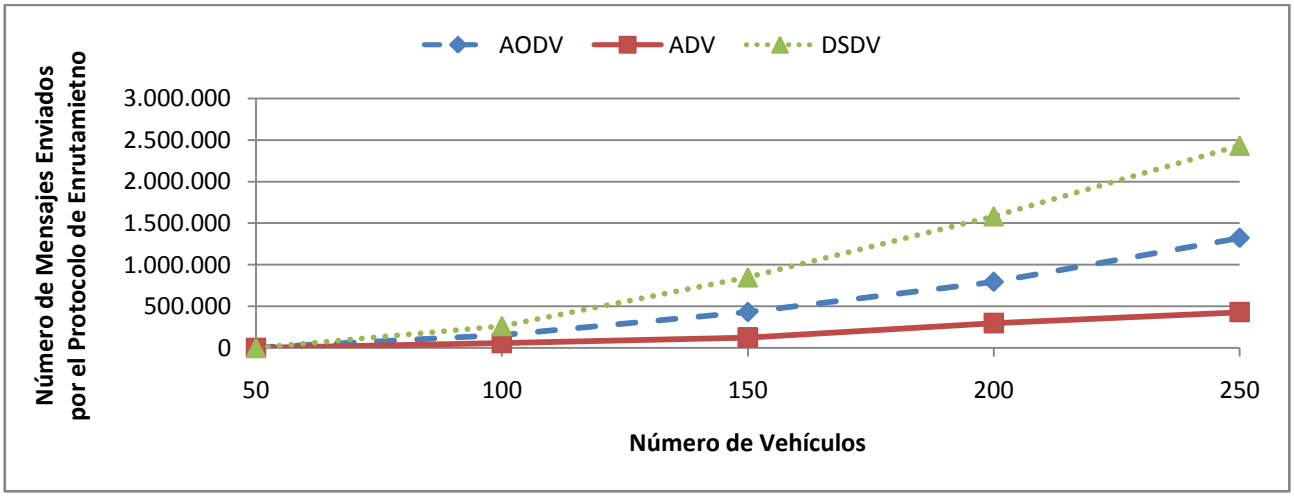


Figura 9.16: Número de Mensajes Enviados por el Protocolo de Enrutamiento para el Benchmark Circular en Función del Número de Vehículos

En la Figura 9.17 se observa como el NRL aumenta a medida que aumenta el número de vehículos. En el caso de los protocolos AODV y ADV el aumento es mínimo en comparación con el protocolo DSDV esto se debe a que el protocolo DSDV es un protocolo proactivo, por lo cual cada nodo establece rutas para todos los demás nodos.Otra razón para el aumento del NRL y la diferencia entre los protocolos es que el número de paquetes recibidos por los vehículos es considerablemente inferior en el caso de DSDV como se puede observar en la Figura 9.14correspondiente al PDR.

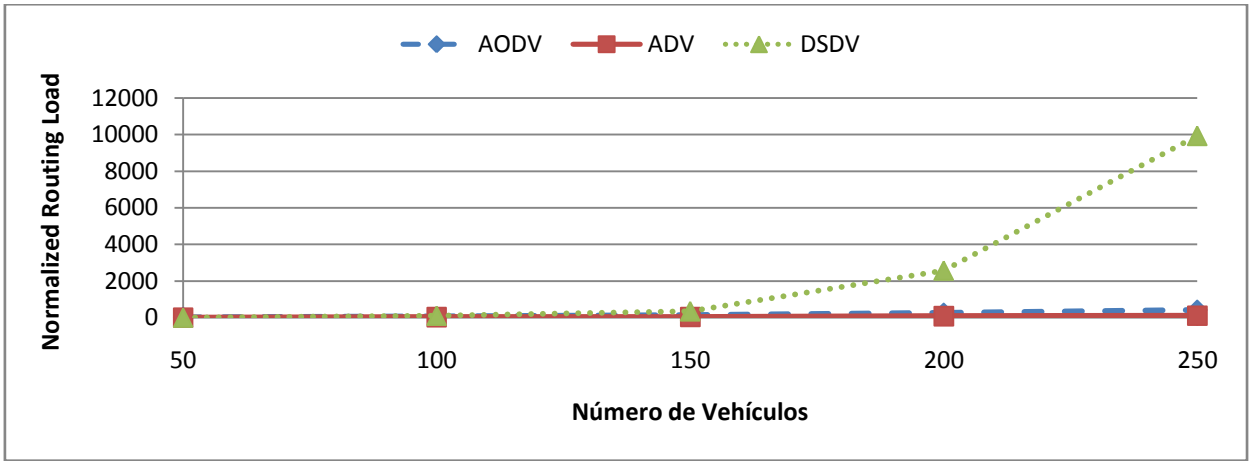


Figura 9.17: Normalized Routing Load para el Benchmark Circular en Función del Número de Vehículos

En laFigura 9.18se puede observar que a medida que aumenta el número de vehículos aumenta el tiempo real de simulación. Esto se debe a que el aumento del número de vehículos implica un aumento en el número de nodos que deben ser movidos por la aplicación *circleController*durante la simulación. Adicionalmente mientras más vehículos existen en la simulación mayor es el número de mensajes UDP enviados por los vehículos y mensajes de control que deben ser enviados por el protocolo de enrutamiento. Estas son también las razones por las cuales aumenta el consumo de memoria a medida que aumenta el número de vehículos como se puede observar en la Figura 9.19.

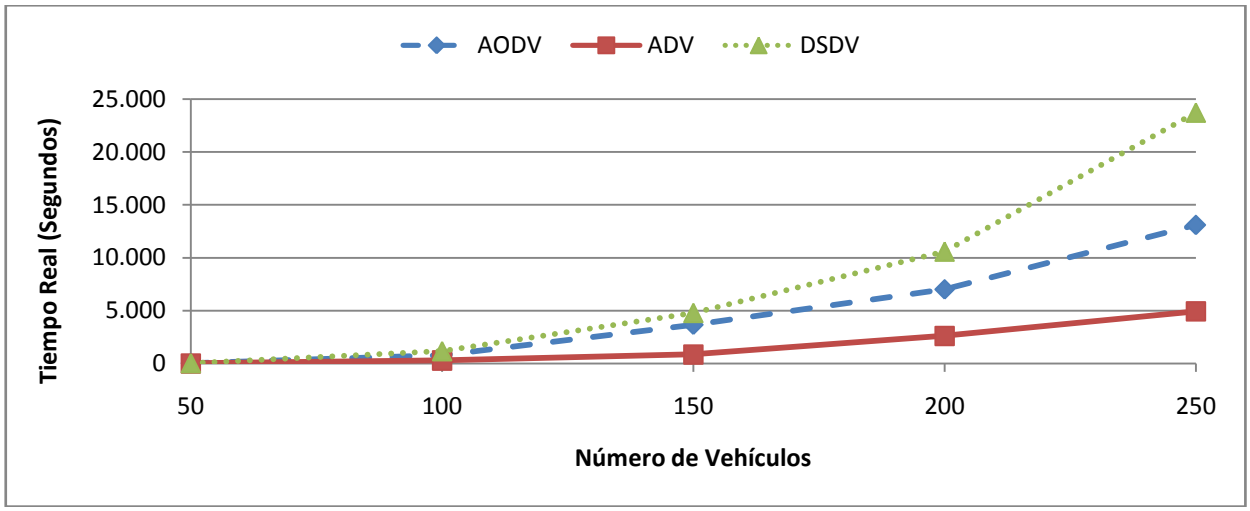


Figura 9.18: Tiempo Real para el Benchmark Circular en Función del Número de Vehículos

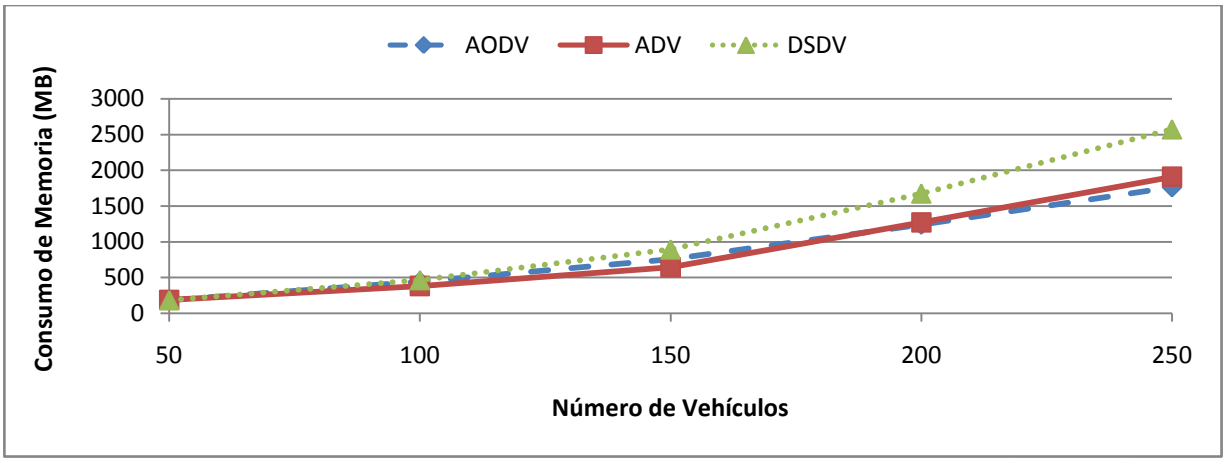


Figura 9.19: Consumo de Memoria para el Benchmark Circular en Función del Número de Vehículos

Por último, en la Figura 9.20se observa el uso del CPU en función del número de vehículos. El uso del CPU es inferior a 90% solo cuando el número de vehículos es igual a 50, sin embargo, a partir de 100 vehículos en adelante el uso del CPU se encuentra entre el 97% y 100%.

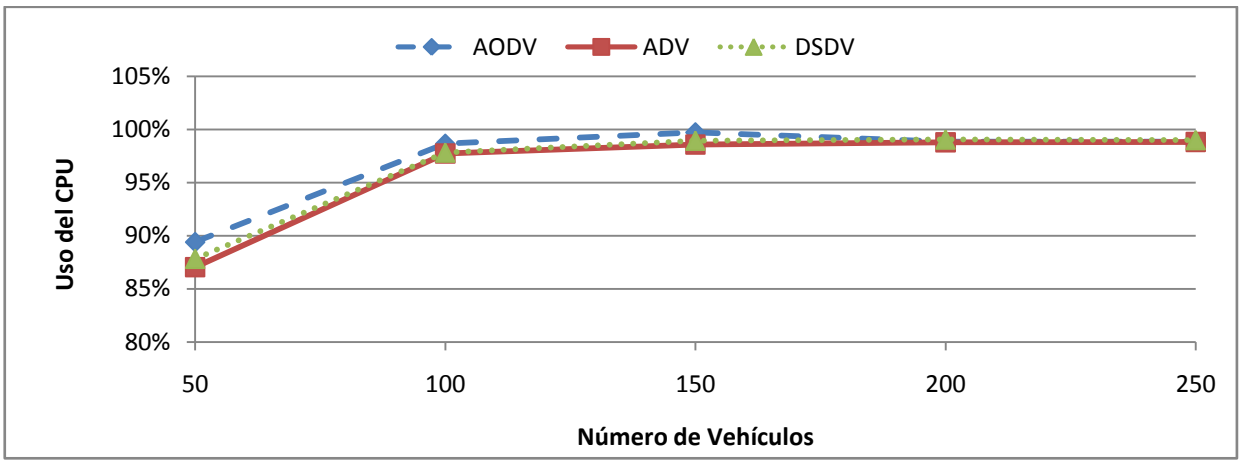


Figura 9.20: Uso del CPU para el Benchmark Circular en Función del Número de Vehículos

9.1.4 Variación de la Probabilidad de Envío de Mensajes Vehículo-a-Vehículo

El parámetro a variar en esta prueba es la probabilidad de enviar un mensaje de un vehículo a otro (car-to-car messages). Mientras la probabilidad sea más baja hay más chance de que el vehículo envíe el mensaje a un RSU, en caso contrario hay más chance que el destinatario sea otro vehículo. Los RSUs solo envían mensajes a vehículos. De esta manera se realizaron simulaciones con las siguientes probabilidades: 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8 y 0.9. En estas pruebas se utilizaron 3 RSUs distribuidos alrededor de la carretera circular.

En la Figura 9.21 se observa cómo afecta la variación de este parámetro en el PDR en los tres protocolos. Los protocolos ADV y DSDV mantienen un PDR constante, siendo el rendimiento del protocolo ADV superior al del protocolo DSDV al mantener un PDR superior al 60% mientras que el protocolo DSDV no logra superar el 20% en cuanto al

PDR se refiere.En el caso del protocolo AODV,el PDR disminuye levemente a medida que aumenta la probabilidad de envío de mensajes vehículo-a-vehículo.

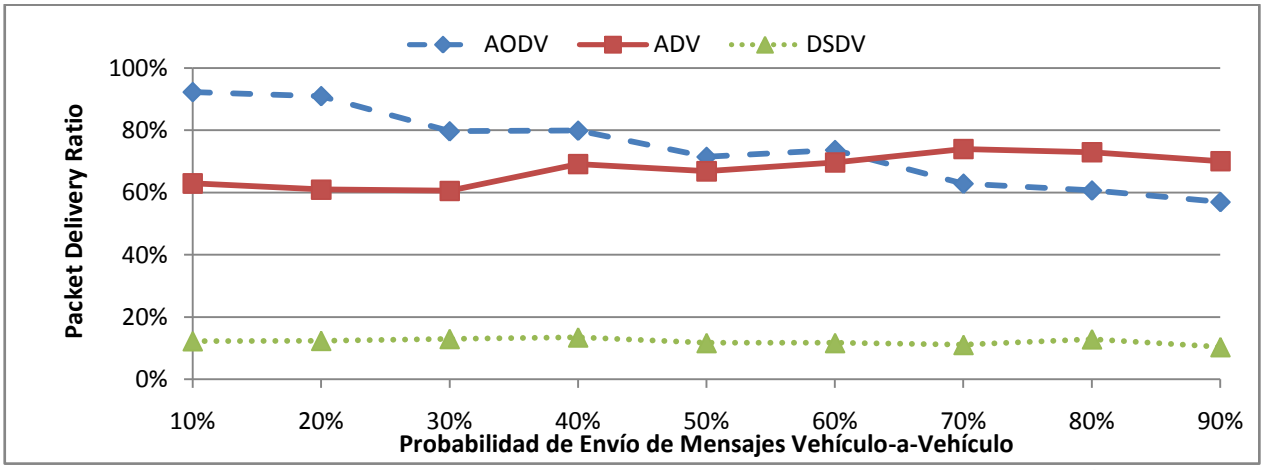


Figura 9.21: Packet Delivery Ratio para el Benchmark Circular en Función de la Probabilidad de Envío de Mensajes Vehículo-a-Vehículo

Al observar la Figura 9.22 correspondiente al *end-to-end delay*, se nota como ocurre el mismo fenómeno, los protocolos ADV y DSDV mantienen valores constantes, aunque en este caso el protocolo DSDV tiene mejor rendimiento que el protocolo ADV. Por otro lado, el protocolo AODV sufre un aumento del *end-to-end delay* a medida que aumenta la probabilidad de envío de mensajes vehículo-a-vehículo. En la Figura 9.23 se observa como el Número de Mensajes enviado por el protocolo AODV aumenta a medida que aumenta la probabilidad de envío de mensajes vehículo-a-vehículo razón por la cual existe unadisminución en el PDR y un aumento en el *end-to-end delay* para dicho protocolo. El aumento del número de mensajes enviados por el protocolo AODV puede deberse al hecho de que mientras más alta sea la probabilidad de envío de mensajes vehículo-a-vehículo mayor es la probabilidad de que el emisor no conozca la ruta hacia el receptor. Este fenómeno no sucede para protocolo DSDV porque este es un protocolo proactivo y el protocolo ADV es a su vez una combinación de las funcionalidades de los protocolos reactivos y protocolos proactivos.

