ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Trabajo Fin de Máster

Máster de Tecnologías de la Información y las Comunicaciones 2009-2010

Evaluación de herramientas de simulación de Redes Vehiculares

AUTOR: Rocío Murcia Hernández
DIRECTOR: Joan García Haro
CODIRECTOR: Esteban Egea López

Septiembre / 2010

Autor	Rocío Murcia Hernández
E-mail del Autor	Rocio.murcia@upct.es
Director(es)	Joan Garcia Haro, Esteban Egea López
E-mail del Director	Joang.haro@upct.es
Codirector(es)	Esteban Egea López
Título del PFC	Evaluación de herramientas de simulación de Redes Vehiculares
Descriptores	Redes Vehiculares, Simulación, Supercomputación

Resúmen

En el **capítulo I** se lista de una manera sencilla cuáles son los principales "ingredientes" que forman las redes vehiculares, haciendo que la idea de VANET así como los elementos importantes para su entendimiento sean claros para el lector. En el **capítulo II** la idea es la de introducir una potente plataforma de trabajo que puede servir de base para lanzar simulaciones de redes vehiculares que requieran una gran cantidad de recursos para su resolución y cómputo. En el **capítulo III** repasaremos los principios de la simulación vehicular estableciendo a continuación una clasificación para las herramientas de simulación VANET. En el **capítulo IV** de este proyecto "bucearemos" por el mundo de simulación del tráfico rodado para introducirnos de lleno en una herramienta ampliamente conocida, SUMO. El **capítulo V** de este proyecto se centra en el análisis de una prometedora herramienta de simulación para la comunicación intervehicular, Veins.

Titulación	Máster de Tecnologías de la Información y las Comunicaciones 2009-2010
Fecha de Presentación	Octubre- 2010

Agradecimientos

Dile a la mañana que se acerca mi sueño que lo que se espera con paciencia se logra – Juan Luis Guerra (Bachata en Fukuoka)

*A **Joan García Haro** por rescatarme de Deutschland y darme una oportunidad vehicular. A **Antonio Urbina** por contestar los mail desde cualquier parte del mundo. A **Esteban Egea** porque siempre te hace pensar. A **Juanba Tomás** por ser paciente comunicador de su conocimiento y gran compañero. A **Carolina García** por su brillante pensamiento matemático y su apoyo incondicional. A **Jesús Vidal** por tener siempre la solución, es un gran ingeniero. A todo el **laboratorio IT-2** por el buen ambiente que se respira. A **Pablo Cases** por hacerme ver que esto de la Universidad me gusta. A **Christoph Sommer** por su capacidad docente y por compartir su trabajo. A Robert Nagel por acogerme en la TUM de nuevo. A mi **familia** por creer en mí. A **Néstor** por sacar lo mejor de mí, por hacerme ver que un océano no es nada.*

Índice

Sumario 6

I- INTRODUCCIÓN 7

I.I) Entendiendo las redes vehiculares 8

I.I.I) Pensamientos vehiculares 8

I.I.II) Aplicaciones VANET 10

I.I.III) Datos de la realidad vial 16

I.II) Proyectos europeos 17

I. III) Los retos VANETS 19

I.III.I) Los principales retos tecnológicos 20

I.III.I.) Protocolos, arquitecturas y estándares 20

I.III.II) Los principales retos socio-económicos 22

I.IV) ¿Por qué la simulación juega un papel principal? 23

I.IV.I) Modelos de simulación 23

I.IV.II) Sobre la simulación de tráfico rodado 24

I.V.III) Simuladores de tiempo vs. eventos 25

II-PLATAFORMAS PARA LA SIMULACIÓN 27

II. I) El Supercomputador Ben-Arabí 28

II. II) Pinceladas para la computación en paralelo 30

II.II.I) Mecanismos de programación paralelizada: MPI y OpenMP 31

II. II.II) Primeros resultados de computación paralela en un modelo matemático 35

III- SIMULADORES OPENSOURCE 39

III.I) Los pilares de la simulación vehicular 40

III.II) Taxonomía de los simuladores vehiculares 41

III.II.II) Simuladores aislados VANET 42

III.II.II) Simuladores integrados VANET 43

III.II.III) Simuladores híbridos VANET 44

III.III) Clasificación actual de herramientas OpenSource 44

III.III.I) Simuladores de Tráfico 44

III.III.II). Network Simulators 50

III.III.III) Simuladores integrados VANET 51

III.III.IV) Simuladores híbridos VANET. 53

IV-SUMO, EL SIMULADOR DE TRÁFICO RODADO 55

IV. I.) ¿Cómo es un simulador de tráfico rodado? **56**

IV. II) Instalación y ejecución de SUMO **59**

IV. II.I) Los paradigmas del software **60**

IV. II.II) Aplicaciones incluidas en SUMO **60**

IV. II.III) Convenio de nombres para los archivos de configuración **61**

IV. II.IV) Datos necesarios para comenzar una simulación en SUMO **61**

IV.III) Sumo: generando escenarios paso a paso **62**

IV.III.I Generación de networks **62**

IV.III.II Generación de rutas **83**

IV.III.III Medidas de simulación **91**

IV.IV) Conclusiones **100**

V-ANÁLISIS DE UN SIMULADOR PARA VANETS **101**

IV. I) Vehicles in Network Simulation, Veins **102**

IV.II) OMNeT++ 4 y su papel como simulador de redes **102**

IV. III) Hacia la bidireccionalidad de las herramientas **103**

IV. III.I) Interfaz TraCi **104**

IV. III.II) Los mensajes de intercambio **106**

IV. IV) La arquitectura Veins **107**

V.V) Veins en marcha **108**

V.VI) Escenarios y resultados **128**

Conclusiones y trabajo futuro **134**

Bibliografía **138**

Relación de Figuras **142**

Sumario

El trabajo llevado a cabo en este proyecto fin de Máster se organiza de la siguiente manera:

En el **capítulo I** se pretende listar de una manera sencilla cuáles son los principales "ingredientes" que forman las redes vehiculares, haciendo que la idea de VANET así como los elementos importantes para su entendimiento sean claros para el lector y se identifiquen como un concepto tecnológico cercano y conocido. Primero pasaremos a entender qué son las redes vehiculares, respondiendo a preguntas como ¿Qué son las *vanets*? ¿Qué aplicaciones ofrecen? Por lo que también ilustraremos con datos de la vida cotidiana como estas redes pueden ser de gran ayuda para mejorar la realidad vial de nuestros días y de un futuro no tan lejano. Al tratarse de un tema de candente actualidad, tras presentar los distintos proyectos europeos enfocados al desarrollo VANET, así como sus principales retos nos centraremos en la simulación de estas redes y en porqué resulta tan importante la simulación para poder obtener soluciones realistas y pragmáticas haciendo un uso eficiente de los recursos, tema central que nos ocupa en este proyecto. En el **capítulo II** de este proyecto se presentan conceptos como la supercomputación, superordenadores, aceleración de cómputo, técnicas de programación paralela, etc. La idea es la de introducir una potente plataforma de trabajo que puede servir de base para lanzar simulaciones de redes vehiculares que requieran una gran cantidad de recursos para su resolución y cómputo. Después de la necesaria introducción teórica pasaremos a ver un primer caso práctico en el cual se utilizan de forma casi natural las técnicas de paralelización en el ámbito vehicular. En el **capítulo III** repasaremos los principios de la simulación vehicular estableciendo a continuación una clasificación para las herramientas de simulación VANET. Llegando por último a una clasificación de los simuladores *opensource* y *freeware* disponibles para la comunidad investigadora. En el **capítulo IV** de este proyecto "bucearemos" por el mundo de simulación del tráfico rodado para introducirnos de lleno en una herramienta ampliamente conocida y utilizada por la comunidad académica internacional que se encuentra continuamente en proceso de desarrollo y que promete brindarnos grandes alternativas de trabajo para la modelización del tráfico vial, SUMO. El **capítulo V** de este proyecto se centra en el análisis de una prometedora herramienta de simulación para la comunicación intervehicular. Un software de tipo híbrido que conecta elegantemente por un lado al programa encargado de la simulación de tráfico vehicular SUMO y por otro al simulador de redes OMNeT, mediante una interfaz desarrollada para tales efectos. Veremos pues su arquitectura, instalación, detalles de ejecución así como alguno de los resultados que pueden obtenerse.

Para finalizar tenemos las conclusiones extraídas de esta proyecto así como posibles trabajos futuros.

I-INTRODUCCIÓN

En este primer capítulo se pretende listar de una manera sencilla los "ingredientes" que forman las redes vehiculares, haciendo que la idea de VANET así como los elementos importantes para su entendimiento sean claros para el lector y se identifiquen como un concepto tecnológico cercano y conocido. Primero pasaremos a entender que son las redes vehiculares, respondiendo a preguntas como ¿Qué son las *vanets*? ¿Qué aplicaciones ofrecen? Por lo que también ilustraremos con datos de la vida cotidiana como estas redes pueden ser de gran ayuda para mejorar la realidad vial de nuestros días y de un futuro no tan lejano. Al

tratarse de un tema de candente actualidad, tras presentar los distintos proyectos europeos enfocados al desarrollo VANET, así como sus principales retos nos centraremos en la simulación de estas redes y en porque resulta tan importante simular para poder obtener soluciones realistas y pragmáticas haciendo un uso eficiente de los recursos, tema central que nos ocupa en este proyecto.

I.I) Entendiendo las redes vehiculares

I.I.I) Pensamientos vehiculares

Conducir, además de caminar, hablar y comer, es la habilidad más ampliamente ejecutada en el mundo de hoy y posiblemente la más desafiante. Esto afirmaba Richard Rothery, uno de los pioneros dentro de la investigación en cuestiones de transporte, ya en el año 1968.

Es por esto que podemos entender como el reto del transporte vial y los consecuentes problemas del mismo, desde la aparición del primer automóvil de vapor en el 1769 por la calles de Paris diseñado por Nicolas-Joseph Cugnot (1)), ha ido creciendo de manera exponencial. Hoy en día nos encontramos con grandes problemas en términos de congestión, seguridad e impacto ambiental, así como con la ambición del ser humano de tener cada vez una vida más fácil, intentando que las máquinas sean el catalizador parar mejorar la calidad de vida en todos los sentidos. Esto lleva a una superación tecnológica cada vez más incipiente y por supuesto esta onda expansiva está muy presente en todo lo relacionado con el transporte vehicular.

En la última década, numerosos esfuerzos han tratado de mitigar los problemas derivados del tráfico rodado y algunas de las soluciones encontradas son por ejemplo: la obtención de información sobre el tráfico y situaciones de riesgo a través de la banda de radio FM, señales con mensajes variables en función de las condiciones en las carreteras espaciadas a pocos kilómetros de separación o en puntos estratégicos, los sistemas de telepeaje parar cobrar las cuotas casi sin interrupción del flujo de tráfico (2). Al mismo tiempo, los vehículos proporcionan una asistencia al conductor así como mecanismos de protección cada vez más eficaces.

Los controles a bordo, fuentes de información actualizadas sobre el estado del vehículo, los mecanismos de seguridad pasiva para proteger a los pasajeros y el vehículo contra las adversas condiciones de conducción (por ejemplo, sistemas de frenos antibloqueo), sistemas de navegación, brújulas, los radares de estacionamiento delantero y trasero, y las cámaras son las más comunes entre las tecnologías de sensores autónomos. Más allá de estas tecnologías, apoyándose en tecnologías heterogéneas (por ejemplo, cámaras de carretera) se encuentran sistemas más complejos que permiten la gestión de flotas y la recopilación de información de tráfico en tiempo real.

Las últimas novedades tecnológicas, especialmente en computación móvil, comunicaciones inalámbricas, y la teledetección están impulsando los sistemas de transporte inteligentes (ITS) a que den un paso más allá. Los vehículos se consideran como sistemas de computación

altamente sofisticados ya que constan de varias computadoras y sensores a bordo, cada uno dedicado a una parte de la operación del automóvil.

La parte innovadora es la inclusión de nuevas comunicaciones inalámbricas, informática y capacidades de detección formando entre los distintos vehículos y entre éstos y la infraestructura que los rodea lo que conocemos como red, en este caso denominada vehicular.

De tal manera que interconectado los distintos vehículos no sólo se consigue la recolección de información sobre sí mismos y su entorno, sino que también se lograría un intercambio de datos en tiempo real con otros elementos cercanos al vehículo en cuestión/unidad de infraestructura de carretera.
En pocas palabras, las soluciones de radio para la comunicación pueden operar más allá de las limitaciones de la línea de visión del radar permitiendo lo que conocemos como comunicación cooperativa (3).

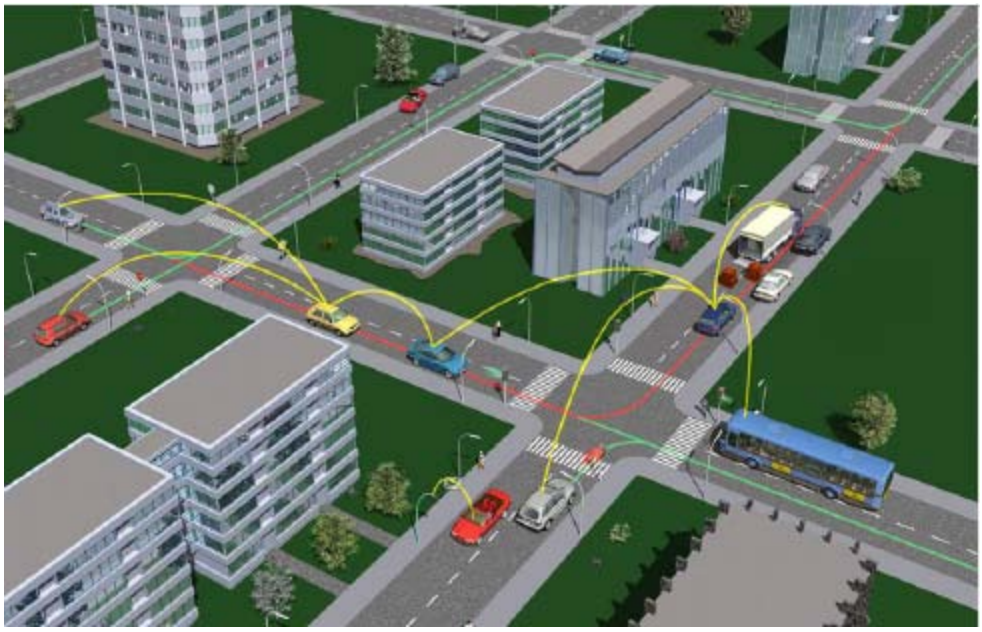


Figura 1 Escenario interurbano VANET

Los vehículos y la infraestructura podrán así trabajar en simbiosis para percibir las situaciones potencialmente peligrosas en un espacio ampliado es por esto que concluimos con que una comunicación adecuada bien intervehicular (V2V) o vehículo a infraestructura (V2I) se contribuirá a mejorar notablemente la seguridad vial, la fluidez del transporte rodado, y a aumentar el nivel de confort de conductores y pasajeros.
En los últimos años, hemos sido testigos de un gran aumento en la investigación y el desarrollo en el ámbito de las redes vehiculares. El baile vehicular está orquestado por diferentes directores, por una lado tenemos la adopción de las tecnologías IEEE 802.11, los grandes fabricantes de vehículos que han asumido de manera natural las tecnologías de la información para hacer frente así a una necesidad de mayor seguridad, mayor preocupación por el medio ambiente y al incremento de la comodidad en los vehículos; y por último los gobiernos

nacionales/regionales han asignado el espectro radioeléctrico vehicular para la comunicación inalámbrica.
Aunque las redes celulares permiten la comunicación de voz y transmisión de información simple correctamente, así como proporcionar servicios de valor añadido a los conductores y los pasajeros, éstas no están preparadas para determinados tipos de comunicaciones como son las del tipo V2V o V2I. Las redes vehiculares ad hoc (VANET), ofrecen la comunicación directa entre los vehículos y hacia-desde las unidades (de infraestructura) en carretera (RSU), ofreciendo la posibilidad de enviar y recibir avisos de peligro o información sobre la situación actual del tráfico con un mínimo de latencia.

I.I.II) Aplicaciones VANET

Los cuatro grandes grupos donde se pueden enmarcar las distintas aplicaciones de las redes vehiculares son las siguientes (4):

	Situación o propósito	Ejemplo de aplicación
I Seguridad Activa	1. Carreteras peligrosas	1 Advertencia de velocidad por curvas, 2 puente debajo de alerta, 3 Advertencia sobre violación de transito, luces o señales de parada.
	2. Condiciones anormales del trafico	1 Vehículos basados en las condiciones de advertencia, 2 Infraestructura basadas en condiciones de advertencia, 3 potenciador de visibilidad, 4 trabajo en Zona de advertencia.
	3. Peligro de colisión	1 Advertencia de ángulo muerto, 2 aviso de cambio de carril, 3 intersección de advertencia de colisión, de avance y de alerta de colisión trasera, 4 Luces de emergencia, freno electrónico de emergencia 6 advertencia de colisión de ferrocarril, 7 advertencia acerca de los peatones que cruzan
	4. Choques inminentes	1 Detención de pre-accidente

	5. Incidente ocurrido	1 Después de una colisión de alerta, 2 advertencia de avería, 3 SOS servicio
II Servicio publico	1. Respuesta de emergencia	1 Aproximaciones de alerta de emergencia del vehículo, 2 vehículo de emergencia señal de anticipación, 3 vehículos de emergencia en la escena de alerta
	2. Apoyo a las autoridades	1. Placa electrónica, 2 licencia de conducir electrónica, 3 inspección de vehículos de seguridad, 4 seguimiento de vehículos robados
III Mejora de la conducción	1. Mejora de conducción	1 Asistente de carretera, 2 asistente de girar a la izquierda, 3 control de velocidad, 4 señalización de abordo de vehículo, 5 gestión de transmisión adaptativa
	2. Eficiencia del trafico	1 Notificación de accidentes 2 control inteligente del flujo de tráfico, 3 Guía de ruta y navegación mejorada, 4 descraga y actualización, 5 localizacion de plazas de aparcamientos
IV Negocios y entretenimientos	1. Mantenimiento de vehículos	1 Diagnósticos inalámbricos, 2 actualización de software, 3 aviso de seguridad
	2. Servicios móviles	1 Servicio de internet, 2 mensajeria instantánea, 3 notificacion de puntos de interés
	3. Soluciones empresariales	1. Gestión de flotas, 2 procesamiento de alquiler de coches, 3 área de control de acceso, 4 seguimiento de la carga

Tabla 1 Grupos de aplicaciones VANET

SEGURIDAD ACTIVA

Las aplicaciones de seguridad activa son consideradas como las típicas y más deseables dentro del grupo de aplicaciones para las VANETS teniendo un impacto directo en la seguridad vial. La intención básica es que la conducción sea más segura mediante la comunicación, lo que puede significar que a los conductores se les advierta sobre una situación peligrosa o incluso que el vehículo pueda tratar de evitar un accidente o reaccionar adecuadamente en caso de un accidente inminente. Estas aplicaciones se categorizan de acuerdo al nivel de peligro:

- El peligro bajo esta asociado a situaciones como por ejemplo la existencia de curvas en la carretera, éstas son estáticas y por lo tanto previsibles.
- El grado de peligro elevado se da cuando ocurren situaciones anormales de circulación y cambio en las condiciones de las carreteras luego estas situaciones son de tipo dinámico y no tan altamente previsibles.
- El peligro es alto cuando las aplicaciones intentan evitar colisiones (por ejemplo, si un vehículo frena de manera brusca en situaciones de tráfico denso). Por último, cuando el peligro se ha convertido en un incidente, es importante advertir a los vehículos aproximándose o llamar para pedir ayuda.

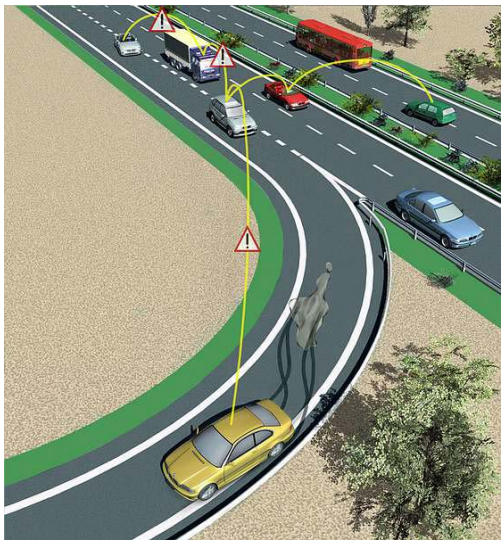


Figura 2 Ejemplo de aplicación de seguridad activa: frenada en curva peligrosa

SERVICIO PÚBLICO

Las redes vehiculares se destinan también a apoyar la labor de servicio público como la policía o las unidades de emergencia. Ejemplos destacados de esta categoría son el apoyo de los vehículos de emergencia por las sirenas o la señal de las capacidades preferentes. El uso de estas aplicaciones, puede hacer que los vehículos de emergencia alcancen su destino mucho más rápido que en la actualidad lo hacen.

MEJORA DE CONDUCCIÓN

Esta categoría incluye las aplicaciones que tratan de mejorar o simplificar la conducción por medio de la comunicación. La idea cuenta con escenarios microscópicos en los alrededores inmediatos de un vehículo, así como la optimización de la eficiencia del tráfico a nivel macroscópico. En el primer caso, las solicitudes se destinan a ayudar al conductor en situaciones de tráfico estándar, como entrar en una autopista. En el segundo caso, la eficiencia en el flujo del tráfico se dirige hacia un área mayor. Esto puede significar que un accidente de alerta se difunda en un área más grande para informar a los vehículos sobre el posible obstáculo para que los conductores puedan tomar una ruta diferente. Otro servicio es la difusión de información sobre el estacionamiento, o incluso la reserva de una plaza de aparcamiento.

MOBILE BUSINESS Y ENTRETENIMIENTO

Aquí, la atención se centra en la prestación de servicios a los clientes, automatización de tareas sobre el vehículo o de las solicitudes de pago, tales como la descarga de música, gestión de flotas, mantenimiento del vehículo, o el pago de estacionamiento o de uso de la carretera.

Veamos a continuación algunas de las posibles aplicaciones que se pueden dar en la vida cotidiana las aplicaciones VANET.

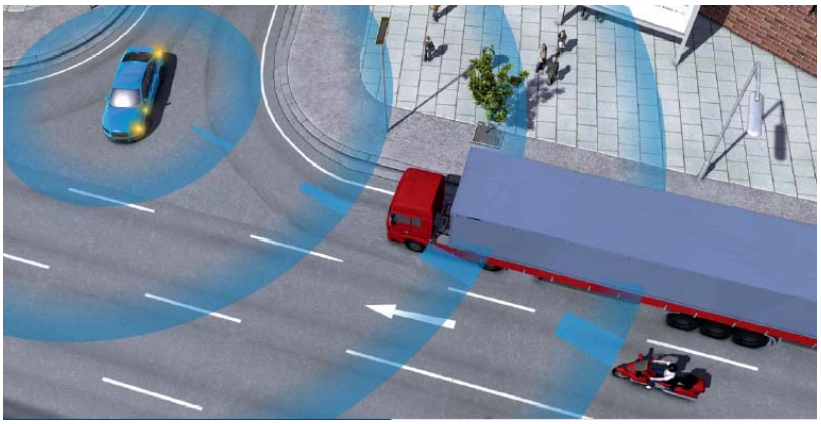


Figura 3 Aplicación VANET para motocicletas

La vulnerabilidad de las motocicletas: los accidentes de motocicletas se deben en primer lugar al error humano, y más concretamente a no ver venir la motocicleta o a una mala interpretación de distancia y velocidad. El acercamiento de advertencia para motocicletas es un sistema de alerta que tiene por objeto apoyar los conductores y motoristas para compensar estos errores de percepción. El otro vehículo es capaz de identificar la motocicleta, y ambos vehículos pueden determinar si una situación crítica tiene probabilidad de que ocurra. Mientras que la motocicleta se encuentra en la carretera principal, un coche se acerca a la intersección de la mano derecha; debido a una obstrucción en la línea directa de visión, el conductor del automóvil pasa por alto a la moto que se aproxima. Cuando entra en la intersección, el conductor del vehículo recibe una advertencia por parte de la motocicleta que se acerca. El piloto también recibe una advertencia en su HMI (*Human-Motor Interface*) y de esta manera se evita un posible choque entre ambos.

Vehículos de emergencia: cuando se trata de situaciones que afectan a la seguridad de vidas es crucial cada minuto y cada segundo. Los usuarios de la carretera se ven obligados a dejar paso a los vehículos de emergencia como ambulancias o coches de policía. Sin embargo, al oír la sirena no siempre se discierne la ubicación y la dirección adecuada del vehículo lo suficientemente rápido como para reaccionar adecuadamente.

Esta aplicación indica al conductor de los vehículos la manera de proceder ante la llegada de un vehículo de emergencia, ayudando a despejar la vía incluso cuando la sirena y las luces del coche de emergencia aún no puede ser audibles o visibles.

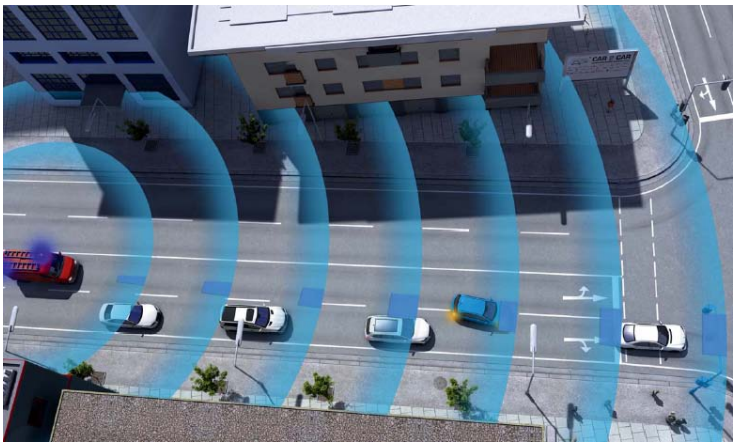


Figura 4 Aplicación VANET para accidentes

Atención: accidente en las inmediaciones: si un vehículo detecta un incidente bien por parte del conductor, visualmente con las luces de emergencia, o con sensores de choque o diagnóstico a bordo esta información será añadida a su lista de mensajes para ser enviados periódicamente a sus vecinos más próximos. Adicionalmente el vehículo involucrado en el accidente y dañado enviará lo que se conoce como *Decentralized Environmental Notification Message*, siendo esta información distribuida entre todos los vehículos que se encuentren en una región más extensa, ayudando así a una detección más temprana del foco del problema y constituyendo una gran ayuda en el caso de tener que encontrar nuevas rutas. Los accidentes y los frenazos forzados son una causa de peligro para todas las personas envueltas en la situación, ayuda o bien los vehículos que ser aproximan. En el caso de encontrarse en una curva con la línea de visión directa obstruida constituye aun más si cabe una posible casusa de accidente en cadena es por esto que un aviso con el tiempo suficiente para que los vehículos colindantes reaccionen reduce significativamente el peligro.