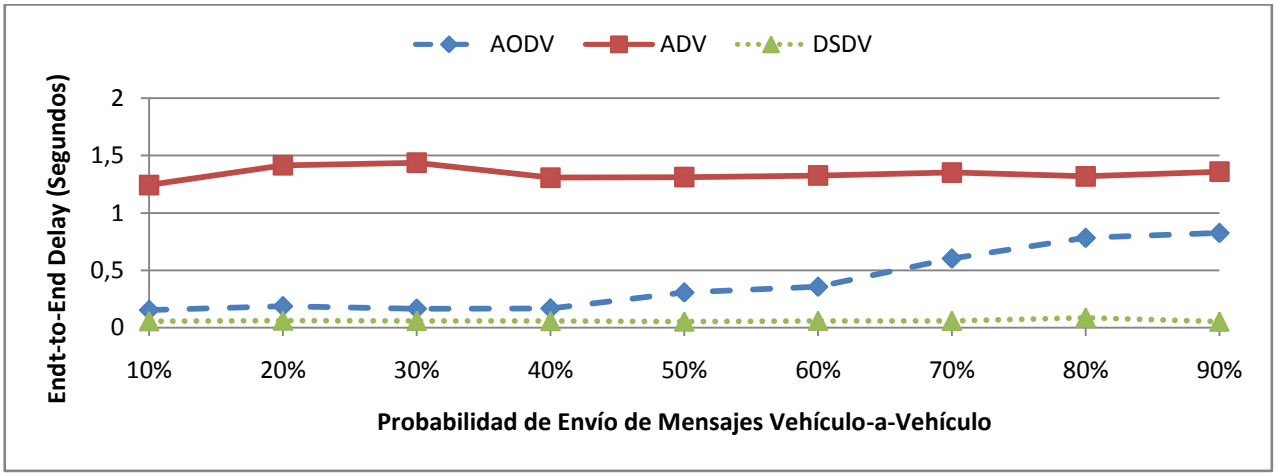


Figura 9.22: End-to-End Delay para el Benchmark Circular en Función de la Probabilidad de Envío de Mensajes Vehículo-a-Vehículo

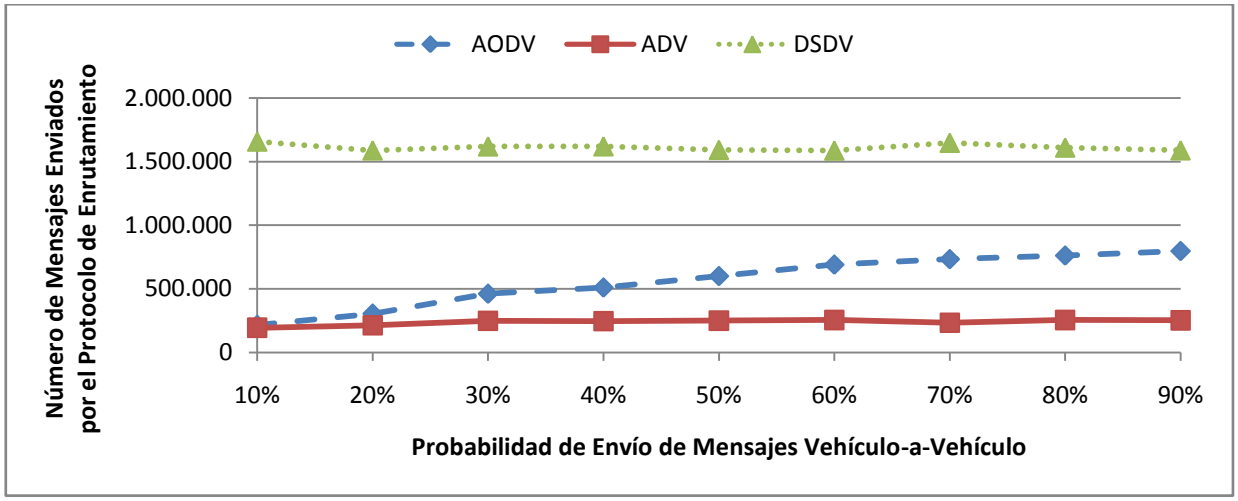


Figura 9.23: Número de Mensajes Enviados por el Protocolo de Enrutamiento para el Benchmark Circular en Función de la Probabilidad de Envío de Mensajes Vehículo-a-Vehículo

En la Figura 9.24 se observa los efectos del aumento de la probabilidad de envío de mensajes vehículo-a-vehículo en el NRL. Los protocolos ADV y DSDV mantienen valores constantes mientras que AODV sufre un aumento. Este comportamiento es esperado debido a que como se observa en la Figura 9.23 el protocolo AODV envía más paquetes de control a medida que aumenta la probabilidad de envío a otros vehículos. El aumento en el número de paquetes enviados por el protocolo también tiene un efecto en el tiempo real de simulación. Se puede observar en la Figura 9.25 como el tiempo real de simulación aumenta cuando se utiliza el protocolo AODV, sin embargo para los protocolos ADV y DSDV los valores se mantienen constantes.

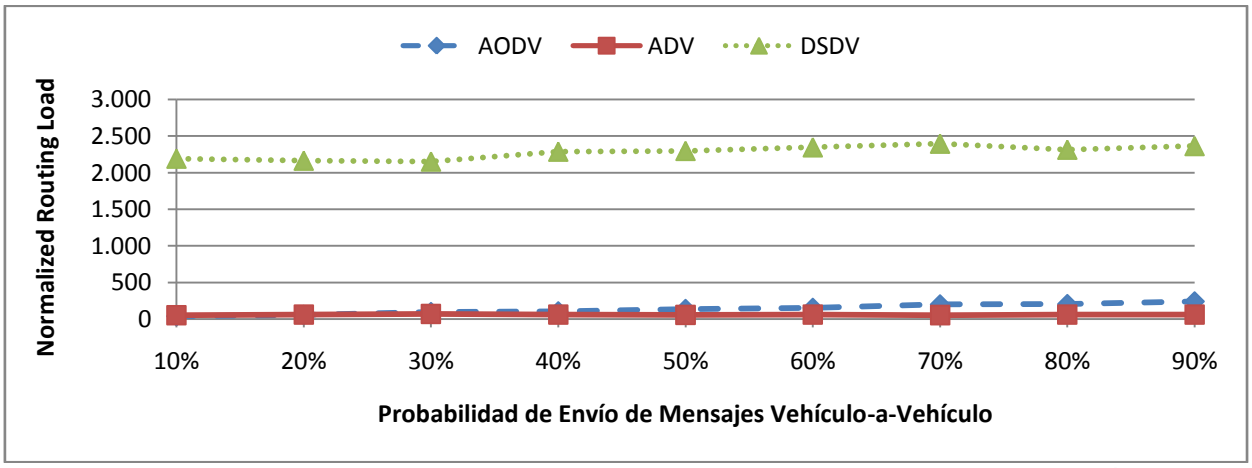


Figura 9.24: Normalized Routing Load para el Benchmark Circular en Función de la Probabilidad de Envío de Mensajes Vehículo-a-Vehículo

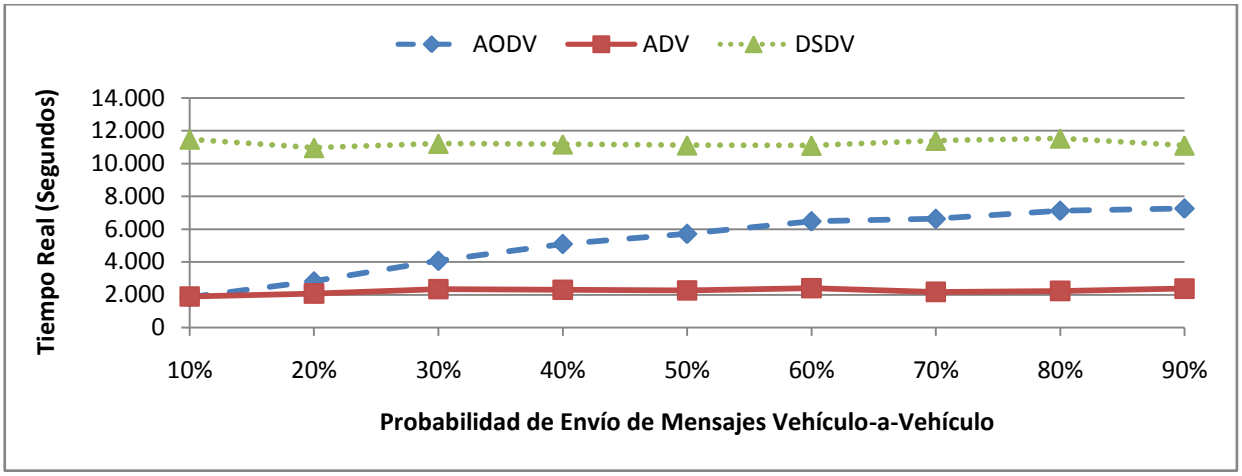


Figura 9.25: Tiempo Real para el Benchmark Circular en Función de la Probabilidad de Envío de Mensajes Vehículo-a-Vehículo

Por último, en la Figura 9.26 se observa cómo afecta el parámetro variado en el número de saltos promedio. Para los tres protocolos se observa que en promedio el número de saltos es constante. El protocolo DSDV bajo este escenario pudo hacer pocas entregas a más de dos saltos lo cual no es ideal para las redes vehiculares que se implementarán en un futuro. Sin embargo los protocolos ADV y AODV sí lograron realizar entregas a aquellos destinos que se encontraban a hasta 4 saltos.

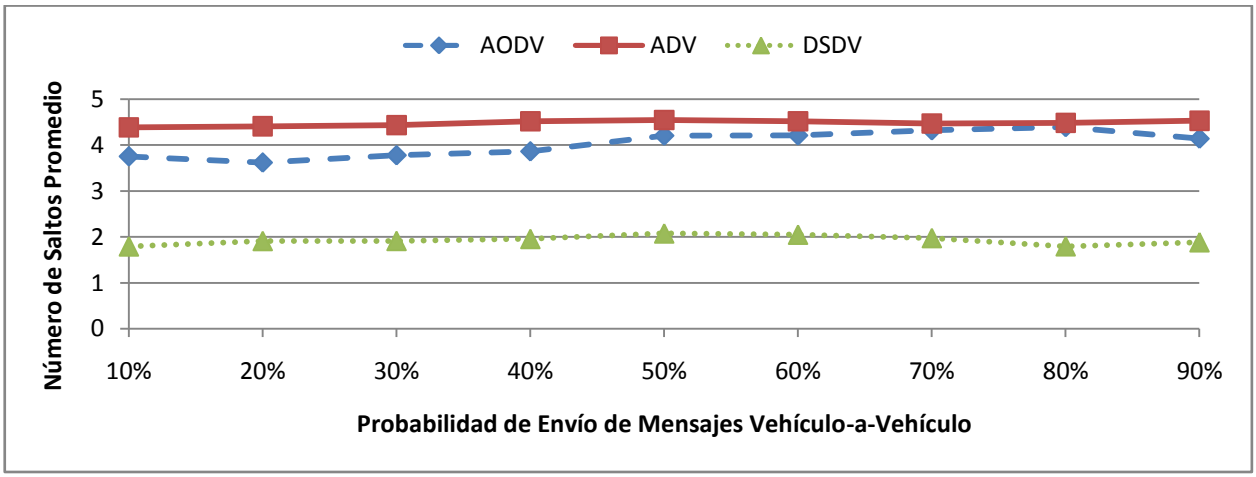


Figura 9.26: Número de Saltos Promedio para el Benchmark Circular en Función de la Probabilidad de Envío de Mensajes Vehículo-a-Vehículo

9.1.5 Variación del Número de RSUs Incluidos en la Simulación

Esta prueba fue diseñada para observar cómo influye la presencia de los RSUs en el comportamiento de la red vehicular simulada. Para ello, los RSUs fueron ubicados alrededor de la carretera circular. Los parámetros más relevantes de valores constantes configurados para este benchmark se encuentran reflejados en la Tabla 9.1. El parámetro a variar en esta prueba es el número de RSUs incluidos en la simulación desde 0 hasta 5 RSUs.

En la Figura 9.27 se puede observar que el número de RSUs no tiene efecto sobre los protocolos DSDV y ADV para las cantidades establecidas. En el caso del protocolo AODV se puede observar que el PDR es inferior cuando el número de RSUs es cero. Esto se puede atribuir al hecho de que AODV es un protocolo reactivo y al no haber RSUs en la

simulación el parámetro que indica la probabilidad de envío de mensajes vehículo-a-vehículo no es tomado en cuenta, o dicho de otra manera el valor de este parámetro se puede tomar como 100% cuando no hay RSUs. Esto hace que la probabilidad de que un nodo no conozca al receptor aumenta a diferencia de cuando existen RSUs en la simulación ya que la probabilidad de envío de mensajes vehículo-a-vehículo en este caso es de 50%.

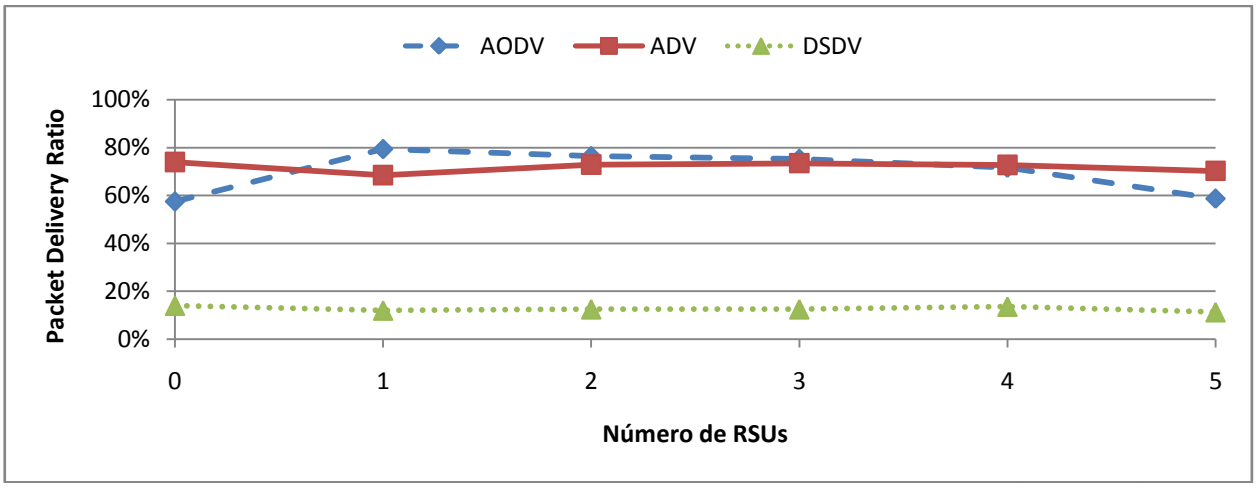


Figura 9.27: Packet Delivery Ratio para el Benchmark Circular en Función del Número de RSUs Incluidos en la Simulación

En la Figura 9.28 se puede comprobar que en efecto el número de paquetes enviados por el protocolo AODV es mayor cuando no hay presencia de RSUs en la simulación para todos los demás casos el número de paquetes enviados por el protocolo de enrutamiento es constante. De manera general el número de paquetes enviado por el protocolo DSDV es superior alde los protocolos ADV y AODV siendo el protocolo ADV el que envía un menor número de paquetes de control. El número de paquetes enviados por los protocolos de enrutamiento también tiene un efecto en el delay sufrido por los paquetes enviados como se puede observar en la Figura 9.29.Se presenta el mismo fenómeno para el caso del protocolo AODV sin RSUs en donde el delay es mayor cuando no hay presencia de RSUs.

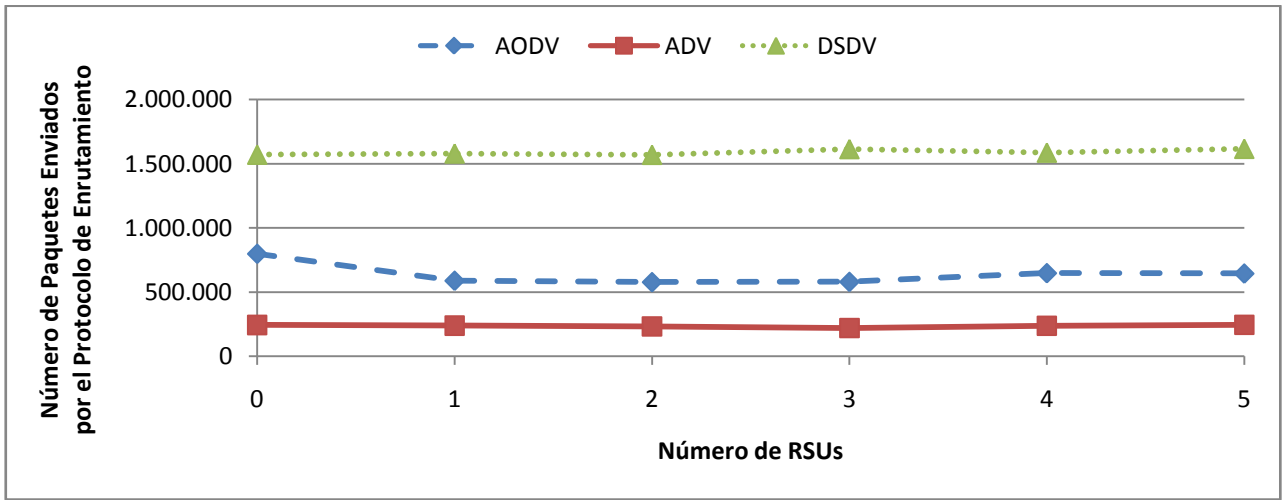


Figura 9.28: Número de Paquetes Enviados por el Protocolo de Enrutamiento para el Benchmark Circular en Función del Número de RSUs Incluidos en la Simulación

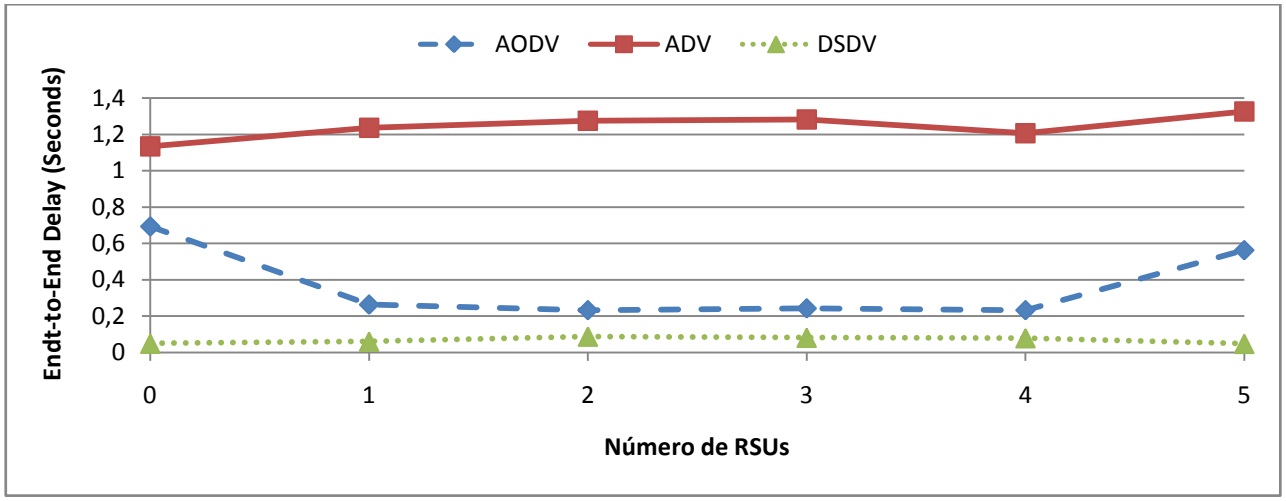


Figura 9.29: End-to End Delay para el Benchmark Circular en Función del Número de RSUs Incluidos en la Simulación

En la Figura 9.30 se observa el número de saltos promedio en función del número de RSUs, como se puede apreciar el número de saltos promedio se mantiene constante para los distintos protocolos de enrutamiento siendo el protocolo ADV el que mayor número de saltos logra alcanzar seguido por el protocolo AODV. El protocolo DSDV solo logra alcanzar aquellos receptores que están a no más de 2 saltos, en promedio.

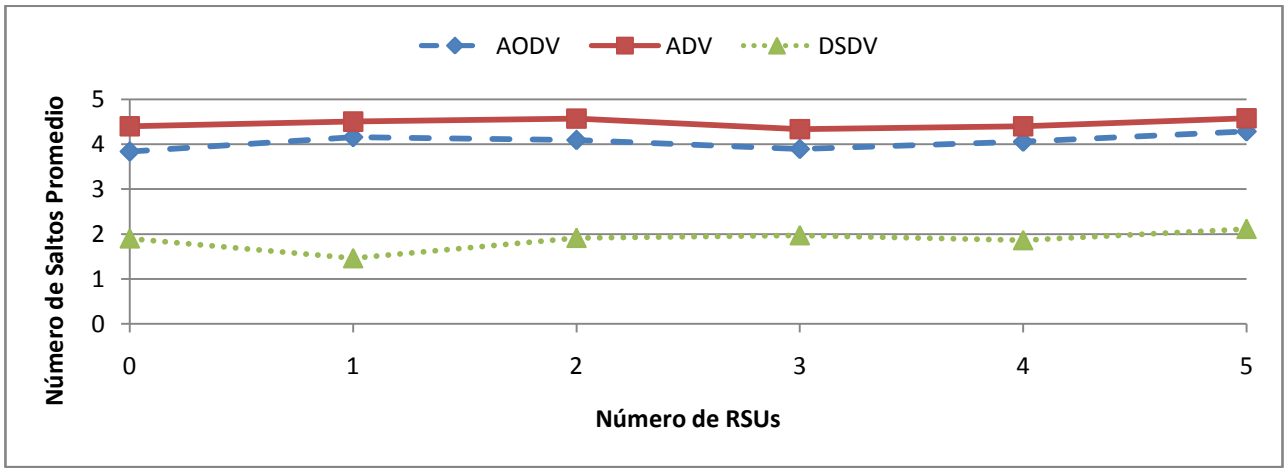


Figura 9.30: Número de Saltos Promedio para el Benchmark Circular en Función del Número de RSUs Incluidos en la Simulación

Por último, en la Figura 9.31y la Figura 9.32 se reportan el NRL y el tiempo real de simulación. Con respecto al NRL, se observa que este se mantiene cuando se varia el número de RSUs. De manera general el NRL es considerablemente mayor para el caso del protocolo DSDV, rondando los 2000 mensajes de control por cada mensaje de datos recibidos. En el caso de los protocolos AODV y ADV, el NRL son supera los 300 mensajes de control por cada mensaje recibido. Con respecto al tiempo real se observan los valores esperados, aunque con valores constantes, el protocolo DSDV toma mayor tiempo para culminar las simulaciones seguido por el protocolo AODV y por último el protocolo ADV.

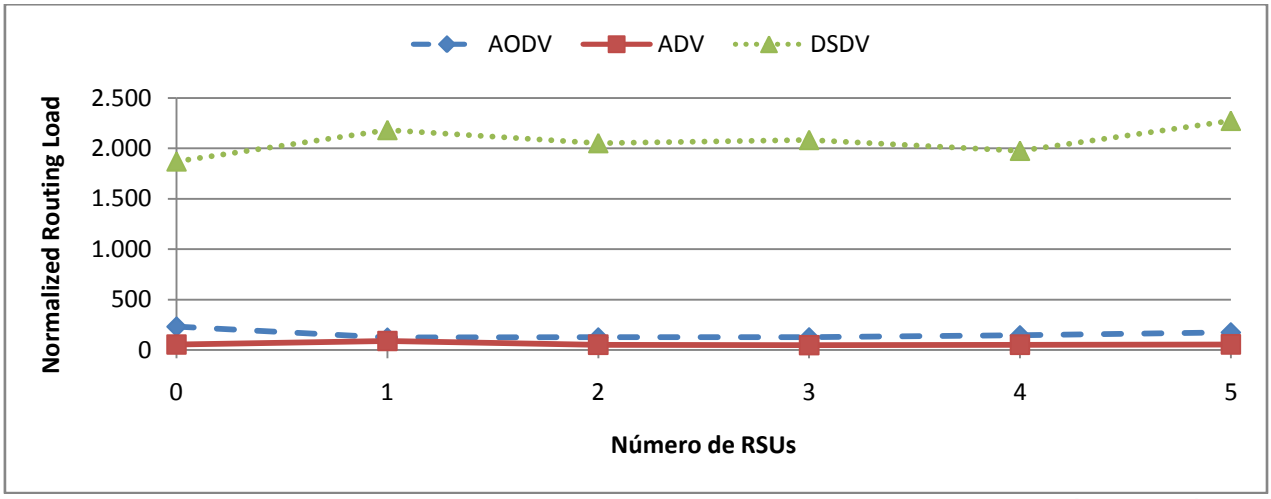


Figura 9.31: Normalized Routing Load para el Benchmark Circular en Función del Número de RSUs Incluidos en la Simulación

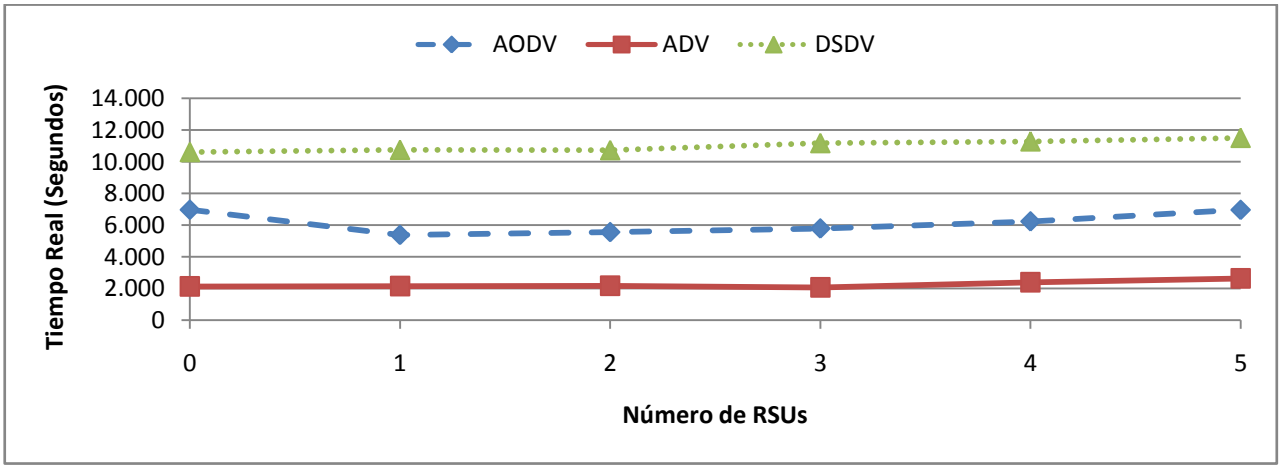


Figura 9.32: Tiempo Real de Simulación para el Benchmark Circular en Función del Número de RSUs Incluidos en la Simulación

9.2 Resultados de las Pruebas Realizadas en el Benchmark de la Ciudad

El objetivo de estas pruebas consiste en evaluar el comportamiento de los simuladores de red en un escenario realista con carreteras derivadas de mapas reales en el cual se forma una red vehicular con o sin la presencia de RSUs. Para estas pruebas se utilizaron trazas de ns-2 hechas sobre una sección del mapa de Caracas, con la intención de obtener resultados que se acerquen más a la realidad.Los valores de los parámetros más relevantes establecidos para las pruebas en este benchmark se muestran en la Tabla 9.2.

Parámetro	Valor por Defecto
Tiempo de duración de la simulación	30 segundos
Número de vehículos	200 vehículos
Número de RSUs	3 RSUs
Estándar WiFi	802.11a
Radio frecuencia	5 GHz
Rango de propagación de la señal	300m
Tamaño del payload de UDP	512 bytes
Bitrate	24 Mbps
Intervalo de envío de mensajes de los vehículos	1.0 segundos
Intervalo de envío de mensajes de los RSUs	0.5 segundos
Probabilidad de envío de mensajes vehículo-a-vehículo	50%
Protocolos de enrutamiento	AODV, ADV Y DSDV

Tabla 9.2: Valores por Defecto de los Parámetros de las Simulaciones Correspondientes al Benchmark de la Ciudad

9.2.1 Variación del Tamaño del Payload UDP de los Mensajes Enviados por los Vehículos

En la Figura 9.33se puede observar el PDR reportado para los 3 protocolos. El protocolo ADV es el que alcanza mejores resultados, con un PDR de hasta 80% mientras que los protocolos AODV y DSDV reportan valores alrededor del 40%. De manera general, el PDR se mantiene constante al variar el tamaño del payload UDP. Este mismo comportamiento ocurre en el caso del *End-to-End Delay*, como se observa en la Figura 9.34, el *End-to-End Delay* se mantiene constante al variar el tamaño del payload UDP. El protocolo ADV es el que reporta valores superiores a los 0.6 segundos mientras que los protocolos AODV y DSDV reportan valores por debajo de los 0.2 segundos. En la Figura 9.35 se observa el número de saltos promedio donde este no es influenciado por el tamaño del payload UDP. Los protocolos ADV y AODV logran alcanzar un número de saltos promedio de alrededor de 4 saltos mientras que el protocolo DSDV alcanza los 3 saltos en promedio.