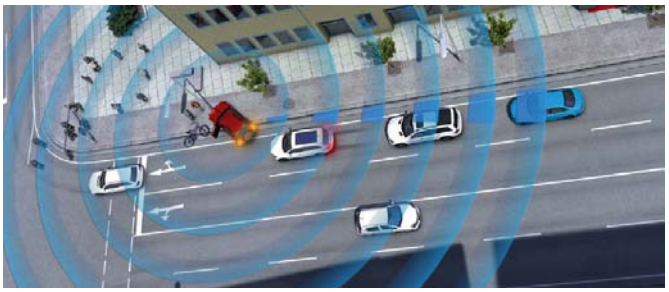


Figura 5 Aplicación VANET para Vehículos de emergencia

Peligro por Obras: en esta aplicación se presenta un claro ejemplo de comunicaciones V2I, donde se advierte a los vehículos que se aproximan a una zona en obras de la situación en el momento, Si bien hoy en día existen señales temporales de tráfico e incluso personas encargadas de indicar que existe una zona de peligro, puede ocurrir que el conductor se dé cuenta demasiado tarde. En el supuesto caso de tener una *Road Side Unit* funcionando, ésta se encargaría de comunicarse directamente con el conductor enviando información de tráfico, obras en las carreteras, restricciones en las mismas, consejos de circulación etc. De tal manera que se aumentaría claramente la seguridad vial y la eficiencia del tráfico evitando

embotellamientos en puntos clave al saber de antemano que es posible tomar una ruta alternativa para llegar a nuestro destino.

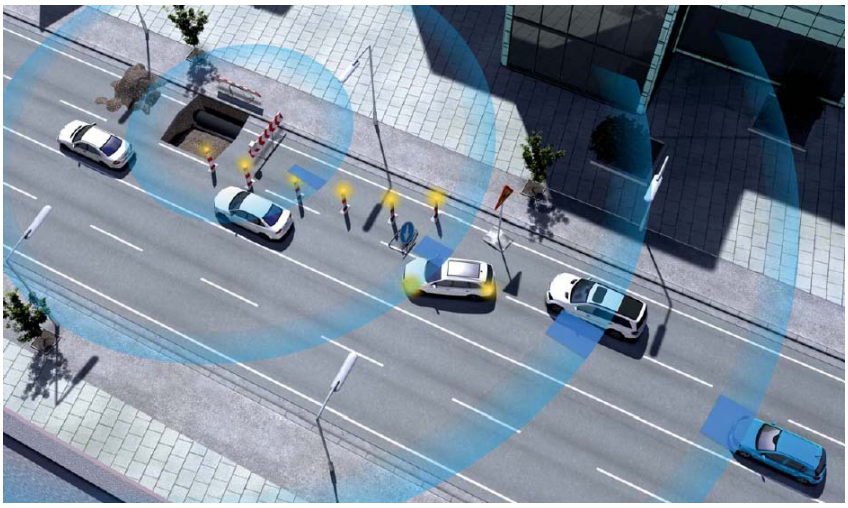


Figura 6 Aplicación VANET para obras

I.III) Datos de la realidad vial

En la siguiente tabla se pueden ver las situaciones de accidentes que más se dan en Alemania Y los cuales podrían bajar su porcentaje con sistemas de ayuda a la conducción a través de las redes vehiculares (5).
Para paliar estas situaciones se proponen los sistemas que siguen, si bien algunos de ellos ya están siendo implementados en los vehículos.

Problema (1): colisión con otro vehículo que está entrando en una intersección
Solución (1): sistemas de ayuda para el frenado y asistente dedicado a las intersecciones.

Problema (2): colisión con otro vehículo mediante aproximación bien que se encuentra parado, comenzando a circular o frenando.
Solución (2): sistemas de ayuda para el frenado y *tempomat* (controlador para la velocidad de crucero).

Problema (3): colisión frontal con otro vehículo.
Solución (3): *lane departure warning system*, es un mecanismo diseñado para advertir a un conductor cuando el vehículo comienza a salirse de su carril (a menos que una señal de la vuelta es en esa dirección las autopistas y carreteras principales).

Problema (4): colisión entre vehículo y peatón.
Solución (4): sistemas de ayuda para el frenado.

Problema (5): colisión con vehículo que viaja en la misma dirección moviéndose lateralmente.
Solución (5): sistema de vigilancia del punto muerto y *lane departure warning system*.

Problema (6): salirse del carril hacia un lateral bien derecha o izquierda.
Solución (6): *lane departure warning system*.

Problema (7): colisión con elemento ajeno a la vía.
Solución (7): detector de obstáculos en la vía.

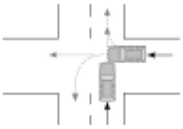
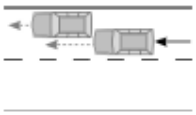
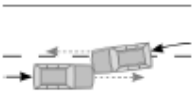
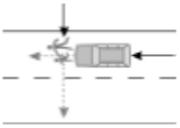
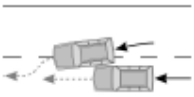
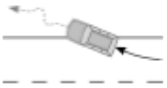
Most frequent accident situation ($n_{data\ pool}$=136,954) [100 %]		**Proportion**
(1) Collision with another vehicle which is turning into or crossing a road		34.5%
(2) Collision with another vehicle - moving forwards or waiting - which is starting, stopping or is stationary		22.2%
(3) Collision with another oncoming vehicle		15.5%
(4) Collision between vehicle and pedestrian		12.1%
(5) Collision with another vehicle moving laterally in the same direction		6.9%
(6) Leaving the carriageway to the right or left		6.3%
(7) Collision with an obstacle in the carriageway		0.1%

Figura 7 Datos de accidentes viales en Alemania para 2009

I.II) Proyectos europeos

Es por todos sabida la alta dependencia que tenemos al transporte en la actualidad. El incremento cada vez mayor del tráfico rodado genera serios problemas sociales, desde la congestión en las redes y zonas urbanas, el daño del medio ambiente y la salud pública, el desperdicio de energía y por encima de todo, el aumento desmesurado de accidentes (2). 40000 personas mueren cada año en las carreteras europeas sin incluir todas aquéllas que se encuentran afectadas de manera más leve. Las congestiones de tráfico afectan a un total del 10% de nuestras carreteras y suponen un gasto total de 50 billones de euros al año, un total del 0,5% del producto interior bruto (PIB) de la unión europea El transporte por carretera supone más de una cuarta parte del consumo total de energía de la UE, y todavía hay alrededor de 1,3 millones de accidentes en las carreteras de la UE cada año. A los conductores europeos de automóviles, un total de 300 millones, les gustaría poder conducir de forma más fácil con menos retrasos y menos riesgos de lesión. Es aquí donde las redes vehiculares entran en juego ya que los sistemas inteligentes pueden ayudar a resolver muchos de los problemas en el transporte por carretera. Éstos pueden apoyar a los conductores a evitar accidentes, e incluso llamar a los servicios de emergencia automáticamente en caso de accidente. También pueden ser utilizados en los sistemas electrónicos de gestión del tráfico o la optimización del rendimiento del motor, mejorando así la eficiencia energética y reduciendo la contaminación.

Debido a la importancia de estos objetivos tanto para el individuo como para las naciones, están en marcha varios proyectos o fueron recientemente terminados, y se han establecido múltiples consorcios para explorar el potencial de las VANET. Estos consorcios y proyectos involucran a varios sectores, incluyendo la industria del automóvil, los operadores, agencias de peaje, y otros proveedores de servicios.Estos proyectos se financian sustancialmente a través de los gobiernos nacionales. Los gobiernos nacionales también contribuyen a asignar espectro con licencia, generalmente en la banda de los 5.8/5.9-GHz y al menos en Japón en la banda de los 700MHz. Algunos de los principales proyectos europeos de actualidad se presentan a continuación, la tabla que sigue contiene información resumida sobre los proyectos más representativos, consorcios y grupos de trabajo. La cantidad de capital invertido así como el interés general por todas las grandes marcas de vehículos, los gobiernos y distintas universidades europeas dan pistas claras de la importancia de las redes vehiculares y del papel indiscutible que jugarán en la realidad vial de todos los que participamos de ella.
Los principales proyectos europeos se presentan en la tabla a continuación:

Nombre del proyecto	Periodo	Financiación	Descripción breve de los objetivos
AKTIV	2006-2010	Ministerio de economía y tecnología de Alemania	Diseño, desarrollo y evaluación de los sistemas de asistencia al conductor, conocimientos y tecnologías de la información eficaz del tráfico. http://www.aktiv-online.org/index.html
CityMobil	2006-2010	Unión Europea	Integración de los sistemas de transporte automatizado en el medio urbano, basado en las implementaciones de la vida real. http://www.citymobil-project.eu/

COOPERS	2007-2010	Unión Europea	Aplicaciones telemáticas para la infraestructura vial de cooperación de gestión del tráfico. http://www.coopers-ip.eu/
CVIS	2007-2011	Unión Europea	Multicanal de terminal con conexión permanente a Internet, arquitectura de comunicaciones abierta, posicionamiento mejorado, aplicaciones comerciales, kit d herramientas (modelos, directrices y recomendaciones) y hojas de rutas de implementación. http://www.cvisproject.org/
EVITA	2008-2011	Unión Europea	Comunicación intravehicular segura y confiable la arquitectura de las redes a bordo del automóvil para impedir las manipulaciones y proteger los datos sensibles dentro de un vehículo. http://evita-project.org/
HAVE-IT	2008-2011	Unión Europea	Realización de la visión a largo plazo de alto grado de automatización de conducción para el transporte inteligente. Desarrollar, validar y demostrar importantes pasos intermedios hacia la conducción altamente automatizada. http://www.haveit-eu.org
AFESPOT	2008-2012	Unión Europea	Creación de redes, localización exacta relativa, mapas locales de tráfico dinámicos, la evaluación basada en escenarios de aplicaciones de seguridad. estrategia de implementación sostenible http://www.safespot-eu.org/
SmartWay	2006-2010	Ministerio de Tierras, Infraestructura, Transportes y Turismo de Japón	Conducir los sistemas de seguridad de apoyo basado en la cooperación de vehículos de carretera http://www.mlit.go.jp/road/ITS/

Tabla 2 Proyectos europeos actuales (2)

I. III) Los retos VANETS

Como toda nueva tecnología a implantar las redes VANET se encuentran con una serie de obstáculos antes de poder ser implantadas, alguno de ellos se puede leer como siguen:

I.III.I) Los principales retos tecnológicos

En el concepto VANET, **no existe un coordinador para las comunicaciones**. Si bien existen aplicaciones que si implican que la infraestructura (V2I) sea un elemento activo de las comunicaciones, sin embargo se espera que la mayoría funcionen de manera descentralizada. Al no existir esta coordinación central o un protocolo tipo *handshaking* y dado que muchas aplicaciones realizaran *broadcasting* con los vehículos de su alrededor será expresamente necesaria la existencia de un canal de control compartido. Por lo que un punto clave para el desarrollo de las redes vehiculares será el binomio entre la existencia de un canal de control y el requerimiento del control distribuido (6).

El problema de los terminales escondidos y expuestos es por todos sabido. El control de acceso al medio juega un papel principal en el diseño de las VANETS. En cuestiones de disponibilidad así como en cuestión de soporte de los elementos aleatorios todo apunta al método CSMA ***Carrier Sense Multiple Access*** **(CSMA),** que es el que contempla el protocolo IEEE 802.11.

El **ancho de banda** de los canales previstos para las aplicaciones VANET se encuentra en rangos que comprende de los 10 a los 20 Mhz. En caso de existir una alta densidad de tráfico rodado, estos canales podrán verse fácilmente congestionados. También es de recibo considerar los problemas que acarrean técnicas como la sincronización multicanal, como es la interferencia co-canal, para el caso de un único emisor por vehículo en el caso de haber de más de un canal.

Siempre tendremos presente que hablamos de una red con una **topología dinámica** basada en la movilidad de los vehículos con el consecuente impacto en la propagación por radio. Y sin olvidar el funcionamiento para altas densidades de tráfico vehicular, donde el principal objetivo será que la red funcione correctamente con un alto nivel de fiabilidad y una baja latencia en las comunicaciones. No hay que olvidar el importante tópico acerca de la **seguridad,** por un lado todos los receptores quieren asegurarse de que la fuente que envía la información es fiable, mientras que por otro, la privacidad de estos puede verse violada.

I.III.I.) Protocolos, arquitecturas y estándares

Las capas de comunicación física y de acceso al medio (PHY/MAC) están basadas en el protocolo IEEE 802.11 con la función de coordinación distribuida (DCF). La calidad garantizada que da soporte al servicio no está garantizada, es por esto que originalmente la ASTM (*American Society for Testing and Material*) modificó el estándar 802.11a para ajustarlo así al entorno vehicular. Basado en este esfuerzo, el IEEE está hoy en día en proceso de estandarización del correspondiente protocolo 802.11p. Este protocolo está basado en la multiplexación OFDM, *orthogonal frequency-division Multiplexing*, mientras que la capa física utiliza canales de 10 Mhz al contrario que el estándar original 802.11a que los usa de 20 Mhz. Se consiguen por tanto unas tasas que van desde 3 hasta 27 Mb/s para cada canal. La banda de frecuencias donde se opera se encuentra en los 5.8/5.9 GHz El tipo de comunicaciones está basado en difusión (*broadcast*) de un salto (*one-hop*) y la capa MAC funciona bajo las líneas dictadas por el mecanismo de acceso al medio CSMA.
En lo que respecta a la priorización el concepto de EDCA *enhanced distributed channel access* introducido para el 802.11e se vuelve a utilizar con los valores de configuración propuestos en el estándar IEEE 1609.4.

La escalabilidad es una cuestión clave para la difusión de información evitando de este modo una tormenta y una sobresaturación de la red. Para asegurarse de que los vehículos entenderán los mensajes independientemente de sus marcas, el mensaje se establecerá del tipo: *Dedicated Short Range Communications Message* según el SAE j2735, mensajes que permiten intercambiar información acerca del estado del vehículo así como las advertencias de una forma estandarizada.

Desde el punto de vista de la pila de protocolos se plantean dos cuestiones:

- ¿Cómo aunar todos los elementos anteriores?
- ¿Qué aspectos se deben considerar para la implementación?

Por un lado, la red vehicular podría ser simplemente una extensión de Internet, aprovechando los métodos basados en *User Datagram Protocol / Internet protocol*, protocolo (UDP/IP); por otro lado existe una necesitad para fomentar las aplicaciones *VANE T centric*, por ejemplo las orientadas a la seguridad del tráfico y la eficiencia. Este tipo de aplicaciones controlan idealmente y acceden tanto a los parámetros de las capas PHY como a la capa MAC. Por tanto, se propuso una arquitectura doble en el llamado estándar IEEE 1609 WAVE.

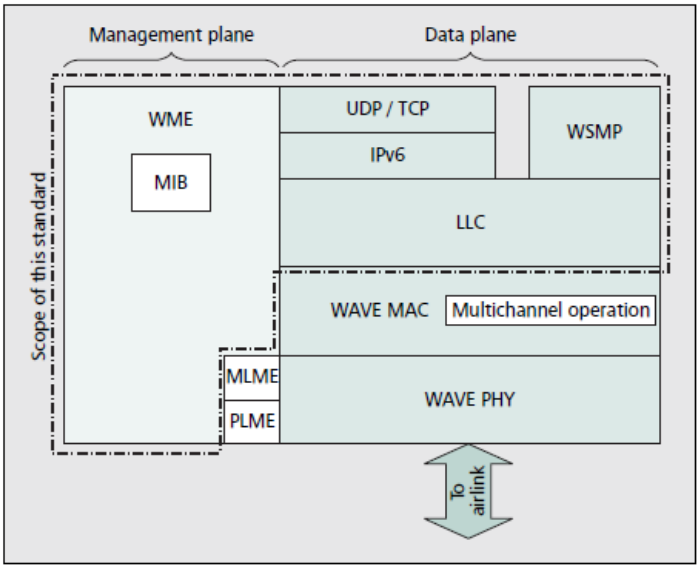


Figura 8 Esquema de entorno IEEE 1609

El entorno IEEE 1609 contiene las capas PHY y MAC correspondientes a IEEE 802.11p y proporciona dos pilas paralelas encima de éstas. Una para UDP/TCP sobre IPv6 y otra denominada WSMP, *Wave Short Message Protocol*. Con este protocolo WSMP, se pueden especificar algunos parámetros de bajo nivel de comunicación como por ejemplo la tasa de datos y el nivel de transmisión de potencia.
El entorno IEEE 1609 está formado por cuatro bloques funcionales:

1. *networking services* (1609.3)
2. *multi-channel operation* (1609.4)
3. *security* (1609.2)
4. *resource manager* (1609.1)

I.III.II) Los principales retos socio-económicos

En este mercado el valor añadido para un cliente de que exista una comunicación directa entre vehículos depende de que otros clientas decidan instalar el equipamiento necesario para la tecnología VANET. Luego la llave del éxito es convencer a los clientes pioneros para que instalen los equipos necesarios. Se están barajando varias opciones, desde la imposición legal de las mismas, ventajas con empresas de seguros, e incluso atractivas aplicaciones para el usuario. Por otro lado no olvidemos lo que respecta a las instalaciones relativas a las nfraestructuras, aquí entran en juego distintas comunidades, como son los operadores de telecomunicaciones y de carreteras, y de cómo la lucha de titanes no pinta sencilla bajo ningún concepto, este hecho queda comprobado con distintas experiencias de aquéllos que han intentado llevar a cabo experimentos de campo (7).

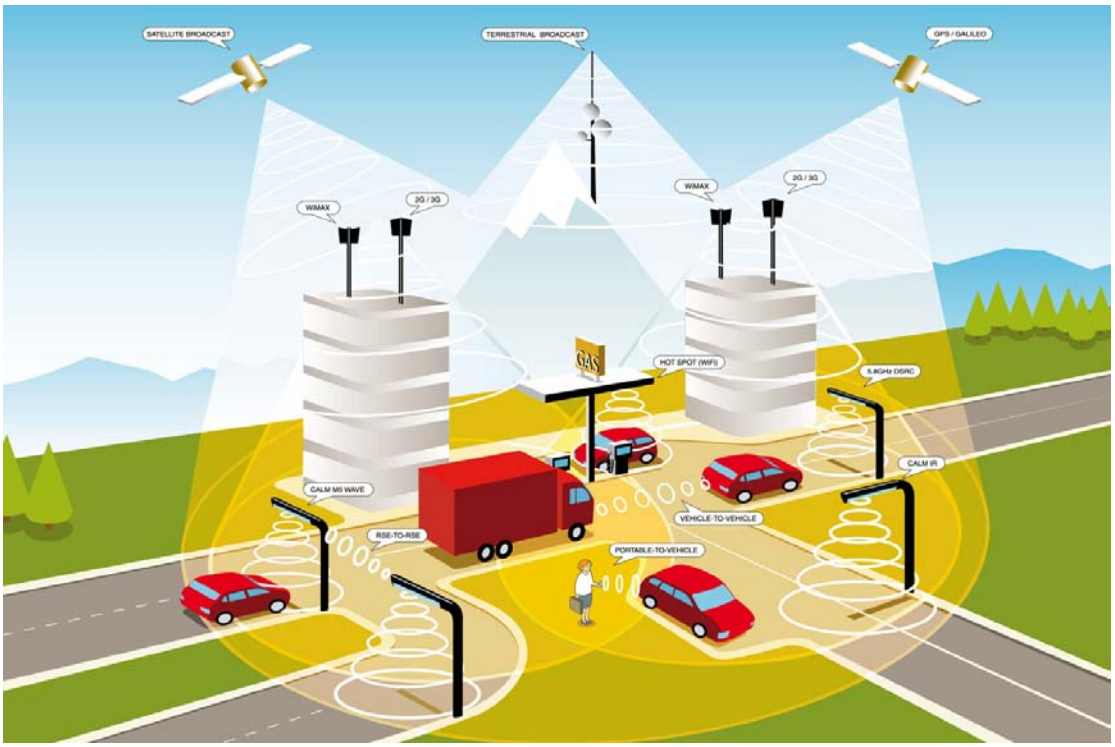


Figura 9 Escenario VANET y sus componentes

Si bien a fecha de hoy, es importante resaltar que gracias a los numerosos proyectos como los referidos anteriormente y a los esfuerzos por parte de numerosos actores, gobiernos, empresas y universidades están dado resultados importantes y suponiendo un notable impulso para las VANETS en su evolución futura. No olvidando que existen numerosos desafíos por delante antes de que veamos implementados como una realidad los sistemas de comunicaciones vehiculares, es por eso que los estudios de campo realizados por los proyectos como SAFESPOT son de suma importancia. Es necesario que se lleve a cabo una experimentación de campo a mayor escala para poder realizar una verificación y validación profunda que aporte una alta confiabilidad al sistema. Esto incluye no sólo la interconexión de datos y tecnologías de red sino también que las propias aplicaciones, en particular aquéllas con las más altas exigencias, funcionen de una manera fiable. Como meta a alcanzar está la de

asegurar el funcionamiento eficiente y eficaz, incluso en situaciones difíciles, aunque sean poco probables en la práctica. Mientras tanto, la integración de mecanismos sólidos de seguridad no debe ser descuidada, especialmente en lo que una arquitectura y protocolos se refiere, garantizando así la conexión intervehicular junto con la protección de la intimidad. Otros aspectos a considerar incluyen los financieros, jurídico, y cuestiones de organización. Por ejemplo algunas de las preguntas que están en el aire son; ¿cuál será el costo de implementación y cómo se cubrirán dichos gastos? ¿La implementación debe aprovechar los sistemas existentes y dispositivos portátiles, como teléfonos inteligentes?¿Cuál será el primer conjunto de aplicaciones desplegadas? ¿Cómo se legislaran todos estos cambios?
La innovación juega un papel primordial en cuestiones de mercado. Es importante que exista una penetración profunda de la solución tanto a nivel de vehículos, como de infraestructuras, sin este paso más allá los beneficios para el conductor estarán muy limitados y nadie está de acuerdo en pagar por algo ahora que algún día tendrá un valor en el futuro. Luego el despliegue de la tecnología para las comunicaciones vehiculares debe de ser algo realista donde los temas de responsabilidad deben estar claramente especificados. La estandarización como en todo este tipo de procesos es un ingrediente que no puede faltar y ésta emana de las contribuciones de los distintos proyectos, los consorcios industriales así como los comités y grupos de trabajo.

Finalmente, indicar que los sistemas cooperativos más avanzados serían probablemente los que están completamente automatizados. Por supuesto para ganarse la amplia aceptación de los usuarios es vital que estos sistemas autónomos demuestren un alto nivel de confianza, convenciendo a los usuarios de los grandes beneficios a nivel personal y de movilidad que estos sistemas pueden ofrecer.

I.IV) ¿Por qué la simulación juega un papel principal?

I.IV.I) Modelos de simulación

Para entender los principales retos y dificultades de las VANETS en comparación con otras redes *wireless* así como para llevar a cabo simulaciones que reflejen la realidad es muy importante la elección del **modelo de tráfico** intentando que contenga todas las propiedades propias del tráfico vehicular así como el comportamiento que **modela el canal radio**.

El diseño y análisis de los flujos realistas de tráfico vehicular se lleva desarrollando durante más de 50 años. Para las simulaciones de VANET los modelos microscópicos son los más ampliamente utilizados ya que proporcionan un comportamiento espacio-tiempo para los vehículos así como sus interacciones a un nivel individual. También se tienen que tener en cuenta las propiedades macroscópicas tales como la cantidad media de vehículos que atraviesan una determinada zona. Los puntos conflictivos, entre otros, a la hora de simular redes vehiculares son los que siguen:

• El acoplamiento entre flujo de tráfico y simuladores de redes.

• Modelado de las reacciones de los conductores a la información adicional proporcionada por las aplicaciones VANET.

• Existencia de referencias para hacer estudios de simulación así como la obtención de resultados comparables con la realidad.

• Manejo de plataformas de computación lo suficientemente potentes para soportar simulaciones de tráfico en tiempo real.

• Además, para los estudios relacionados con la seguridad, es necesaria la existencia de modelos de accidentes.

En lo que respecta el modelo de radio para el canal se tiene que extraído de las medidas del mundo real los modelos deterministas deben de evitarse ya que no captan todos los efectos probabilísticos del desvanecimiento (*fading*) a pequeña escala que tengan una repercusión especial en la recepción del paquete. La distribución de Nakagami (8) se ha propuesto como una buena candidata y se usa hoy en día ya que modela una gran variedad de las condiciones del canal. Si bien, algunos elementos medioambientales como por ejemplo las condiciones climáticas, o los edificios que rodean la red vehicular se omiten a la hora de relazar la simulación.

I.IV.II) Sobre la simulación de tráfico rodado

El volumen de tráfico vehicular en los últimos años ha superado de manera desbordante las capacidades de las carreteras en las distintas naciones, es por esto que debe entenderse la dinámica del flujo de tráfico y obtener una descripción matemática del proceso se ha convertido cada vez más es una necesidad de primer orden. Estas palabras corresponden a un artículo clásico publicado en 1959 por Greenberg (9), si bien éstas se pueden adoptar como declaraciones contemporáneas, ya que desde la década de los cincuenta la cantidad de tráfico ha aumentado vertiginosamente y el problema de los límites de las carreteras ha ido creciendo de la mano con dicho aumento.