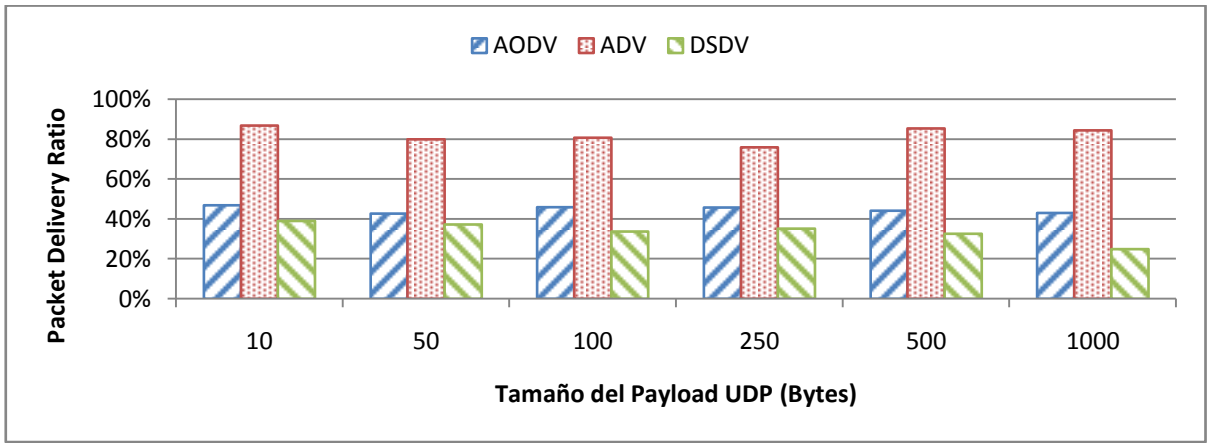


Figura 9.33: Packet Delivery Ratio para el Benchmark de la Ciudad en Función del Tamaño de la Carga Útil UDP

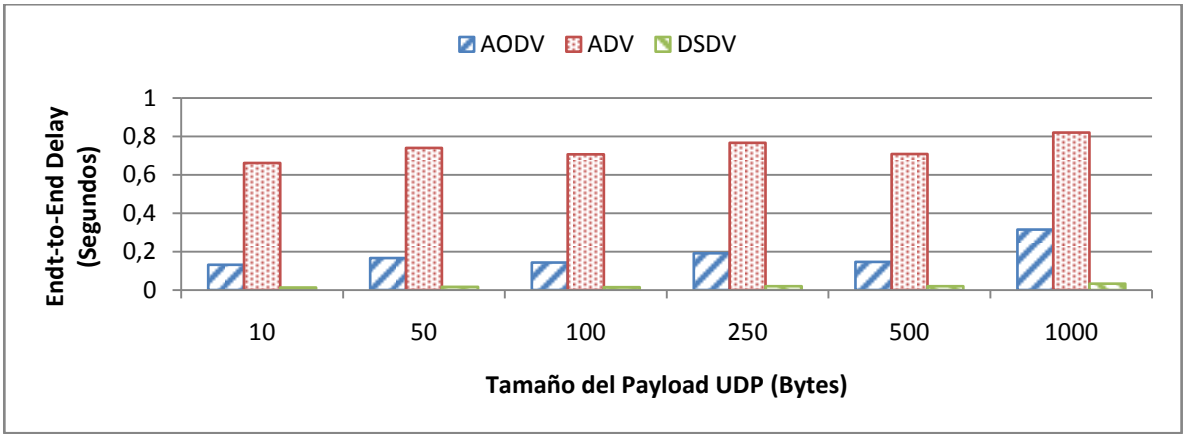


Figura 9.34:End-to-End Delay para el Benchmark Circular en Función del Tamaño de la Carga Útil UDP

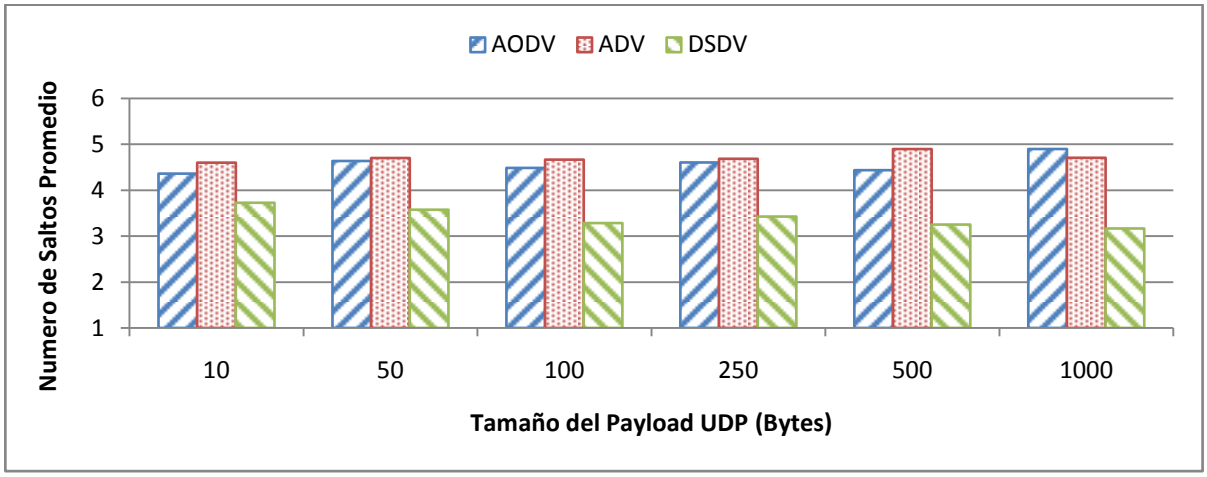


Figura 9.35:Número de Saltos Promedio para el Benchmark de la Ciudad en Función del Tamaño de la Carga Útil UDP

En la Figura 9.36 se observa cómo el NRL no se ve afectado por la variación del tamaño del payload UDP. El protocolo DSDV al ser un protocolo proactivo requiere del envío de hasta 1000 mensajes de control por cada mensaje de datos, mientras que los protocolos reactivos ADV y AODV no superan los 250 mensajes de control por cada mensaje de datos. En la Figura 9.37 se puede apreciar que el envío de mensajes de control por parte del protocolo DSDV es ampliamente superior que el de los protocolos ADV y AODV. Esto a su vez tiene efecto sobre el tiempo requerido para ejecutar la simulación, como se puede observar en la Figura 9.38. El protocolo DSDV es el que mayor tiempo requiere para ejecutar las simulaciones, hasta 8000 segundos, mientras que el protocolo AODV alrededor de 4000 segundos y el protocolo ADV requiere alrededor de 2000 segundos .

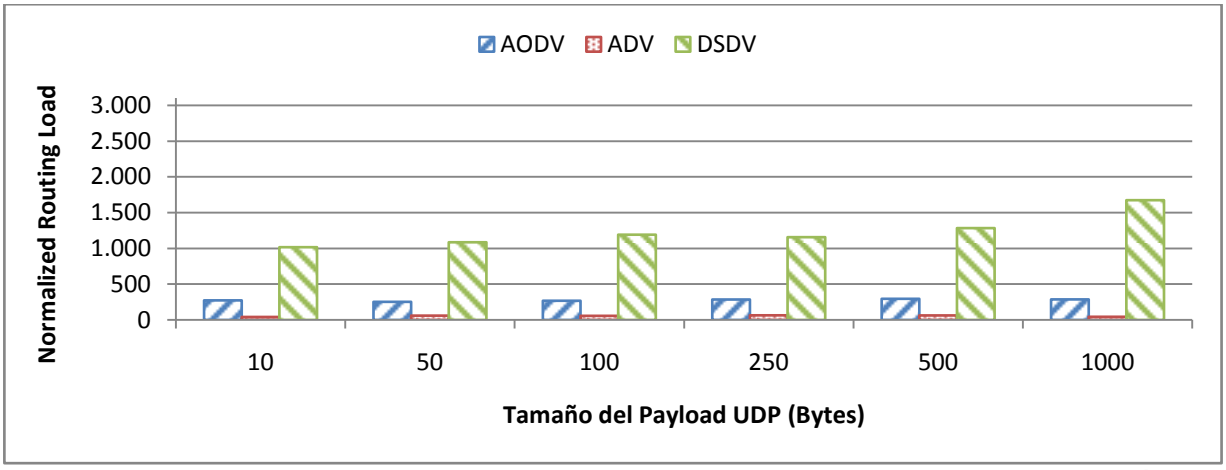


Figura 9.36:Normalized Routing Load para el Benchmark de la Ciudad en Función del Tamaño de la Carga Útil UDP

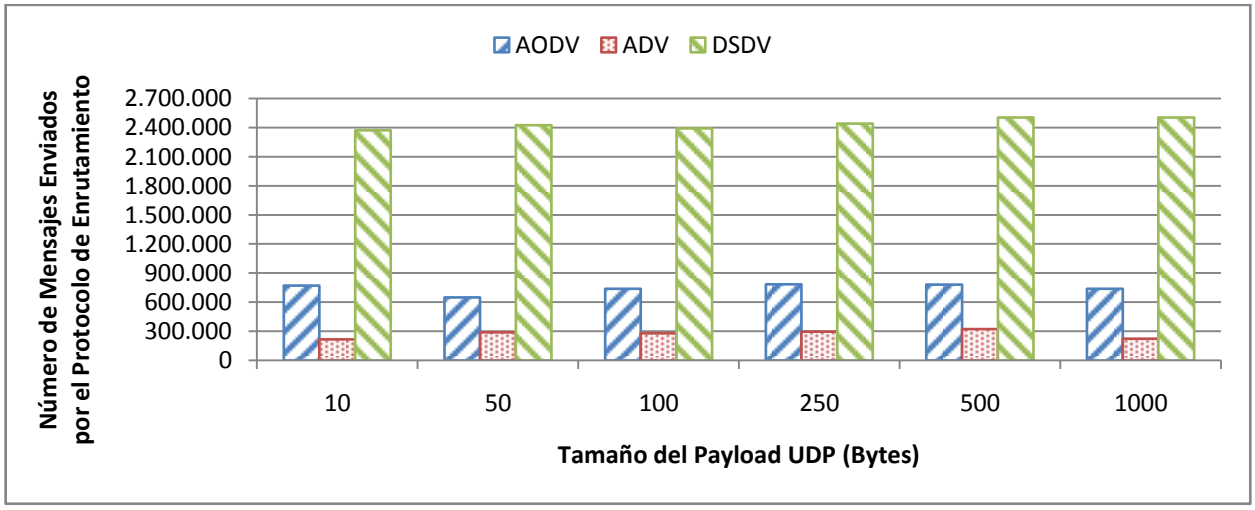


Figura 9.37: Número de Mensajes Enviados por el Protocolo de Enrutamiento para el Benchmark de la Ciudad en Función del Tamaño de la Carga Útil UDP

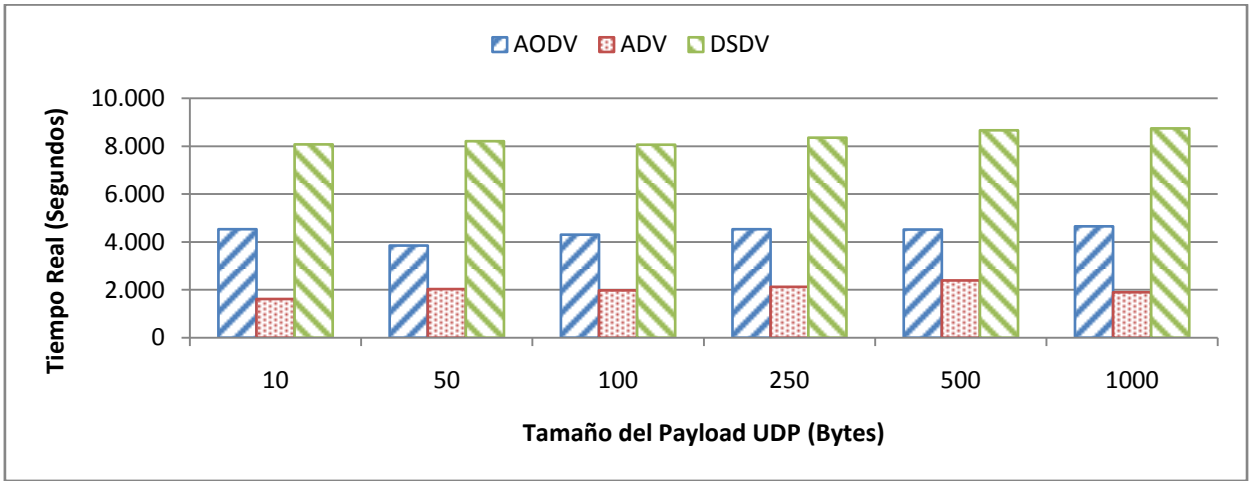


Figura 9.38: Tiempo Real para el Benchmark de la Ciudad en Función del Tamaño de la Carga Útil UDP

9.2.2 Variación del Número de Vehículos Incluidos en la Simulación

El objetivo de esta prueba es evaluar el efecto que tiene la cantidad de vehículos sobre el desempeño del simulador NCTUns y el desempeño de los protocolos de enrutamiento en el benchmark de la ciudad. Para estas simulaciones se utilizaron los valores por defecto presentados en la Tabla 9.2 y solo se varió el número de vehículos. Los valores para el número de vehículos utilizados fueron: 50, 100, 150, 200 y 250.

En la Figura 9.39 se puede apreciar el PDR reportado para los tres protocolos de enrutamiento. Los tres protocolos reportan una disminución en el PDR a medida que aumenta el número de vehículos incluidos en la simulación. Los protocolos ADV y DSDV reportan valores superiores al protocolo AODV y esta tendencia se mantiene a lo largo de las distintas cantidades de vehículos. Este fenómeno se puede deber al hecho de que mientras más vehículos existan en la red mayor es la competencia por el medio y el número de colisiones ya que el medio es compartido por un mayor número de vehículos.Esto también tiene un efecto sobre el end-to-end delay, al observar la Figura 9.40, se obtiene que el protocolo AODV presenta valores inferiores a los protocolos ADV y DSDV. Adicionalmente se puede observar que para los tres protocolos el end-to-end delay aumenta a medida que aumenta el número de vehículos.

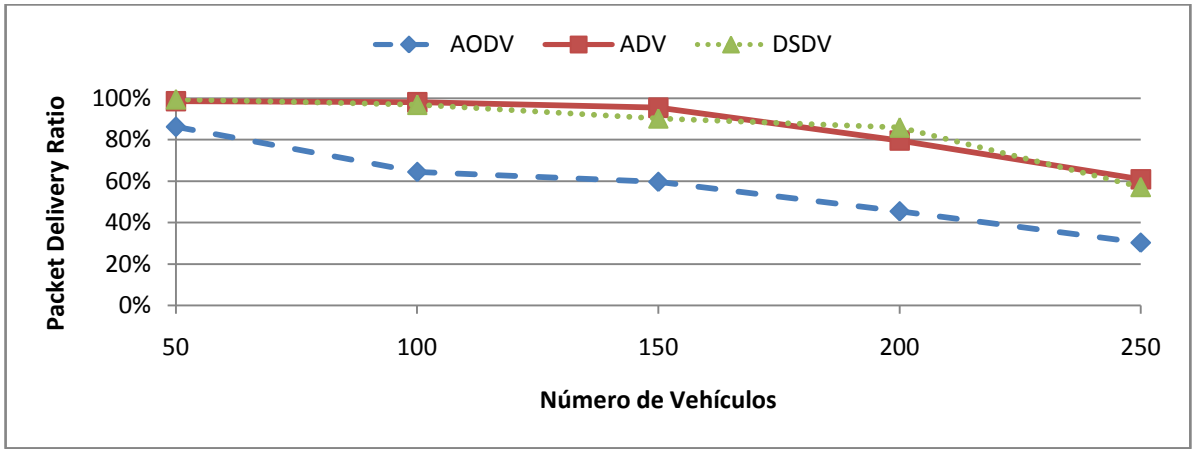


Figura 9.39: Packet Delivery Ratio para el Benchmark de la Ciudad en Función del Número de Vehículos Incluidos en la Simulación

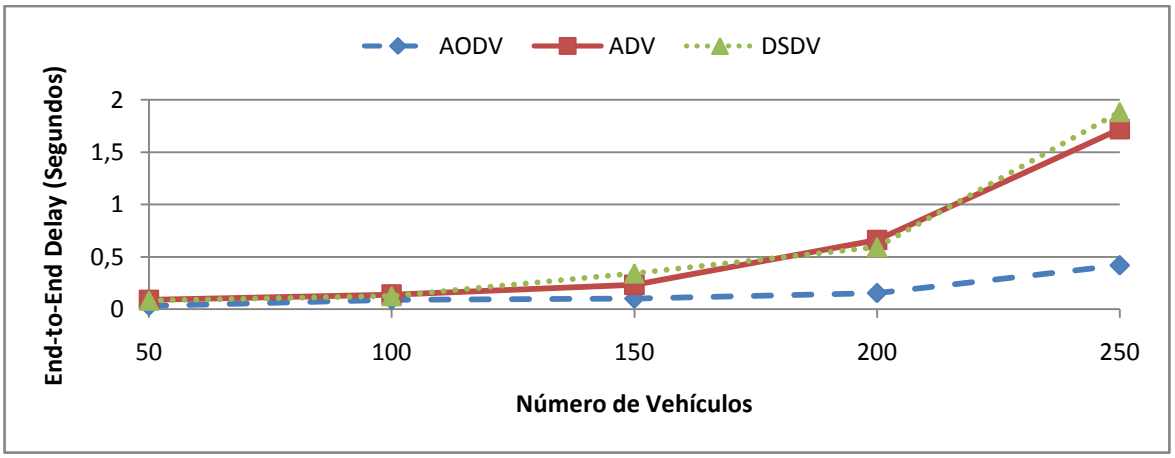


Figura 9.40: End-to-End Delay para el Benchmark de la Ciudad en Función del Número de Vehículos Incluidos en la Simulación

En la Figura 9.41 se observa cómo queda afectado el número de saltos promedio cuando se varía el número de vehículos. Para los tres simuladores el número de saltos promedio reportado es similar. De manera general el número de saltos promedio se encuentra alrededor de los 4 saltos y sufre un ligero aumento a medida que aumenta la cantidad de vehículos incluidos en la simulación. Este comportamiento es esperado ya que los vehículos están distribuidos en una porción de la red de carreteras de una ciudad, mientras un mayor número de vehículos se incluyen en la simulación mayor es la probabilidad de que un vehículo emisor este a mayor distancia del vehículo receptor.

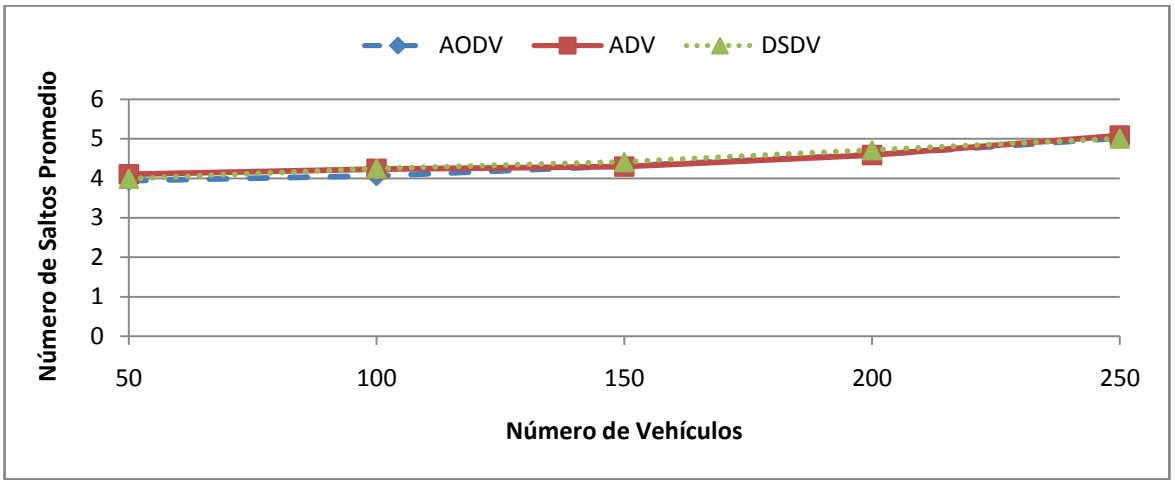


Figura 9.41:Número de Saltos Promedio para el Benchmark de la Ciudad en Función del Número de Vehículos Incluidos en la Simulación

La variación del NRL para los tres protocolos de enrutamiento se puede observar en la Figura 9.42. Los tres protocolos presentan un aumento en el NRL a medida que aumenta el número de vehículos. Este mismo fenómeno se puede observar en la Figura 9.43, en este caso correspondiente al número de mensajes enviados por los protocolos de enrutamiento. El NRL aumenta puesto que aumenta el número de mensajes enviados por los protocolos de enrutamiento y el PDR presenta una desmejora (como se observó en la Figura 9.39). El número de mensajes enviados por los protocolos de enrutamiento aumenta en el caso de los protocolos reactivos ADV y AODV ya que es más probable para un nodo emisor no conocer la ruta hacia el destino. En el caso de DSDV, al ser un protocolo proactivo, el número de mensajes aumenta ya que al existir más vehículos, existen más rutas que deben ser actualizadas por todos los vehículos.

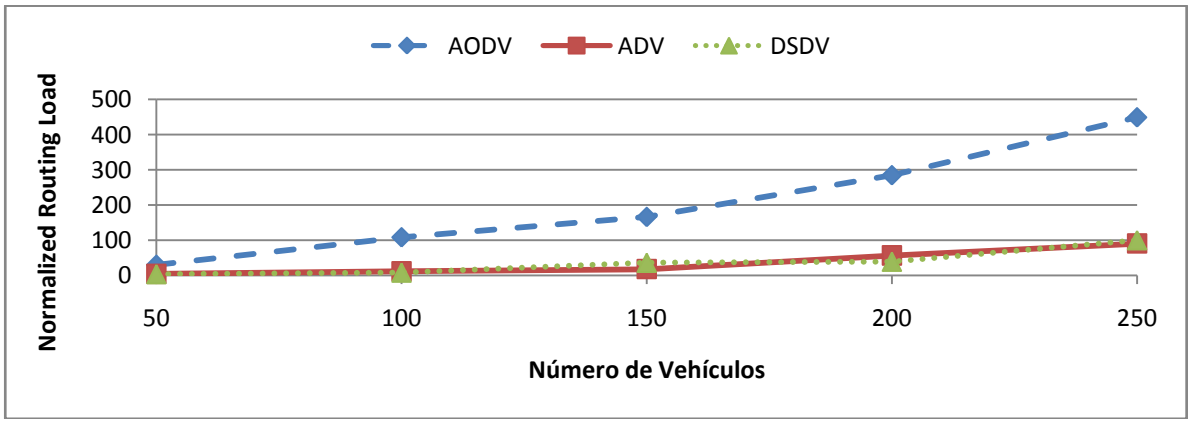


Figura 9.42: Normalized Routing Load para el Benchmark de la Ciudad en Función del Número de Vehículos Incluidos en la Simulación

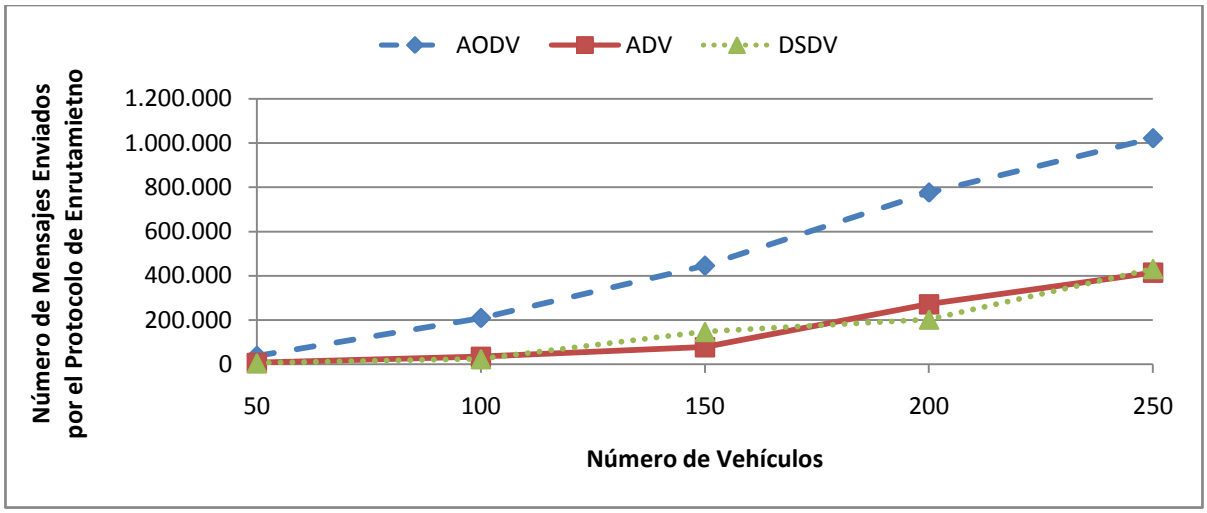


Figura 9.43: Número de Mensajes Enviados por el Protocolo de Enrutamiento para el Benchmark de la Ciudad en Función del Número de Vehículos Incluidos en la Simulación

En la Figura 9.44 se observan los valores correspondientes al tiempo real de simulación. Como es de esperarse, los tiempo de simulación aumentan a medida que aumenta el número de vehículos. En este caso, el protocolo AODV reporta mayores tiempos de simulación que los protocolos ADV y DSDV.

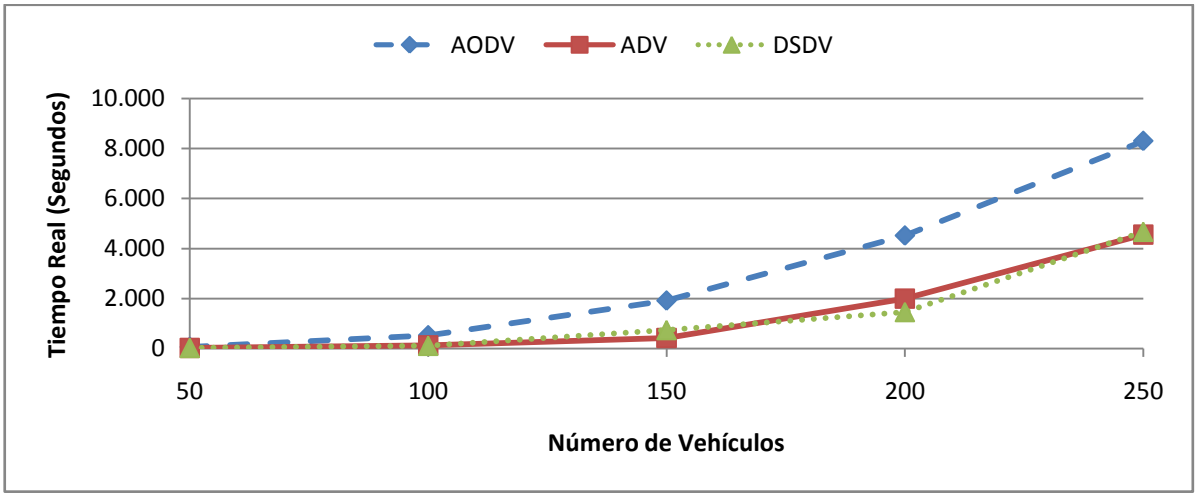


Figura 9.44: Tiempo Real para el Benchmark de la Ciudad en Función del Número de Vehículos Incluidos en la Simulación

Por último, en la Figura 9.45y la Figura 9.46 se reportan el uso del CPU y la memoria consumida para esta prueba, respectivamente. En el caso del uso del CPU, se observa que este se encuentra alrededor de un 99% para los tres protocolos cuando el número de vehículos supera los 100. Sin embargo para una cantidad de 50 vehículos los protocolo ADV y DSDV presentan un uso del CPU de alrededor de un 75% mientras que AODV mantiene un uso del CPU de 99%. En el caso de la cantidad de memoria consumida, los tres protocolos presentaron un consumo de memoria semejante. De manera general el consumo de memoria aumenta a medida que aumenta el número de vehículos. Esto se debe a que cada vehículo implica la ejecución de un proceso para el envío de mensajes a otros vehículos y el movimiento de cada vehículo debe ser controlado por el simulador.