Introducir modificaciones en una infraestructura vial e incluso predecir adecuadamente las rutas a elegir por los vehículos en una determinada zona de una urbe cualquiera no es tarea fácil de manejar sin las herramientas adecuadas. Si bien es sabido que el estudio de las carreteras y todos los elementos que en ellas se encuentran, se ha convertido en un área de gran relevancia llevándose a cabo importantes inversiones para la investigación y el desarrollo de nuevas medidas y tecnologías que permitan aumentar la seguridad vial y diseñar de la manera más óptima posible las infraestructuras así como el despliegue de elementos de control que en ellas se hallan.

Históricamente ha resultado difícil evaluar la seguridad de los nuevos diseños y despliegues de infraestructuras para tráfico rodado basándose en la carencia de modelos predictivos eficaces, capaces de aportar resultados fehacientes que se ajustaran a la realidad. Es decir la simulación de tráfico vehicular ha supuesto un hito importantísimo en el área ingenieril ya que con la

herramienta de simulación y la adecuada modelización de los escenarios se pueden obtener grandes resultados de una manera eficiente.

Existen numerosas razones por las que estos modelos de tráfico han ido desarrollándose cada vez de manera más profesional hasta alcanzar una gran similitud con la realidad vehicular.

- Simular el efecto de control de medidas en determinada situación como por ejemplo:
  - Límites de velocidad.
  - Simular la prohibición de adelantamiento para los camiones, sobre todo en cuesta arriba o cuesta abajo.
  - Restricciones o de cambio de carril.
  - Tráfico o en el control de flujo en las rampas.

- Para simular el efecto de la nueva infraestructura antes de que ésta se haya construido.
- Para simular la influencia de los vehículos con sistemas de control de crucero.
- Por último se puede incluso simular nuevas normas de tráfico antes de que se establezcan.

En resumen, la simulación de tráfico vehicular contribuye al estudio del tráfico en su amplia variedad de campos sin tener que realizar experimentos de campo, permitiendo jugar con las variables que pueden ser modificadas para llegar a soluciones que no impliquen inversiones a ciegas y sin necesidad de construir nuevas y costosas infraestructuras.

I.IV.III) Simuladores de tiempo vs. eventos

Las redes inalámbricas están creciendo rápidamente tanto en tamaño y complejidad, por lo que resulta cada vez más difícil investigar este tipo de redes que escalan a grandes tamaños y que son complejas analíticamente. Por lo tanto, las herramientas de simulación por ordenador son cada vez herramientas más imprescindibles en el estudio de dichas redes (10).

Encontramos dos grandes bloques de clasificación para los simuladores por un lado los simuladores de tiempo y por otro los simuladores por eventos.

En los **simuladores basados en tiempo** tenemos una variable que registra el tiempo actual, la cual se va incrementando en pasos fijos. Después de cada incremento comprobamos qué acontecimientos pueden ocurrir en el punto de la simulación actual para manejarlos adecuadamente. Por ejemplo, supongamos que queremos simular la trayectoria de un proyectil. Para t=0, asignamos al misil una posición inicial y la velocidad. En cada paso de tiempo se calcula una nueva posición y velocidad utilizando las fuerzas que actúan sobre el mismo. Este tipo de simuladores son muy adecuados en este caso porque existe un evento, movimiento, que ocurre para cada paso de tiempo, Para proceder a detener la simulación o bien se le da un punto en el tiempo para finalizar dicha simulación o bien hasta que el sistema alcance un determinado estado. La principal ventaja de estos simuladores subyace que en son especialmente eficaces a la hora de simular escenarios donde existe una gran frecuencia en la aparición de eventos y esto implicaría una sobrecarga de cálculo para aquellos simuladores

basados en eventos. En el caso de las redes son especialmente apropiados para simular detalles de la capa física por la gran cantidad de detalles que en ella se dan.

Los **simuladores basados en eventos** son adecuados para simular situaciones donde los acontecimientos no se producen en intervalos regulares. Pongamos por ejemplo la cola de un banco donde los clientes no llegan en intervalos regulares, o bien que en determinados puntos del día pueden formar una gran cola. Este enfoque utiliza una lista de eventos que ocurren en distintos momentos, y el simulador los maneja en función del orden ascendente de tiempo. La manipulación de un evento puede modificar la lista de acontecimientos posteriores y a su vez el simulación hace saltar "el tiempo'' a la hora del evento siguiente.

II-PLATAFORMAS PARA LA SIMULACIÓN

En el segundo capítulo de este proyecto se presentan conceptos como la supercomputación, superordenadores, aceleración de cómputo, técnicas de programación paralela, etc. La idea es la de introducir una potente plataforma de trabajo la cual puede servir de base para lanzar simulaciones de redes vehiculares que requieran una gran cantidad de recursos para su resolución y cómputo. Después de la necesaria introducción teórica pasaremos a ver un primer caso práctico en el cual se utilizan las técnicas de paralelización en el ámbito vehicular.

II. I) El Supercomputador Ben-Arabí

Situado en el Parque científico de la Región de Murcia (11) el supercomputador *Ben Arabí* se presenta como el cuarto superordenador más potente en España. El sistema supercomputador Ben Arabí está basado en una arquitectura de multiproceso simétrico pura, sin nodos fuertemente acoplados, con una potencia máxima de cálculo total de 10,611 TFlops. El sistema Ben Arabí está formado por dos arquitecturas claramente diferenciadas:

Por un lado **(i)** un nodo central *HP Integrity Superdome SX2000* (12) con 128 núcleos del procesador *Intel Itanium-2 dual-core Montvale* (1,6 Ghz, 18 MB de caché L3) y 1,5 TB de memoria compartida, llamado ***Ben***. En la arquitectura de memoria compartida todos los procesadores del sistema pueden acceder directamente a todas las posiciones de memoria. El espacio de memoria física es único y global. La utilización de este espacio de memoria común evita la duplicación de datos y el lento trasvase de información entre los procesos. El sistema está organizado en células o nodos, las cuales están formadas por un procesador y una memoria. Los procesadores pueden acceder a la memoria de las demás células a través de la red de interconexión. El acceso a la memoria local (en la misma célula) es más rápido que el acceso remoto (en una célula diferente).

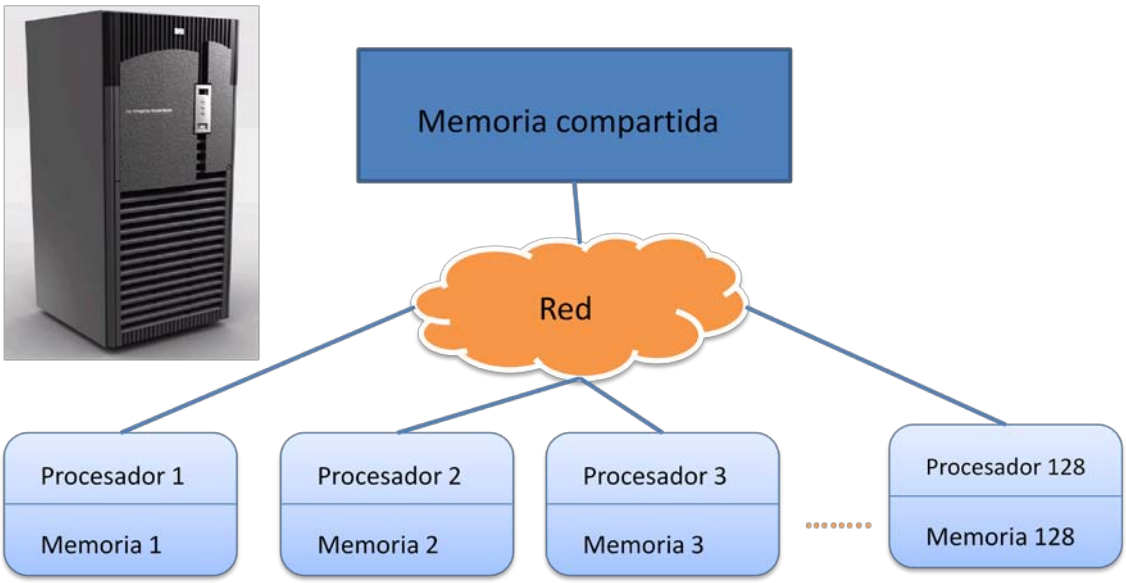


Figura 10 Esquema del superdome HP Integrity SX2000

Por otra parte, **(ii)** un clúster formado por 102 nodos de cálculo, que ofrecen un total de 816 núcleos del procesador *Intel Xeon Quad-Core E5450* (3 GHz y 6 MB de caché L2,) y 1072 GB de memoria distribuida, llamado ***Arabí***.

A continuación podemos ver resumidas las características del clúster computacional de nodos finos en formato *Blade* de alta eficiencia energética:

Capacidad: 9; 72 Tflops
Procesador: *Intel Xeon Quad-Core E5450*
Número de nodos: 102
No de núcleos: 816 (8 núcleos por cada nodo)
Memoria/Nodo: 32 nodos de 16 GB y 70 de 8 GB
Memoria/Core: 3 MB (6 MB compartidos entre 2 núcleos)
Frecuencia de reloj: 3 Ghz

En este segundo sistema, el espacio de memoria es distribuido, cada procesador tiene su espacio de direcciones exclusivo. Por lo que es preciso el reparto de datos de entrada a cada procesador. El procesador que envía los datos tiene que gestionar la comunicación aunque no tenga implicación directa en el cálculo del procesador destino. La necesidad de sincronización dificulta la programación porque la paralelización se realiza de forma manual. Los procesos se van a comunicar entre sí enviando y recibiendo mensajes, cada proceso utiliza sólo memoria local. Si el tamaño de los datos a procesar es muy grande hay que hacer un particionado que aloje una parte de ellos en la memoria local de cada unidad de procesamiento. La transferencia de información requiere que se creen operaciones cooperativas para ser utilizadas en cada proceso: por ejemplo, una operación de envío de mensaje debe tener una operación de recepción del mensaje en algún proceso.

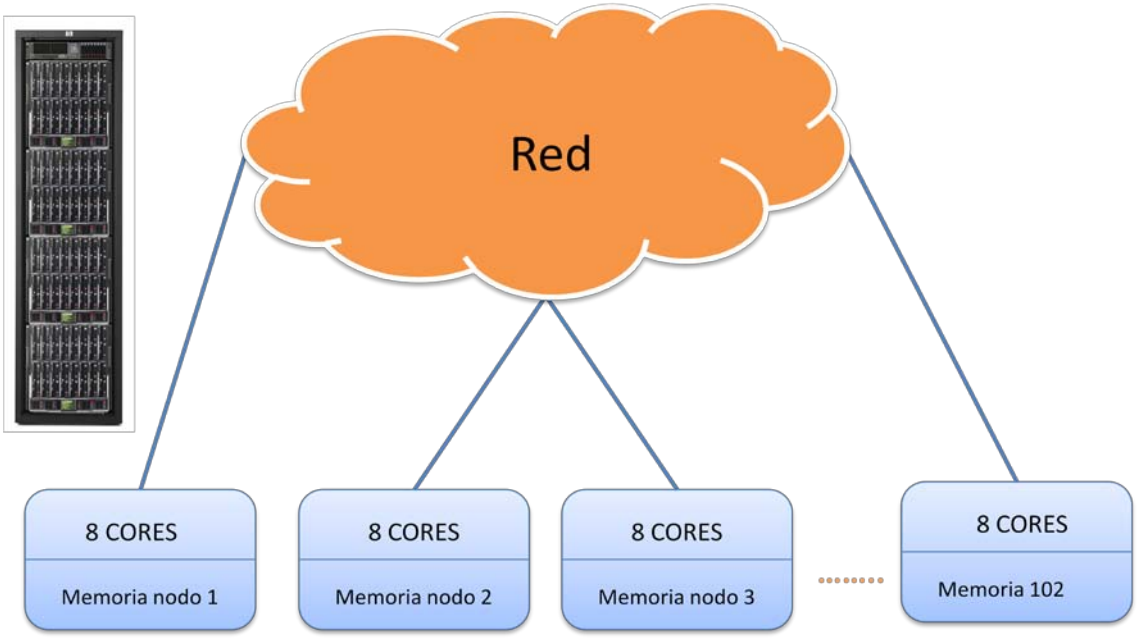


Figura 11 Esquema para el Cluster de 102 Nodos con 8 cores por cada nodo de cálculo

II. II) Pinceladas para la computación en paralelo

El pilar del proceso de paralelización es discernir entre un programa secuencial donde existen declaraciones de datos seguidas de instrucciones que serán ejecutadas en orden, es decir una detrás de otra, frente a la programación concurrente o paralela donde existe un conjunto de programas secuenciales ordinarios que son ejecutados como procesos paralelos. Es importante manejar cómodamente los conceptos asociados a la computación paralela, por ejemplo:

- Tarea: secuencia de comandos, programas o documentos para ejecutarlos.
- Proceso: operación o conjunto combinado de operaciones con datos, o bien una secuencia de acontecimientos definida única y delimitada, que obedece a una intención operacional en condiciones predeterminadas. También se denomina proceso a una función que se está ejecutando.
- *Thread* (hilo de ejecución): es una característica que permite a una aplicación realizar varias tareas a la vez (concurrentemente). Es básicamente una tarea que puede ser ejecutada en paralelo con otra tarea. Los hilos de ejecución que comparten los mismos recursos, sumados a estos recursos, son en conjunto conocido como un proceso.
- Modelo de programación *fork-join*: una tarea muy pesada se divide en *K* hilos (*fork*) con menor peso, para luego recolectar sus resultados al final y unirlos en un solo resultado (*join*). Un *thread* maestro se ejecuta secuencialmente hasta que se encuentra con una directiva y se produce una bifurcación con nuevos *threads*. Estos *threads*, a su vez, pueden ser distribuidos y ejecutados en diferentes procesadores, reduciendo el tiempo de ejecución. Los resultados de cada ejecución de *threads* pueden combinarse posteriormente.