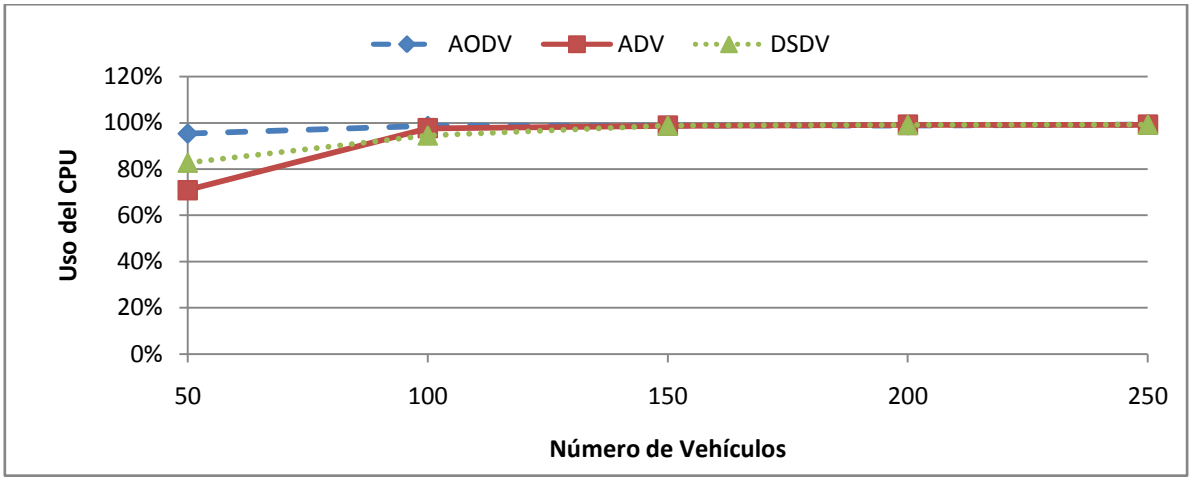


Figura 9.45: Uso del CPU para el Benchmark de la Ciudad en Función del Número de Vehículos Incluidos en la Simulación

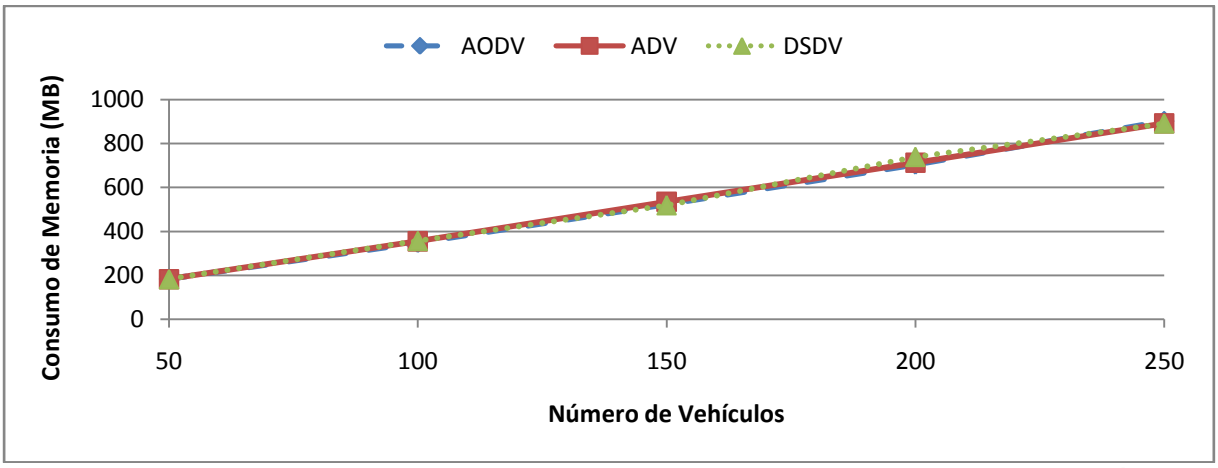


Figura 9.46: Consumo de Memoria para el Benchmark de la Ciudad en Función del Número de Vehículos Incluidos en la Simulación

9.2.3 Variación de la Probabilidad de Envío de Mensajes Vehículo-a-Vehículo

El objetivo de esta prueba es el mismo de la prueba realizada en el benchmark circular (Sección 9.1.4), la diferencia es que en este caso el movimiento de los vehículos es especificado por una traza ns-2 tomada de una red de carreteras de la vida real. Los RSUs son ubicados en posiciones fijas dentro del área que cubre la red de carreteras de tal manera que estos no se solapen y evitando que los RSUs generen interferencias entre sí. Los parámetros más relevantes de valores constantes configurados para este benchmark se encuentran reflejados en la Tabla 9.2.

En la Figura 9.47 se puede observar los resultados correspondientes al PDR en el caso de los tres protocolos de enrutamiento. Los protocolos ADV y DSDV presentan un ligero aumento a medida que crece la probabilidad de envío de mensajes vehículo-a-vehículos mientras que el protocolo AODV presenta un comportamiento constante a lo largo de las distintas probabilidades. Estopuede deberse a que los vehículos tienen más chance de que estén rodeados de otros vehículos que de estar cerca de un RSU, lo que provoca que a mayor probabilidad, los vehículos cercanos recibirán más mensajes. El protocolo DSDV presenta valores de alrededor de 20% mientras que el protocolo AODV presenta valores

alrededor de 50% y finalmente el protocolo ADV es el que mejor desempeño muestra en cuanto al PDR con valores superiores al 60%.

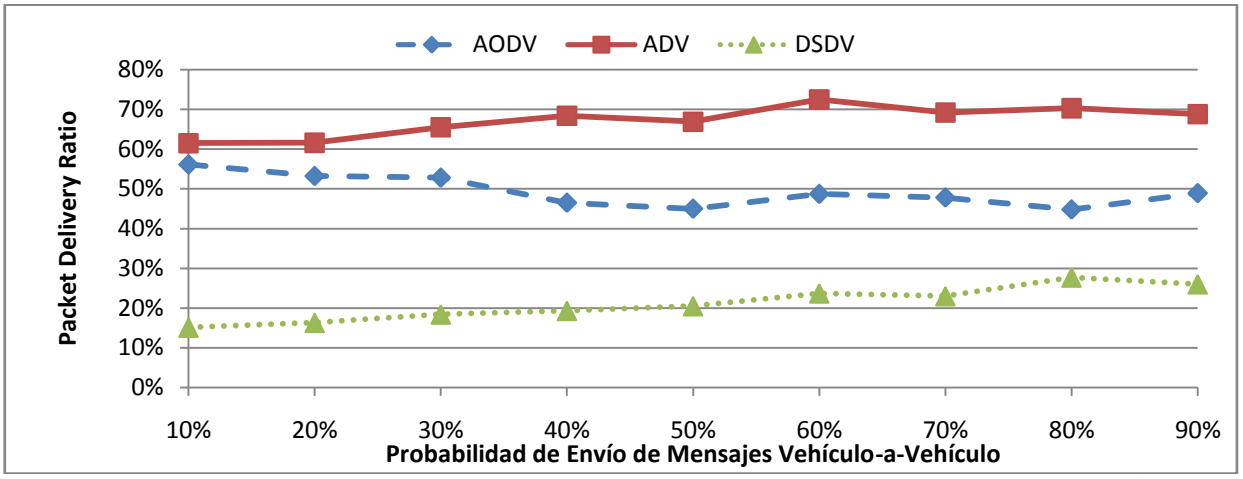


Figura 9.47: Packet Delivery Ratio para el Benchmark de la Ciudad en Función de la Probabilidad de Envío de Mensajes Vehículo-a-Vehículo

En la Figura 9.48 se observa el end-to-end delay promedio para los tres protocolos. El protocolo ADV presenta valores por encima de los 0.6 segundos mientras que los protocolos AODV y DSDV mantienen valores por debajo de los 0.2 segundos. En la Figura 9.49 se puede observar el número de saltos promedio para los tres protocolos de enrutamiento. Los protocolos ADV y AODV muestran un ligero aumento en cuanto al número de saltos a medida que aumenta la probabilidad de envío de mensajes vehículo-a vehículo, mientras que el protocolo DSDV disminuye ligeramente el número de saltos. De manera general el número de saltos promedio en los tres protocolos es de alrededor de 4 saltos.

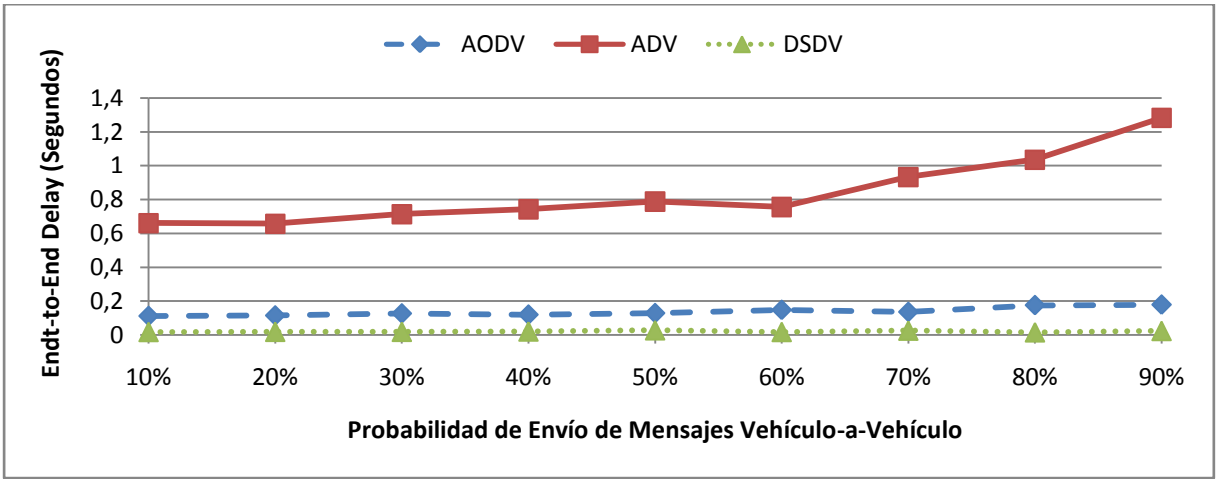


Figura 9.48: End-to-End Delay para el Benchmark de la Ciudad en Función de la Probabilidad de Envío de Mensajes Vehículo-a-Vehículo

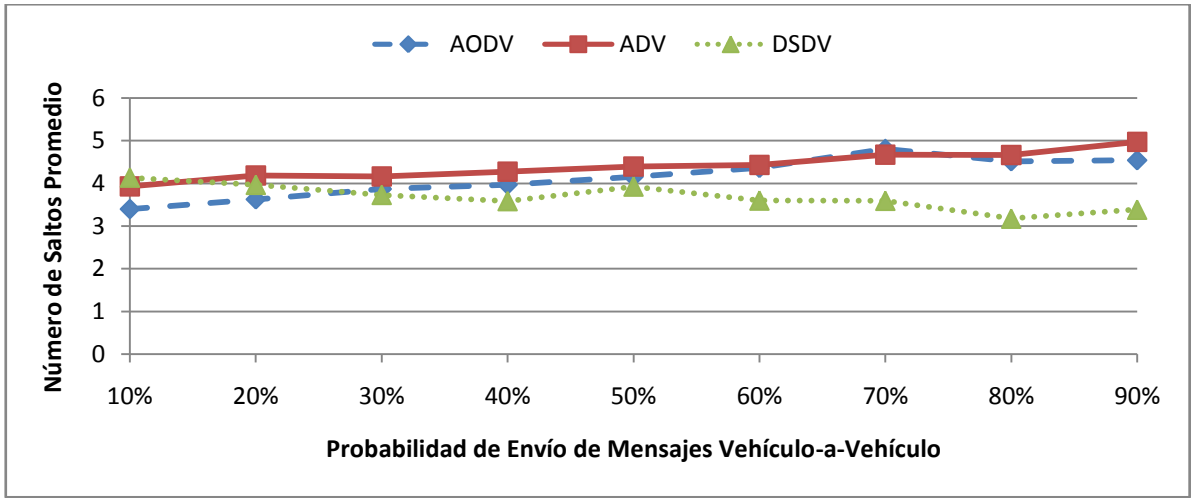


Figura 9.49: Número de Saltos Promedio para el Benchmark de la Ciudad en Función de la Probabilidad de Envío de Mensajes Vehículo-a-Vehículo

En la Figura 9.50 se puede apreciar el NRL reportado por los tres protocolos de enrutamiento. Los protocolos AODV y ADV presentan valores constantes del NRL mientras que el protocolo DSDV presenta una disminución en los valores correspondientes al NRL a medida que aumenta la probabilidad de envío de mensajes. En la Figura 9.51 se puede apreciar que el número de mensajes enviados por el protocolo DSDV se mantiene constante, sin embargo, como se observó en la Figura 9.47, el PDR correspondiente al protocolo DSDV aumenta a medida que aumenta la probabilidad de envío a otros vehículos. Es por esta razón que se observa una disminución en el NRL en el caso del protocolo DSDV.

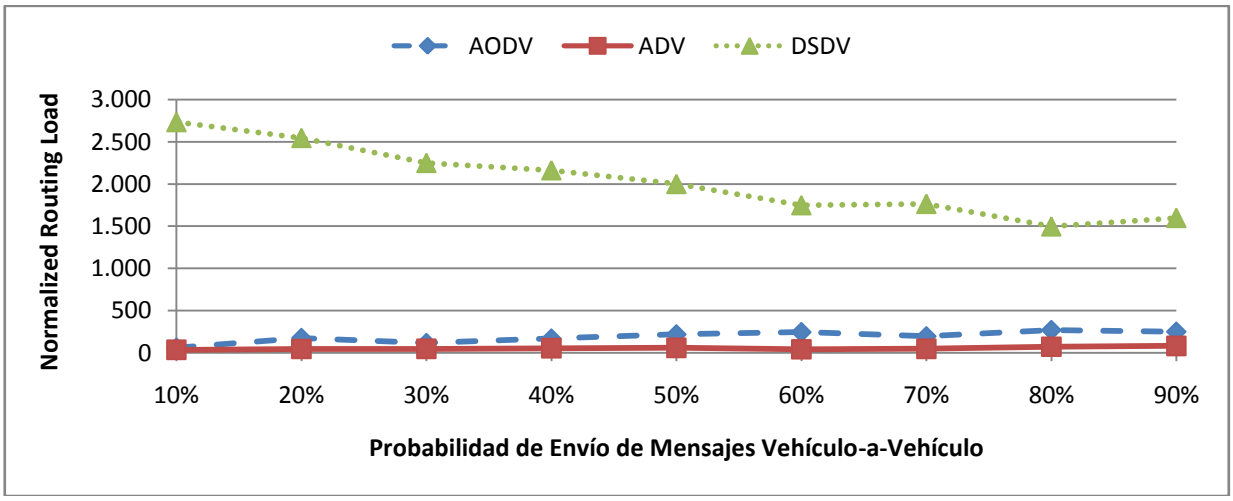


Figura 9.50: Normalized Routing Load para el Benchmark de la Ciudad en Función de la Probabilidad de Envío de Mensajes Vehículo-a-Vehículo

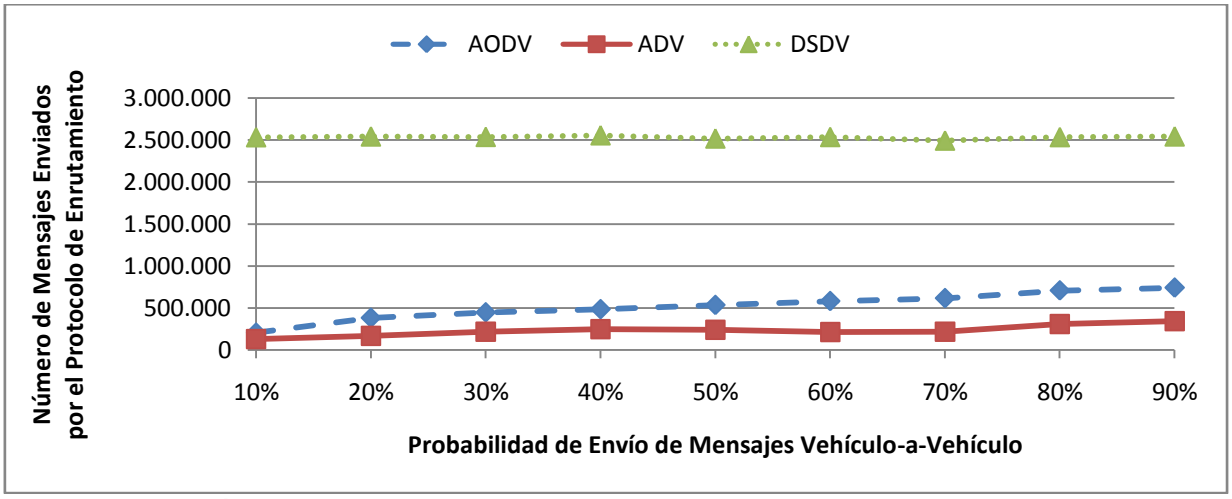


Figura 9.51: Número de Mensajes Enviados por el Protocolo de Enrutamiento para el Benchmark de la Ciudad en Función de la Probabilidad de Envío de Mensajes Vehículo-a-Vehículo

Finalmente, en la Figura 9.52 se observa los valores correspondientes al tiempo real de simulación. Los tiempos de simulación se mantienen constantes a través de las distintas probabilidades. Se mantienen la tendencia en la cual el protocolo DSDV es el que requiere mayor tiempo para realizar las simulaciones, seguido por el protocolo AODV y por último el protocolo ADV el cual presenta los tiempos más cortos.