El motivo de paralelizar es tan simple como obtener el resultado en el menor tiempo posible. Las simulaciones vehiculares más realistas que pueden incorporan modelos complejos a nivel matemático y de comunicaciones requieren una capacidad de cómputo tal que en la mayoría de los casos no se contempla en aquellos ordenadores que no cuenten con el título de supercomputador. Lo que obliga a que en muchos casos (con computadores convencionales) se tengan que restringir parámetros de las simulaciones como son el número total de nodos, complejidad y extensión del escenario simular o detalles de comunicación (p.e. modelos simplistas del canal de propagación); dando lugar a resultados de simulación cuestionables y poco acordes con la realidad.

Aunando recursos de hardware y las técnicas de programación paralela se pueden llegar a conseguir mejoras asombrosas en problemas que en principio se podían considerar inabordables.

El procedimiento de paralelismo se puede ilustrar como sigue en la siguiente figura:

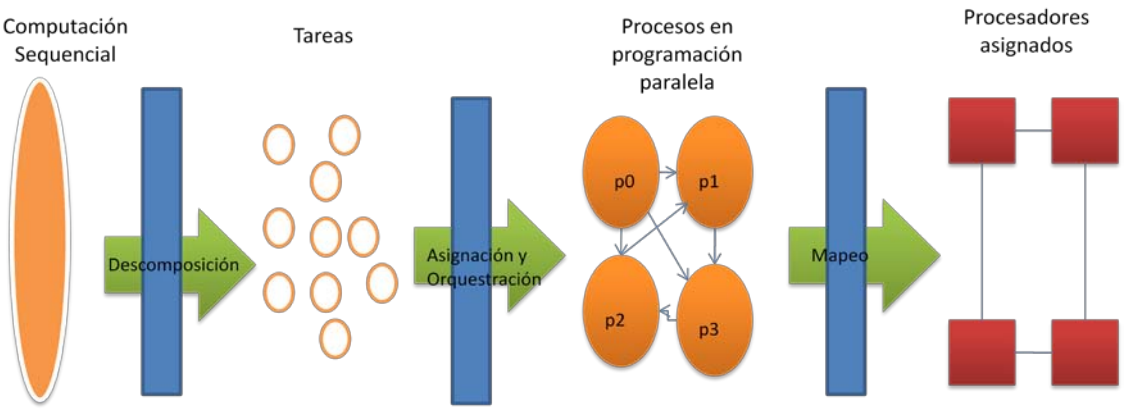


Figura 12 Procedimiento genérico para conseguir paralelismo

II.I I.I) Mecanismos de programación paralelizada: MPI y OpenMP

Una vez introducidos y entendidos los conceptos básicos de paralelización pasaremos a ver las técnicas y herramientas existentes para llevar a cabo nuestro objetivo; conseguir una eficiencia de cómputo factible.

MPI: *Message Passage Interface*

MPI (13) es una librería *opensource* específica para el paso de mensajes, propuesta como estándar y consensuada en un comité de proveedores, desarrolladores y usuarios de todo el mundo. Esta API (*Application Program Interface*) es compatible con los lenguajes de programación C, C++, Fortran y Ada. La ventaja de MPI sobre otras bibliotecas de paso de mensajes, es que los programas que utilizan la biblioteca son portables, rápidos y su diseño está enfocado en obtener un mayor rendimiento en máquinas de procesamiento paralelo y *clusters*, es decir en sistemas de memoria distribuida. El paso de mensajes es una técnica empleada en programación concurrente para aportar y permitir la exclusión mutua, de manera similar a como se hace con herramientas clásicas como son los semáforos, monitores, etc. Los elementos principales que intervienen en el paso de mensajes son el proceso que envía, el que recibe y el mensaje. Dependiendo de si el proceso que envía el mensaje espera a que el mensaje sea recibido, se puede hablar de paso de mensajes síncrono o asíncrono. En el paso de mensajes asíncrono, el proceso que envía, no espera a que el mensaje sea recibido, y continúa su ejecución, siendo posible que vuelva a generar un nuevo mensaje que puede ser enviado antes de que se haya recibido el anterior. Por este motivo se suele emplear buzones, en los que se almacenan los mensajes a la espera de que un proceso los reciba. En el paso de mensajes síncrono, el proceso que envía el mensaje espera a que un proceso lo reciba para continuar su ejecución.

OpenMP para plataformas paralelas

OpenMP (14) es un estándar abierto para proveer de mecanismos de paralelización a multiprocesadores de memoria compartida. La API de OpenMP soporta programación de memoria compartida de tipo multi-plataforma en los lenguajes de programación Fortran, C y C++, y en todas las arquitecturas, incluyendo plataformas bajo sistema operativo Unix y Windows. OpenMP es un modelo escalable y portable desarrollado por distribuidores de hardware y software que proporciona a los programadores de memoria compartida una interfaz simple y flexible para desarrollar aplicaciones de tipo paralelo que pueden correr tanto en un PC como en un supercomputador.
OpenMP utiliza el modelo de programación paralela *fork-join* (15) con generación de múltiples *threads*, donde una tarea muy pesada se divide en *k* hilos (*fork*) con menor peso, para luego recolectar sus resultados al final y unirlos en un solo resultado (*join*). Un *thread* maestro corre secuencialmente hasta que se encuentra con una directiva de OpenMP y se produce una bifurcación con *threads* esclavos, siendo el *thread* maestro el que los pone en marcha. Estos *threads* pueden ser distribuidos y ejecutados en diferentes procesadores, reduciendo el tiempo de ejecución.

En la figura a continuación se muestra un esquema en el que podemos discernir las principales diferencias entre un sistema de memoria compartida (donde la API a utilizar es OpenMP) y un sistema de memoria distribuida (donde la API a utilizar es MPI)

En el caso de la izquierda al ser una memoria de tipo compartida todos los procesos podrán acceder a los recursos comunes y en este caso la adecuación de OpenMP como herramienta orquestal de este proceso para la concurrencia es perfecta. La API de OpenMP organiza el acceso a los datos y el sincronismo en las operaciones por parte de los procesos sobre los datos contenidos en la memoria compartida. Para el caso de la derecha la memoria es distribuida y por tanto, la solución para ejecutar de una manera eficaz y correcta una concurrencia de cálculo, será la de hacer uso del paquete MPI diseñado específicamente para este tipo de sistemas. Éste permite que los procesos que comuniquen entre sí, se organicen y se distribuyan la información mediante unos mensajes específicos.

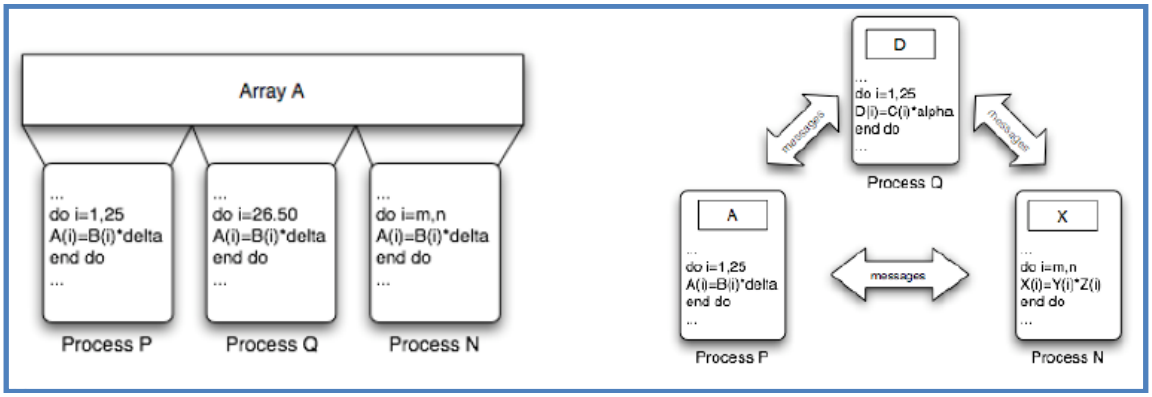


Figura 13 Sistema de memoria compartida frente a paso de mensajes en memoria distribuida MPI (8)

Ahondemos pues en la sintaxis para OpenMP puesto que este paquete será utilizado en un proyecto de cálculo concurrente propuesto en secciones posteriores de este capítulo.

Sintaxis para OpenMP

A continuación se exponen las principales directivas (16) que definen la sintaxis del lenguaje de programación paralela OpenMP. Como puede apreciarse se trata de una sintaxis bastante sencilla, pero cabe destacar que lo complejo de este tipo de programación es la de tener siempre en cuenta qué bucles acceden a que variables y qué datos dependen de otros valores anteriores y por tanto, no son susceptibles de ser paralelizados.

Cláusulas de alcance de datos

- private. erán variables privadas a los *threads*, no se inicializan antes de entrar y no se guarda su valor al salir.
- firstprivate. Privadas a los *threads*, se inicializan al entrar con el valor que tuviera la variable correspondiente.
- lastprivate. Privadas a los *threads*, al salir se quedan con el valor de la última iteración.
- shared. Compartidas por todos los *threads*.
- default. Indica cómo serán las variables por defecto.

Constructor *parallel*: controla el paralelismo y crea los múltiples *threads* que corren concurrentemente, donde cada uno de estos hilos de ejecución accederá a esta zona para ejecutar su tarea correspondiente.

```
#pragma omp parallel [cláusulas]
        Bloque
```

- Se crea un grupo de *threads*.
- El que los pone en marcha actúa de maestro.
- Con cláusula *if* se evalúa su expresión.
- El número de *threads* a crear se obtiene por variables de entorno o llamadas a librerar.
- Hay barrera implícita al final de la región.
- En caso de anidamiento cada esclavo creará otro grupo de *threads* esclavos de los que será el maestro.

Constructor *FOR*

```
#pragma omp for [clausulas]
        bucle for
```

- Las iteraciones se ejecutan en paralelo por *threads* que ya existen.
- La parte de inicialización del *for* debe ser una asignación.
- La parte de incremento debe ser una suma o resta.

▪ La parte de evaluación es la comparación de una variable entera sin signo con un valor, utilizando un comparador mayor o menor (puede incluir igual).
▪ Los valores que aparecen en las tres partes del *for* deben ser enteros.
▪ Hay barrera al final a no ser que se utilice la cláusula *nowait*.
▪ Hay una serie de cláusulas (*private*, *firstprivate*, *lastprivate* y *reduction*) para indicar la forma en que se accede a las variables.

Constructor *SECTIONS*

```
#pragma omp sections [clausulas]
     [# pragma omp section]
          bloque
     [# pragma omp section]
          Bloque
```

▪ Cada sección se ejecuta por un *thread*.
▪ Hay barrera al final de *sections*, a no ser que se utilice la cláusula *nowait*.
▪ Se puede indicar mediante cláusulas (*private*, *firstprivate*, *lastprivate* y *reduction*) para indicar la forma en la que se accede a las variables.

Constructores de ejecución SECUENCIAL

```
#pragma omp single [cláusulas]
          Bloque
```

▪ El bloque se ejecuta por un único *thread*.
▪ Hay barrera al final, a no ser que se utilice la cláusula *nowait*.

```
#pragma omp master [cláusulas]
          Bloque
```

▪ El bloque lo ejecuta el *thread* maestro.
▪ No hay sincronización al entrar ni al salir.

```
#pragma omp ordered [cláusulas]
          Bloque
```

▪ El programa se ejecutará (dentro de un for) de una manera secuencial.

Constructores de SINCRONIZACIÓN

```
#pragma omp critical [nombre]
          Bloque
```

- Se asegura exclusión mutua en la ejecución.
- Con el nombre identificamos secciones críticas distintas.

```
#pragma omp barrier
```

- Sincroniza todos los *threads* del equipo.

II. II.II) Primeros resultados de computación paralela en un modelo matemático

En la última sección de este capítulo queremos ilustrar mediante un primer trabajo de paralelización aplicado sobre un modelo matemático vehicular, como la técnica de programación basada en la API de OpenMP proporciona resultados que mejoran visiblemente el tiempo de cómputo, ambientados esto si en una plataforma de supercomputación con sistema de memoria compartida.

Indicar que este trabajo se basa en un modelo matemático desarrollado por Juan Bautista Tomás Gabarrón compañero de equipo de investigación y que fue llevado a cabo a su vez por Carolina García Costa y Rocío Murcia Hernández. Nuestras simulaciones se han ejecutado dentro del entorno del clúster Arabí, utilizando 2, 4 y 8 núcleos o *cores* de cada uno de los nodos para comparar los diversos resultados en lo relativo al tiempo de simulación. Para este caso llama la atención que ejecutemos en una plataforma de memoria distribuida un programa paralelo bajo el estándar OpenMP diseñado para sistemas de memoria compartida, es importante tener en cuenta que para cada nodo existe una memoria propia del mismo bien de 16 GB y o de 8 GB, y será esta la que se compartirá con los distintos *cores* a utilizar, luego estamos bajo el marco de un sistema de memoria compartida.