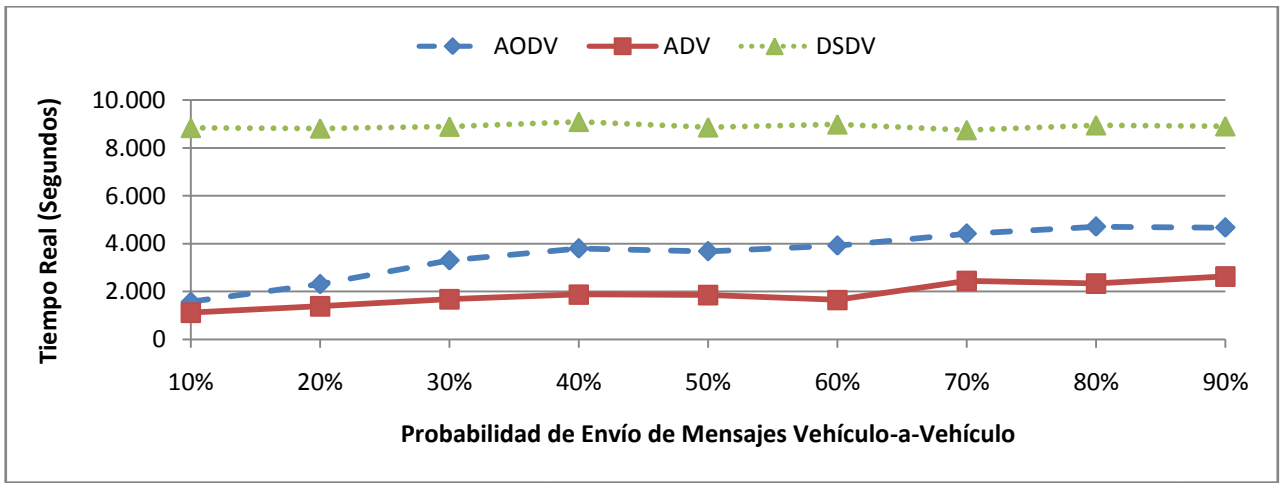


Figura 9.52: Tiempo Real de Simulación para el Benchmark de la Ciudad en Función de la Probabilidad de Envío de Mensajes Vehículo-a-Vehículo

9.2.4 Variación del Intervalo de Envío de Paquetes de Datos

El parámetro a variar en esta prueba es el intervalo para el de envío de mensajes de los vehículos, dicho de otra manera, lo que se varía en cada simulación es la cantidad de mensajes que envían los vehículos por segundo. La idea es observar cómo influye la variación de este parámetro en el comportamiento de la red simulada por cada protocolo de enrutamiento. Cada simulación se realizó con los siguientesintervalos para el envío de mensajes: 2.0, 1.0, 0.5, 0.25, 0.1667, 0.125 y 0.1 segundos, lo que es el equivalente a enviar 0.5, 1, 2, 4, 6, 8 y 10 datagramas UDP por segundo.

En la Figura 9.53 se muestra el PDR para los tres protocolos. De manera general, el PDR disminuye a medida que aumenta el número de paquetes enviados por segundo. Este comportamiento se debe a que cuando los vehículos envían más paquetes por segundo se crea más congestión en la red aumentando así el número de colisiones y dificultando la entrega satisfactoria de los paquetes a sus respectivos destinos. Por estas razones, como se puede ver en la Figura 9.54 se aprecia el cómo se refleja un aumento en el end-to-end delay para los tres protocolos. El protocolo ADV presenta valores superiores a los protocolos AODV y DSDV. Sin embargo, cabe destacar que el protocolo ADV logró los valores más altos en cuanto al PDR lo que significa que pudo entregar satisfactoriamente mayor cantidad de paquetes que los protocolos AODV y DSDV.

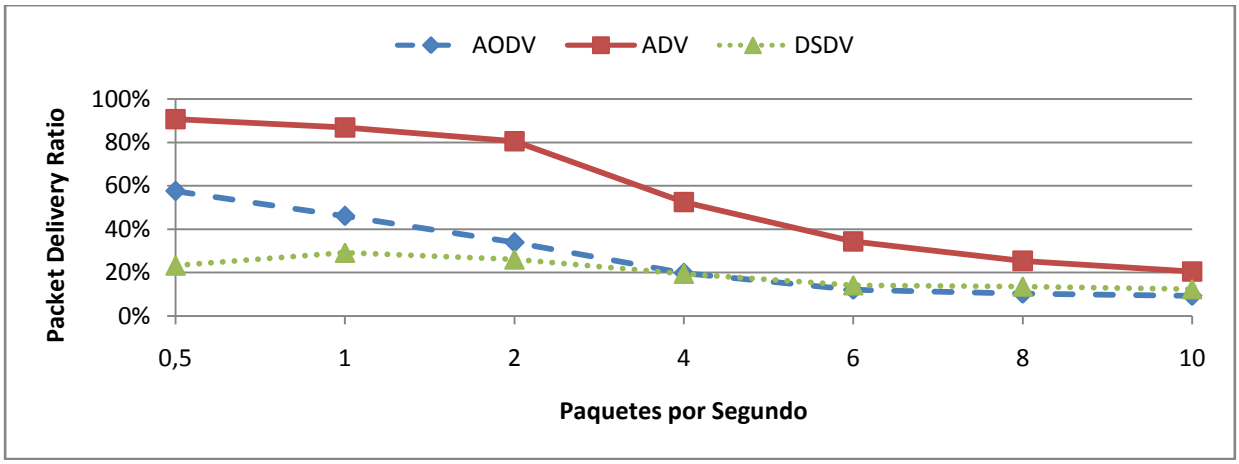


Figura 9.53: Packet Delivery Ratio para el Benchmark de la Ciudad en Función del Intervalo de Envío de Paquetes de Datos

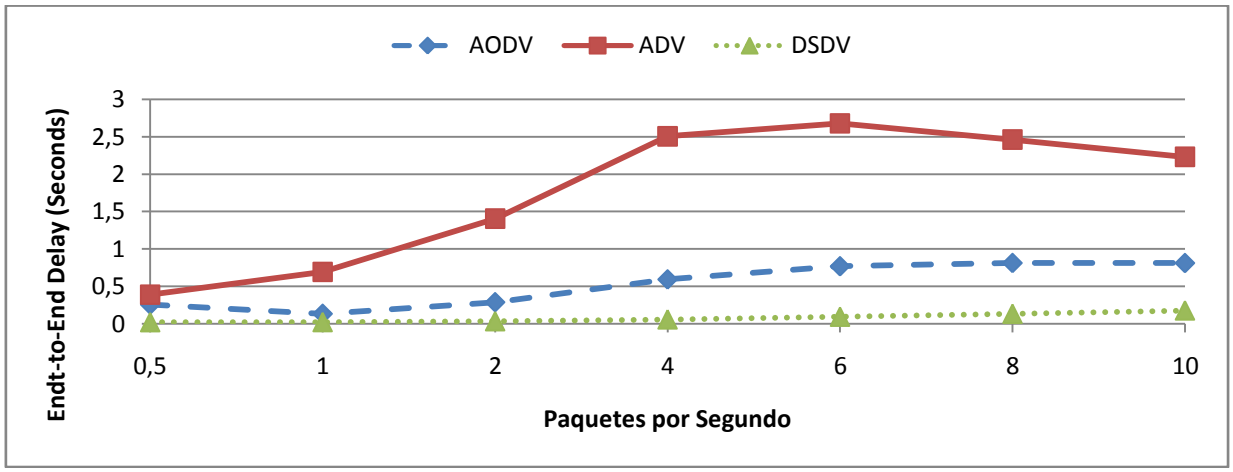


Figura 9.54: End-to-End Delay para el Benchmark de la Ciudad en Función del Intervalo de Envío de Paquetes de Datos

En la Figura 9.55 se puede observar los valores correspondientes al número de saltos promedio para los tres protocolos. Los protocolos ADV y AODV lograron entregar paquetes a destinos que están alrededor de 4 saltos mientras que el protocolo DSDV presenta un promedio de 3 saltos.

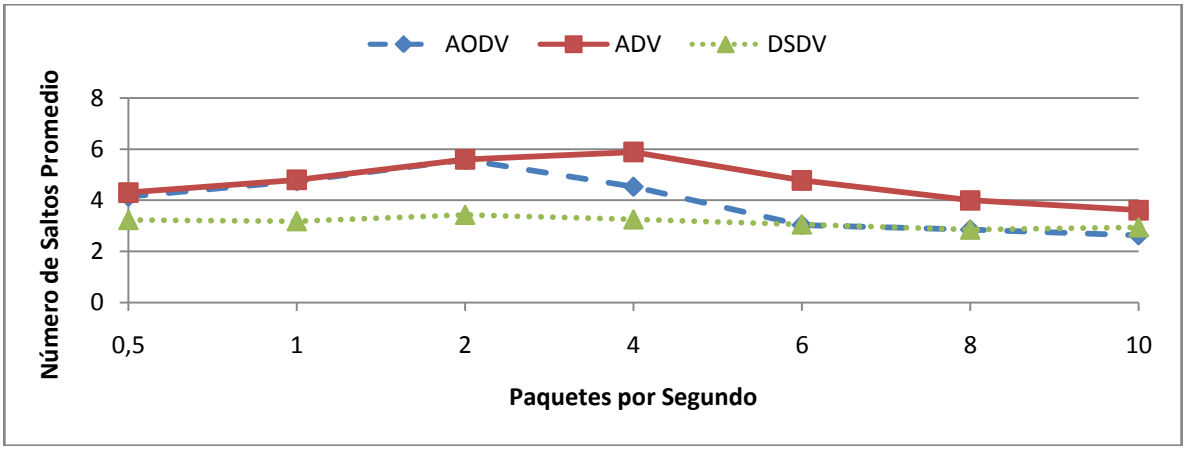


Figura 9.55: Número de Saltos Promedio para el Benchmark de la Ciudad en Función del Intervalo de Envío de Paquetes de Datos

El NRL se mantiene constante para los protocolos ADV y AODV, como se puede apreciar en la Figura 9.56. Sin embargo, el protocolo DSDV presenta una disminución en el NRL a medida que se envían más paquetes por segundo. Al observar la Figura 9.57 se puede notar que el número de mensajes de control enviados por los protocolos ADV y AODV se mantienen constantes mientras que para el protocolo DSDV se nota una disminución en el número de paquetes de control enviados. Esta disminución en el número de paquetes de control enviados mientras el PDR se mantiene constante como se observó en la Figura 9.53revela la razón de la disminución del NRL para el protocolo DSDV.

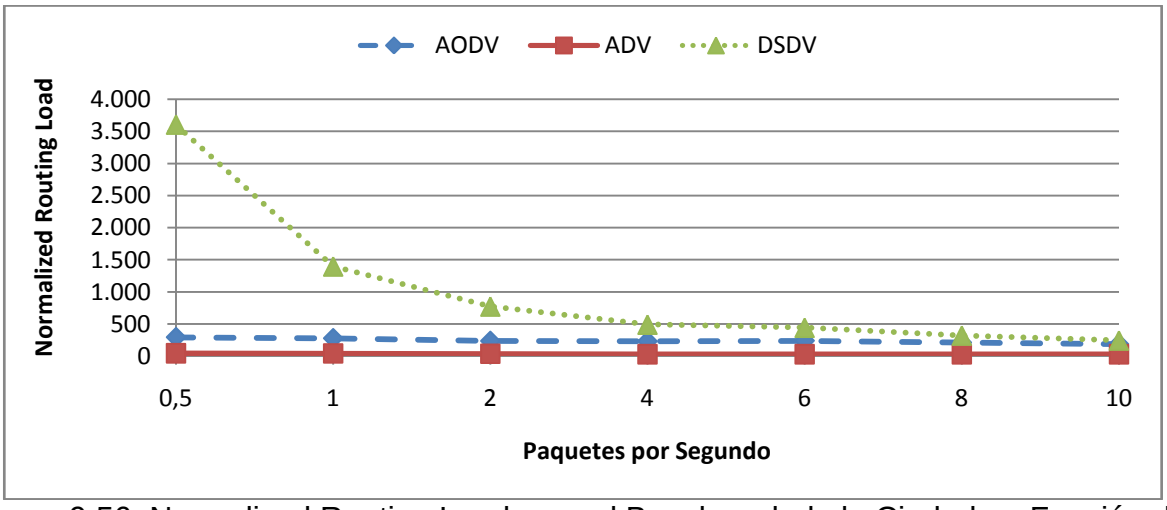


Figura 9.56: Normalized Routing Load para el Benchmark de la Ciudad en Función del Intervalo de Envío de Paquetes de Datos

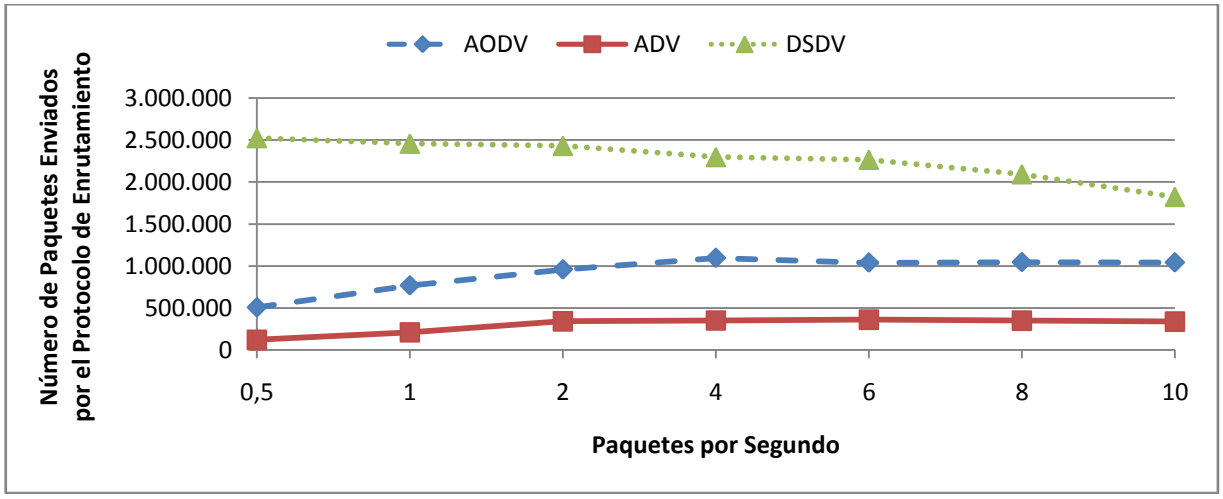


Figura 9.57: Número de Paquetes Enviados por el Protocolo de Enrutamiento para el Benchmark de la Ciudad en Función del Intervalo de Envío de Paquetes de Datos

Finalmente, en la Figura 9.58 se observan los valores presentados por los tres protocolos correspondientes al tiempo real de simulación. El protocolo DSDV,al ser un protocolo proactivo,mantiene tiempos constantes y superiores a los tiempos presentados por los protocolos ADV y AODV. Estos dos últimos protocolos muestran un aumento en los tiempos de simulación a medida que los vehículos envían más paquetes por segundo ya que al ser protocolos reactivos, el envío de más paquetes de datos aumentan la posibilidad de envío de más paquetes de control por parte de los protocolos de enrutamiento.