No entraremos en los detalles matemáticos del modelo en sí ni de programación, si bien comentar que se trata de una modelización matemática basado en un mecanismo de CCA (*Chain/Cooperative Collision Avoidance*) para evitar colisiones que se presenta como un método novedoso para reducir el número de accidentes en la carretera proporcionando a los vehículos capacidades de comunicación cooperativa, de tal manera que sean capaces de reaccionar ante posibles riesgos de accidente. El mecanismo CCA generará una notificación que será entregada como un mensaje, a través de un esquema de tipo *one-hop* (un salto), a todos aquellos vehículos que se encuentren en peligro potencial de formar parte de un accidente (17).

Se tendrá en cuenta a su vez que en el momento de implantar dicho mecanismo para mejorar la seguridad, el despliegue de la tecnología CCA se supone que no será del 100% de manera instantánea en todos los vehículos, sino que se realizará una implantación gradual de dicha tecnología en los vehículos dotándolos del hardware y software necesarios para llevar a cabo este tipo de comunicaciones vehiculares. Dentro del modelo matemático se distinguen tres bucles claramente susceptibles a ser paralelizados pues son los que aumentan de manera dramática el tiempo de cómputo:

1. *BUCLE A: Multiplicamos un vector por una matriz*

Se trata de paralelizar la multiplicación de un vector por una matriz, de modo que cada *thread* se encargará de multiplicar el vector por una columna de la matriz. Veamos como ejemplo el pseudocódigo asociado a esta operación a continuación:

```
void mv(double *m,int fm,int cm,int ldm,double *v,int fv,double *w,int
fw) {
int i,j,iam,nprocs;
double s;
#pragma omp parallel private(iam,nprocs) {
nprocs=omp_get_num_threads();
iam=omp_get_thread_num();
#pragma omp master
THREADS=nprocs;
#pragma omp for private(i,s,j) schedule(dynamic)
for (i = 0; i < fm; i++) {
#ifdef DEBUG
printf("thread %d fila %d \n",iam,i);
#endif
s=0.;
for(j=0;j<cm;j++)
s+=m[i*ldm+j]*v[j];
 w[i]=s;

}
}
}
```

2. *BUCLE B: Modificamos la distancia media intervehicular en el rango acordado*

En este caso se paralelizará el bucle que varía la distancia media intervehicular de los vehículos dentro de un rango, un total de 65 iteraciones que se repartirán entre los diferentes *threads*.

3. *BUCLE C: Variamos el porcentaje de despliegue de la tecnología CCA*

Se quiere realizar un estudio del impacto en la seguridad vial y en el número medio de accidentes en función del grado de implantación del mecanismo de *Chain/Cooperative Collision Avoidance* para mejorar la seguridad en las carreteras. Debido al gran número de combinaciones que obtendremos en función del porcentaje de despliegue de la tecnología CCA en las entidades móviles se quiere paralelizar este bucle de modo que cada thread se encargue de evaluar un conjunto de combinaciones. Del porcentaje seleccionado dependerá el número de combinaciones totales a simular, este viene dado por una distribución de tipo binomial, esto es:

$$\binom{n}{m} = \frac{n!}{(n-m)!\ m!}$$

Ecuación 1 Distribución binomial

Siendo *n* el número total de coches y *m* el número de vehículos dado por el despliegue de CCA (es decir, dotados con la tecnología CCA). Como ejemplo indicar que si el índice de penetración o despliegue de CCA= 50% tenemos 184.756 combinaciones posibles, convirtiéndose claramente en el punto crítico de cálculo. Para el ejercicio realizamos distintas combinaciones de paralelismo en función de los bucles A, B y C, obteniendo los resultados siguientes, que se muestran en la tabla a continuación:

Programa	Combinación
Programa 1	Modelo sin paralelizar
Programa 2	Paralelismo en el bucle A
Programa 3	Paralelismo en el bucle B
Programa 4	Paralelismo en el bucle C
Programa 5	Paralelismo en bucles A+B
Programa 6	Paralelismo en bucles A+C
Programa 7	Paralelismo en bucles B+C
Programa 8	Paralelismo en bucles B+C

Tabla 3 Combinaciones para ejecutar paralelismo

Los parámetros que configurarán los diversos escenarios de simulación son los siguientes:

- N= número de coches
- % de CCA
- Distancia media intervehicular
- Número de *cores* a utilizar

En nuestro experimento mantenemos fijo el número de coches y tomamos porcentajes de despliegue de CCA en saltos del 25%, al igual que un rango específico de distancias intervehiculares mientras que como indicamos el número de *cores* a utilizar por nodo serán de 2, 4 y 8.

Como conclusiones finales podemos afirmar que hemos conseguido alcanzar una reducción absoluta del tiempo de cálculo del 83 %, lo que supone una mejora muy destacable. Por otra parte, comparando los mejores tiempos de cómputo entre los dos extremos tecnológicos bajo estudio, es decir, la utilización de 2 *cores* frente a 8 *cores*, pertenecientes a la arquitectura compartida de los nodos del clúster Arabí, se obtiene una mejora del 67, 7 %.

Las disminuciones más significativas en el tiempo de cálculo las obtuvimos para aquellos programas en los que se incluyó el paralelismo en el Bucle C, el cual implica una paralelización de todas las combinaciones que se dan al variar el porcentaje de despliegue de la tecnología de *Chain Collision Avoidance*.

Para comparar los resultados hemos seleccionado en todos los casos los resultados derivados de las configuraciones del 50% en el despliegue de CCA, siendo esta opción la más pesada

computacionalmente hablando, ya que da lugar al mayor número de posibles combinaciones dentro de nuestro modelo y por tanto constituye el mayor pico de cómputo al realizar las simulaciones.

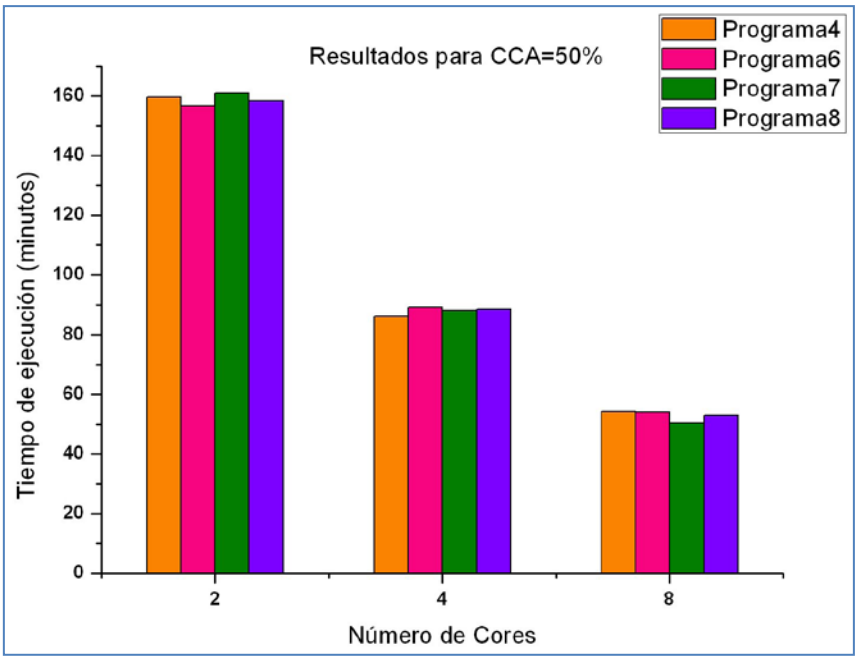


Figura 14 Resultados para configuraciones de CCA con despliegue del 50% para 20 vehículos

Hemos encontrado una relación directa de mejora del tiempo de ejecución al duplicar el número de *cores* utilizado, obteniendo un decremento del tiempo de cálculo aproximadamente del 40 %. Por lo tanto, tras ver la mejora tan significativa que se obtiene al aumentar el número de *cores*, nuestra intención es realizar las simulaciones en el nodo central del supercomputador *Ben-Arabí* y utilizar hasta 128 *cores*, para así poder aumentar el número de vehículos, lo que aumenta considerablemente el número de combinaciones.

III- SIMULADORES *OPENSOURCE*

En el capítulo III que nos ocupa repasaremos los principios de la simulación vehicular estableciendo a continuación una clasificación para las herramientas de simulación VANET. Llegando por último a una clasificación de los simuladores *opensource* y *freeware* disponibles para la comunidad investigadora.

III.I) Los pilares de la simulación vehicular

La comunicación vehicular se considera como una tecnología clave para mejorar la seguridad vial y el confort de conductores y pasajeros a través de lo que se denomina como Sistemas de Transporte inteligente (ITS). Las VANET (*Vehicular Ad-hoc Networks*) se definen como redes MANET (red móvil ad-hoc) con alta movilidad en sus nodos, construidas a partir de las rutas vehiculares y por lo tanto caracterizadas por velocidades muy altas y con grados de libertad limitados a los patrones de movimiento para los nodos. Tales características particulares, a menudo hacen que los protocolos estándar de red sean ineficientes o no utilizables en ambientes VANET. (18)

Las actividades de investigación en cuestiones VANET se están convirtiendo cada vez en un factor importantísimo para los avances en los múltiples campos de aplicación en el desarrollo de las comunicaciones V2V y V2I. Los resultados obtenidos de las simulaciones se utilizan para poder convertir en una realidad tales ideas como disminuir la congestión de tráfico, mejorar los servicios informativos para los conductores y sobre todo proporcionar aplicaciones que permitan mejorar la seguridad en las carreteras. (19)

Es aquí donde entran en juego los distintos protocolos a simular tales como: *wireless, multihop routing* y *broadcas*t, los cuales se modelan también en distintos simuladores de redes. Si bien será necesario contar con otras herramientas qué proporcionan la movilidad de los vehículos en movimiento ajustándose en lo más posible a la realidad. La simulación de escenarios VANET está estrechamente ligada al modelado de las transmisiones del canal radio entre nodos y por tanto, se necesitan las posiciones exactas de los nodos simulados. Es entonces cuando nos decantamos por los modelos microscópicos que modelarán el comportamiento de los vehículos de manera individual y las interacciones entre ellos, siendo éstos los más adecuados para las redes VANET.

Lo ideal sería probar y evaluar las distintas evaluaciones de los protocolos en experimentos de campo pero hay que tener en cuenta las dificultades logísticas, el factor económico y las limitaciones de la tecnología. Todo esto hace que recurrir a la simulación sea la solución más óptima para la validación de protocolos en las primeras fases de desarrollo, constituyendo a su vez un primer paso para el desarrollo de las tecnologías VANET en el mundo real.

Otra hándicap a tener en cuenta son las capacidades de computación necesarias a la hora de simular determinados escenarios VANET, complejos y de amplia extensión, puesto que a mayor nivel de detalle mayores capacidades computacionales se requieren, sobre todo en lo que refiere a la simulación de red.

Históricamente la movilidad ha sido siempre un punto conflictivo, del mismo modo ha ido aumentando la necesidad de obtener un modelo más realista de los movimientos de los vehículos, que capture el comportamiento de los mismos. Por otra parte, los modelos de movilidad están obligados a ser dinámicamente reconfigurables con el fin de reflejar los efectos de un protocolo de comunicación en particular.

La comunidad científica por lo tanto comenzó a trabajar en el desarrollo y/o la reforma de modelos de movilidad específicos a las propuestas de vehículos. Después de algunos años de emocionantes desarrollos y evoluciones una gran variedad de modelos están disponibles, los

cuales varían desde el más trivial hasta los más realistas. Lamentablemente, este desarrollo ha sufrido de falta de coordinación, ya que la tónica general ha sido que cada grupo de investigación desarrolle un modelo bajo el cumplimiento de sus propias necesidades específicas. Es por esto que para alguien que comienza en este ámbito de investigación es difícil entender las características de cada modelo, comparar sus ventajas y desventajas, y elegir la mejor solución para sus necesidades.
Si bien gracias a los esfuerzos de los investigadores, se han ido mejorando los modelos de movilidad de tráfico rodado así como las herramientas para su estudio desencadenado un estudio más profundo de escenarios más complejos y cada vez más grandes, así como un análisis más preciso de protocolos VANET y sus aplicaciones. Aunque todavía queda un gran camino por recorrer.

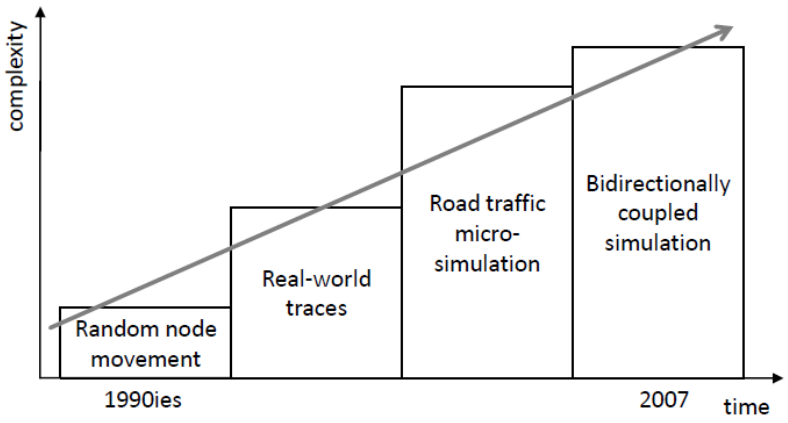


Figura 15 Evolución histórica de las estrategias para los modelos de movilidad en la investigación VANET (19)

Resumiendo acerca de la simulación en redes vehiculares:
La simulación es una manera de ejecutar experimentos en un entorno controlado. Las comunicaciones para redes VANET se han convertido de una manera cada vez más incipiente en un tema de gran interés en la comunidad investigadora así como para las grandes compañías de automoción. El objetivo de la investigación VANET es la de desarrollar un sistemas de comunicaciones que permita de una forma rápida y eficiente la distribución de los datos de tal modo que los pasajeros se beneficien en cuestiones de seguridad y confort. Es crucial por tanto, testear y evaluar las implementaciones de los protocolos en el mundo real, es por esto que las simulaciones son usadas siempre en primer lugar a la hora de desarrollar un protocolo de comunicaciones.