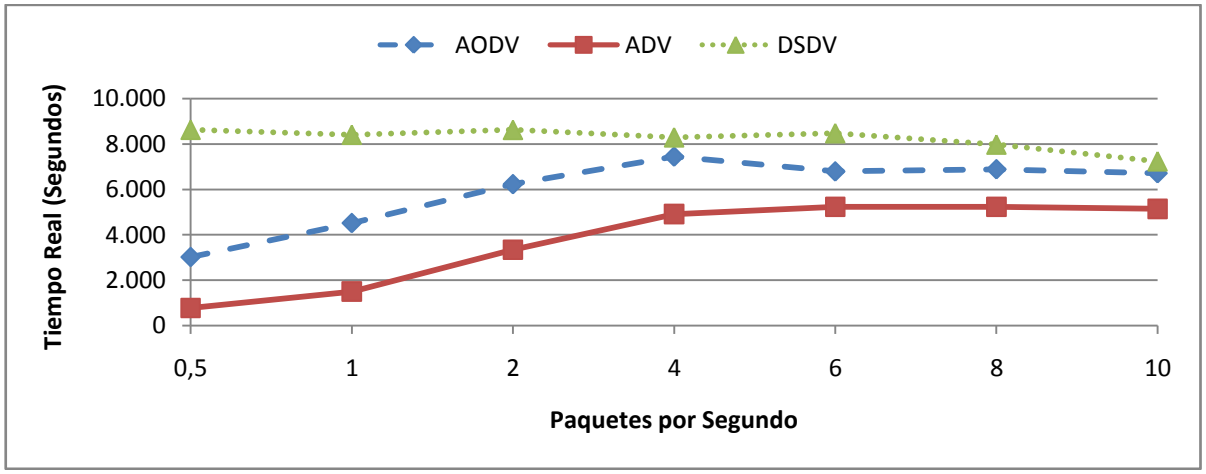


Figura 9.58: Tiempo Real de Simulación para el Benchmark de la Ciudad en Función del Intervalo de Envío de Paquetes de Datos

9.3 Especificaciones Técnicas para el Desarrollo y Uso de los Benchmarks

El simulador NCTUns ha sido desarrollado específicamente para la distribución de Linux Fedora. La implementación y ejecución de los benchmarks fue realizada en computadores HP xw4600 con procesadores Intel Core 2 Duo E6750 de 2.67 GHz y 4 GB de memoria RAM. La Tabla 9.3 presenta las versiones utilizadas de los componentes necesarios para el desarrollo y ejecución de los benchmarks.

Componente	Especificación
Sistema Operativo	Fedora Core 12
Kernel	2.6.31.6
NCTUns	6.0
GNU C++	4.4.4
GNU Make	3.81

Tabla 9.3: Especificaciones Técnicas de Componentes Necesarios para los Benchmarks

Las trazas formato ns-2 utilizadas fueron creadas utilizando SUMO versión 0.15.0. La versión de la herramienta ns-2 Trace Toolkit utilizada fue la v3.5.0.

10. Conclusiones y Trabajos Futuros

El estudio de las redes vehiculares es de gran importancia para la solución de problemas inherentes al tráfico vehicular. Se han desarrollado estándares para la comunicación V2V y V2I en redes vehiculares. También se han propuesto protocolos para el enrutamiento en redes vehiculares y aplicaciones para mejorar la seguridad física de los usuarios. Estas tecnologías en desarrollo requieren el uso de herramientas de simulación para la realización de experimentos principalmente por motivos de seguridad física de los conductores durante las pruebas y de reducción de costos, debido a la gran cantidad de vehículos que deben ser involucrados para obtener resultados realistas.

Los simuladores de redes vehiculares son herramientas que también han ido evolucionando y adaptándose a las exigencias de las características únicas que poseen las redes vehiculares. Existen varias maneras de llevar a cabo la simulación de redes vehiculares ya sea combinando el uso de simuladores de redes con simuladores de tráfico o utilizando simuladores que integren ambos tipos de simulación. Es importante para las investigaciones que realizan simulaciones de redes vehiculares determinar las capacidades que tienen los simuladores de redes vehiculares existentes en la actualidad. Es por ello que en este trabajo se desarrolló un conjunto de benchmarks para evaluar el desempeño de uno de estos simuladores: NCTUns.

El simulador NCTUns provee simulaciones de alta fidelidad debido al uso de la pila de protocolos TCP/IP en las simulaciones y permite correr aplicaciones reales en las simulaciones lo cual es una gran ventaja en comparación con otros simuladores.Debido a las características que tienen las redes vehiculares, es de suma importancia realizar simulaciones de escenarios con gran cantidad de vehículos, sin embargo, bajo la implementación utilizada,se encontró que NCTUns no permite la simulación de una cantidad de nodos mayor a 254 lo cual es una gran limitante debido al alto número de vehículos que transitan en las calles y que va en aumento. Otra desventaja de NCTUns con respecto a otros simuladores son los altos tiempos requeridos para culminar las simulaciones como quedó evidenciado en los experimentos realizados utilizando ambos benchmarks.

Al analizar los resultados de los experimentos realizados se puede señalar que los protocolos de enrutamiento para redes móviles no son aptos para las redes vehiculares. Una razón para esto es el alto dinamismo que implican las redes vehiculares en comparación con otras redes móviles. De manera general, el protocolo de enrutamiento ADV presentó el mejor desempeño en la mayoría de los experimentos realizados. El protocolo DSDV presentó el peor desempeño en el benchmark circular, sin embargo en el benchmark de la ciudad obtuvo un rendimiento similar al de ADV. El protocolo AODV tuvo un rendimiento similar en ambos benchmarks siempre por debajo del protocolo ADV.

Fundamentado en el desarrollo de este Trabajo Especial de Grado, se propone como trabajo futurola entonación de los parámetros de los protocolos de enrutamiento de redes móviles para su adaptación a las redes vehiculares.Finalmente, se propone como trabajo futuro el desarrollo de un protocolo de enrutamiento más adecuado para las exigencias de las redes vehiculares.

Referencias Bibliográficas

[1] J. Harri, F. Filali, and C. Bonnet, "Mobility Models for Vehicular Ad Hoc Networks: A Survey and Taxonomy," *Communication Surveys & Tutorials,* vol. 11, no. 4, pp. 19-41, December 2009.

[2] H. Moustafa, S. Senouci, and M. Jerbi, Vehicular Networks: Techniques, Standards and Applications, Boston: Auerbach Publications, July 2008.

[3] C. Huang, J. Chen, and Y. Chang, Telematics Communication Technologies and Vehicular Networks: Wireless Architectures and Applications, December 2009.

[4] Y. Toor, P. Muhlethaler, A. Laouiti, and A. De La Fortelle, "Vehicle Ad Hoc Networks: Applications and Related Technical Issues," *Communication Surveys & Tutorials,* vol. 10, no. 3, pp. 74-88, September 2008.

[5] R. Popescu-Zeletin, I. Radusch, A. Rigani and I. Radusch,, Vehicular-2-X Communication, State-of-the-Art and Research in Mobile Vehicular Ad Hoc Networks, Springer, May 2010.

[6] M. Lusheng and D. Karim, "A Survey of IEEE 802.11p MAC Protocol," September 2011.

[7] *IEEE 1609 - Family of Standards for Wireless Access in Vehicular Environments (WAVE),* September 2009.

[8] T. Issariyakul and E. Hossain, Introduction to Network Simulator NS2, London: Springer Science+ Business Media, July 2012.

[9] T. Henderson, G. Riley, and S. Floyd, "ns-3 Project Goals," *Proceeding from the 2006 Workshop on ns-2: the IP network simulator,* vol. 10, October 2006.

[10] R. Barr, *JiST - Java in Simulation Time User Guide,* September 2003.

[11] R. Barr, *SWANS - Scalable Wireless Ad hoc Network Simulator User Guide,* March 2004.

[12] A. Varga and R. Hornig, "An Overview of the OMNeT++ Simulation Enviroment," in *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems*, March 2008.

[13] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, "SUMO - Simulation of Urban MObility-an Overview," October 2011.

[14] C. Gawron, "Simulation-Based Traffic Assignment: Computing User Equilibria in Large Street Networks," February 1999.

[15] F. Karnadi, Z. Mo, and K. Lan, "Rapid Generation of Realistic Mobility Models for VANET," in *Wireless Communications and Networking Conference, 2007 (WCNC 2007). IEEE*, Kowloon, March 2007.

[16] F. Martinez, J. Cano, C. Calafate, and P. Manzoni, "Citymob: a Mobility Model Pattern Generator for VANETs," in *Communications Workshops 2008 (ICC Workshops' 08)*, 2008.

[17] J. Harri and F. Filali, "VanetMobiSim - Vehicular Ad Hoc Network Mobility Extension to the CanuMobiSim Framework," 2006.

[18] J. Harri, M. Flore, F. Filali, and C. Bonnet, "Vehicular Mobility Simulation with VanetMobiSim," *Simulation,* vol. 87, no. 4, pp. 275-300, April 2011.

[19] M. Piorkowski, M. Raya, A. Lugo, P. Papadimitratos, M. Grossglauser, and J. Hubaux, "TraNS: Realistic Joint Traffic and Network Simulator for VANETs," *ACM SIGMOBILE*

Mobile Computing and Communications Review, vol. 12, no. 1, pp. 31-33, January 2008.

[20] R. Mangharam, D. Weller, R. Rajkumar, P. Mudalige, and F. Bai, "GrooveNet: A Hybrid Simulator for Vehicle-to-Vehicle Networks," in *Third Annual International Conference on Mobile and Ubiquitous Systems: Networking & Services, 2006. IEEE*, July 2006.

[21] K. Konishi, K. Maeda, K. Sato, A. Yamasaki, H. Yamaguchi, K. Yasumoto, and T. Higashino, "MobiREAL Simulator --Evaluating MANET Applications in Real Enviroments," in *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems 2005, (MASCOTS 2005)*, September 2005.

[22] S. Wang and C. Lin, "NCTUns 5.0: A Network Simulator for IEEE 802.11p and 1609 Wireless Vehicular Network Researches," in *Vehicular Technology Conference, 2008. VTC 2008-Fall.*, September 2008.

[23] S. Wang, C. Chou, C. Lin, and C. Huang, "The Protocol Developer Manual for the NCTUns 6.0 Network Simulator and Emulator," National Chiao Tung University, Department of Computer Science, Network and System Laboratory, Tajwan, January 2010.

[24] S. Wang, Y. Chiu, Y. Tzeng, M. Hsu, Y. Cheng, W. Liu, and T. Ho, "NCTUns 4.0: An Integrated Simulation Platform for Vehicular Traffic Communication, and Network Researches," in *Vehicular Technology Conference, 2007. VTC-2007 Fall.*, Baltimore, October 2007.

[25] S. Wang and Y. Huang, "NCTUns Distributed Network Emulator," *Internet Journal,* vol. 4, no. 2, pp. 61-94, June 2012.

[26] S. Wang and H. Kung, "A New Methodology for Easily Constructing Extensible and High-Fidelity TCP/IP Network Simulators," *Computer Networks,* vol. 40, no. 2, pp. 257-278, October 2002.

[27] S. Wang, C. Chou, C. Huang, C. Hwang, Z. Yang, and C. Chiou, "The Design and Implementation of the NCTUns 1.0 Network Simulator," *Computer Networks,* vol. 42, no. 2, pp. 175-197, June 2003.

[28] S. Wang, C. Chou, and C. Lin, "The GUI User Manual for the NCTUns 6.0 Network Simulator and Emulator," National Chiao Tung University, Department of Computer Science, Network and System Laboratory, Tajwan, January 2010.

[29] S. Wang, C. Chou, and C. Lin, "The Design and Implementation of the NCTUns Network Simulation Engine," *Simulation Modelling Practice and Theory,* vol. 15, no. 1, pp. 57-81, August 2009.

[30] S. Wang and Y. Lin, "NCTUns Network Simulation and Emulation for Wireless Resource Management," *Wiley Wireless Communications and Mobile Computing,* vol. 5, no. 8, pp. 899-916, December 2005.

[31] S. Wang and C. Chou, "NCTUns Tool for Wireless Vehicular Communication Network Researches," *Simulation Modelling Practice and Theory,* vol. 17, no. 7, pp. 1211-1226, August 2009.

[32] F. Martinez, C. Keong Toh, J. Cano, C. Calafate, and P. Manzoni, "A Survey and Comparative Study of Simulators for Vehicular Ad Hoc Networks (VANETs)," *Wireless Communications and Mobile Computing,* vol. 99, no. 7, pp. 1189-1212, October 2009.

[33] L. Acosta and D. Hernandez, *Propuesta de un Conjunto de Benchmarks para Evaluar el Desempeño de Simuladores de Red en el Area de Redes Vehiculares,* Tesis, Universidad Central de Venezuela, Mayo 2013.

[34] E. Gamess, L. Acosta, and D. Hernández, "Analyzing Routing Protocol Performance Versus Bitrate in Vehicular Networks," in *Global Information Infrastructure and Networking Symposium (GIIS), 2012*, Choroni, Aragua, Venezuela, December 2012.

[35] K. Dalal, P. Chaudhary, and P. Dahiya, "Performance Evaluation of TCP and UDP Protocols in VANET Scenarios using NCTUns-6.0 Simulation Tool," *International Journal of Computer Applications,* vol. 36, no. 6, pp. 6-9, December 2011.

[36] X. Campos and D. Pastor, *Performance Evaluation of Manhattan Downtown Scenarios for Vehicular Ad Hoc networks With CityMob and NCTUns,* Tesis, Universitat Politècnica de Catalunya, June 2011.

[37] K. Dalal, P. Chaudhary, and P. Dahiya, "Performance Evaluation of AODV and ADV Protocols in VANET Scenarios," *International Journal of Computer Applications,* vol. 3, no. 1, pp. 50-55, February 2012.