III.II) Taxonomía de los simuladores vehiculares

En un principio, los mundos de los modelos de movilidad y simuladores de redes no fueron creados para comunicarse y si diseñados para ser controlados por separado, sin casi

ninguna interacción entre ellos. Al imaginar las aplicaciones prometedoras que podrían obtenerse a partir de las VANET, escenarios donde la comunicación vehicular puede alterar la movilidad, y donde la movilidad mejoraría las capacidades de la red se presenta una situación no esperada y que supone un obstáculo importante para el desarrollo de las redes vehiculares. Con estos precedentes, nace la necesidad inminente de crear una interacción entre un modelo de movilidad y un simulador de red (de comunicaciones). A continuación veremos los diferentes enfoques para conseguir nuestro objetivo: simular redes vehiculares.

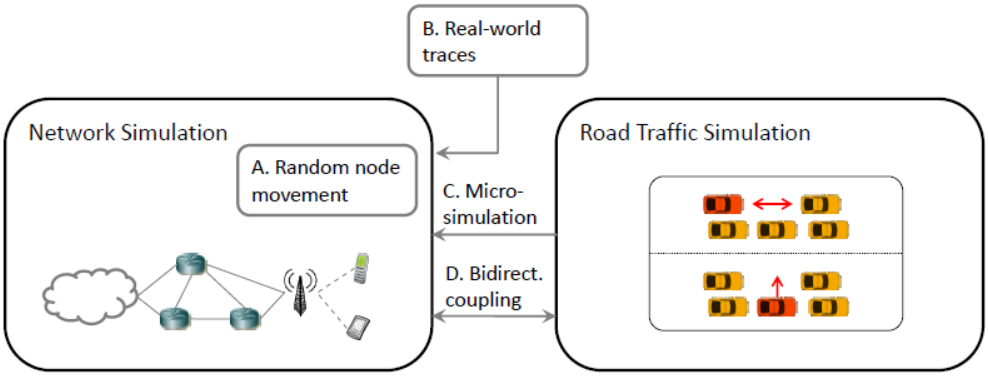


Figura 16 Técnicas para la simulación de redes vehiculares (19)

Nuestro fin es el de trabajo con simuladores de tipo *OpenSource* para estudiar sus propiedades, configuraciones e implementaciones. Una vez elegida la/s herramienta/s pasaremos a la simulación de un llamado escenario vehicular.
La clasificación que seguiremos para las **herramientas VANET**, en este apartado es la siguiente:

A) Simuladores Aislados.

B) Simuladores integrados.

C) Simuladores híbridos.

III.II.I) Simuladores Aislados VANET

Inicialmente, la movilidad se veía por parte de los simuladores de red como perturbaciones al azar de las configuraciones óptimas estáticas. Entonces se tuvo que pensar en la forma de proporcionar control a estos simuladores de redes sobre los patrones de movilidad y se les dotó con la posibilidad de cargar escenarios de movilidad. Los diferentes modelos deben ser generados antes de la simulación y deben ser traducidos por el simulador de acuerdo con el formato de traza predefinido. En estos casos las modificaciones del escenario de la movilidad no son posibles, y no existe, por tanto la interacción entre estos dos mundos. Por desgracia, todos los modelos históricos y una gran cantidad de los modelos recientes de movilidad a disposición de la comunidad de investigación entran en esta categoría. A pesar de la limitación descrita anteriormente, la comunidad se ha adaptado muy

bien a estos modelos aislados, ya que esta estrategia permite la evolución independiente de los modelos de movilidad y de red. Además debemos añadir que la necesidad de esta interacción bidireccional ha aparecido recientemente teniendo interés significativo para las aplicaciones específicas vehiculares, tales como la seguridad o la gestión del tráfico.

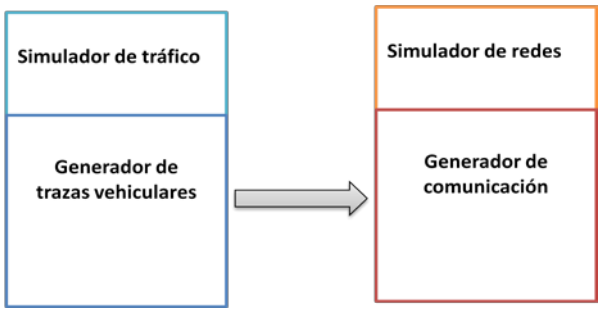


Figura 17 Esquema para Simulador Aislado

III.II.II) Simuladores integrados VANET

Si los simuladores de red no pueden interactuar plenamente con los simuladores de movilidad una solución es sustituir ambos por un simulador único de eventos del tipo *off-the shelf*. En consecuencia, se crearon nuevos simuladores de red simplistas, donde se compensa la falta de protocolos elaborados con una colaboración directa entre la red y el generador de movilidad. El principal objetivo de estos simuladores es tener ambos modelos trabajando e interactuando de manera eficiente. Sin embargo, la principal limitación es en realidad la mala calidad del simulador de redes. De hecho, este enfoque ha sido hasta ahora utilizado para estudiar los efectos básicos de la red, no pudiendo pasar la prueba de los últimos protocolos de enrutamiento ad hoc móviles que requieren capas MAC y físicas estandarizadas y realistas. Como nota adicional indicar que la principal tendencia hoy en día en desarrollo por parte de los simuladores de redes es un casamiento entre los protocolos estándares y la eficiencia computacional a través de la computación paralela y distribuida.

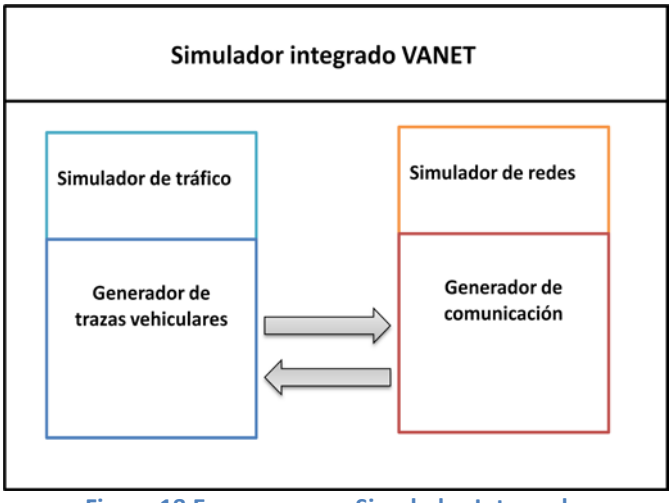


Figura 18 Esquema para Simulador Integrado

III.II.III) Simuladores híbridos VANET

La última aproximación presentada para los simuladores VANET, es crear una unión hibrida entre ambos simuladores de redes y de movilidad a través de una interfaz diseñada a tal efecto. En consecuencia, ambos simuladores trabajan en paralelo y por lo tanto pueden interactuar dinámicamente entre sí mediante la alteración de los patrones de movilidad basados en los flujos de red y viceversa. Con este enfoque híbrido y mediante el mecanismo de interconexión de simuladores independientes y validados nos encontramos en condiciones de beneficiarnos de las mejores propiedades de simulación de ambas herramientas. Así los modelos de movilidad de última generación se pueden adaptar a trabajar con simuladores de red modernos y eficientes. Sin embargo, computacionalmente exige numerosos recursos ya que ambos simuladores necesitan correr paralelamente y el desarrollo de la interfaz de interconexión no es una tarea sencilla en función de la red específica y simuladores de tráfico. Ambas comunidades se encuentran muy predispuestas y con un gran interés para trabajar conjuntamente.

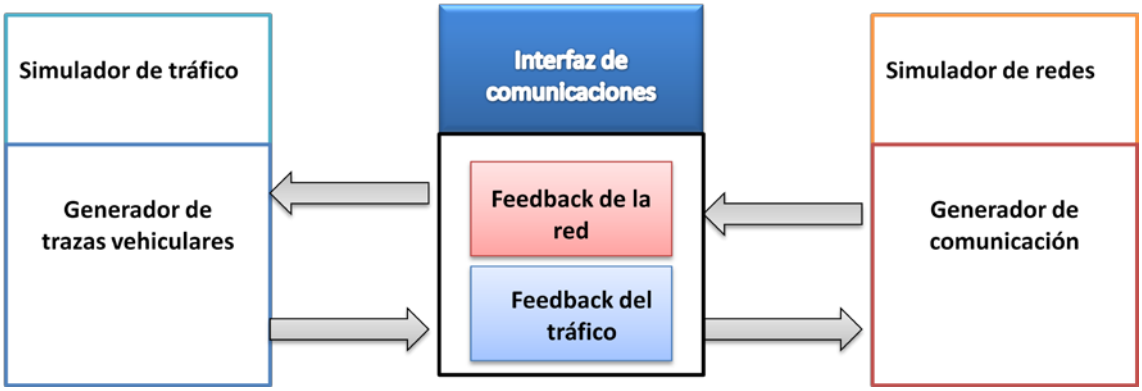


Figura 19 Esquema para Simulador Híbrido

III.III) Clasificación actual de herramientas *OpenSource*

A continuación pasaremos a ver una clasificación actual de las distintas herramientas *OpenSource* y *Freeware* de las que disponemos en la actualidad, haciendo un barrido desde las herramientas de movilidad, simuladores de redes hasta llegar a las herramientas VANET de interés, las de tipo integrado y las híbridas, que serán con las que más adelante trabajaremos.

III.III.I) Simuladores de Tráfico

El desarrollo de simuladores de tráfico comenzó hace décadas con el modelado de las secciones críticas de autopistas o intersecciones urbanas para la ingeniería de tráfico (rodado). La mayor parte de las herramientas de simulación realistas de tráfico vial, se han desarrollado para analizar la movilidad vehicular en ambos niveles microscópicos y macroscópicos, con un

alto nivel de detalle. Inicialmente fueron creadas para el transporte, el tráfico y la sociedad civil ingenieros y por tanto, su diseño no estaba centrado en la generación de trazas de movilidad utilizables por simuladores de redes.

Una segunda limitación significativa de simuladores de tráfico es la complejidad de su calibración. El nivel de realismo que se da en la planificación del tráfico y del transporte es tan alto, que los usuarios potenciales necesitan ajustar una gran cantidad de parámetros, de los cuales cada uno de ellos tiene una influencia determinada en el tráfico vehicular. Este proceso requiere mucho tiempo, y en gran parte, les parece inútil a los investigadores de redes, ya que las simulaciones para redes no parecen requerir ese nivel de detalle para la movilidad vehicular. El objetivo de los modelos de movilidad vehicular es proporcionar los patrones de movilidad adaptados a las necesidades específicas para la simulación de redes. De este modo el conjunto de parámetros se ve significantemente reducido, por lo que la calibración del simulador es más rápida.

Por último nos queda hacer referencia a que la licencia comercial que en muchos casos se debe adquirir, puede suponer una gran inversión del capital destinado al proyecto. Por esta razón, una gran parte de la comunidad empezó a trabajar en el desarrollo de modelos "libres" de movilidad vehicular adaptados a la investigación de la red, y que prevén interacciones con simuladores de red y serán precisamente estos simuladores a los que nos referiremos a lo largo de este capítulo.

Pasemos a ver las distintas herramientas para la generación de movilidad e tipo *OpenSource* y *Freeware*:

Texas Model

El modelo de simulación de tráfico en intersecciones TEXAS Model (*Traffic EXperimental and Analytical Simulation Model*) (20) fue desarrollado por el "*Center for Highway Research*" de la Universidad de Texas (Austin, EE.UU.) en los años 70. Se distribuye bajo licencia de software libre, y tanto el código fuente como su documentación de desarrollo están disponibles en Internet. (21)

Se centra en el estudio del comportamiento del tráfico en intersecciones de vías y en sus aproximaciones. Para ello, simula los efectos que puedan tener diferentes situaciones como un cambio en la geometría de la intersección, en las características de los vehículos y los conductores que la atraviesan, en las condiciones estadísticas del flujo del tráfico, en los distintos controles de acceso a la intersección o en la temporización de los semáforos presentes. El modelo TEXAS puede aplicarse por tanto, para la evaluación previa de nuevos diseños o para determinar la influencia que diversos cambios en el entorno tengan sobre las intersecciones ya construidas, de forma que su comportamiento frente al tráfico mejore. Los resultados que aporta el programa pueden suponer un gran ahorro de recursos en comparación con los obtenidos mediante los actuales estudios de campo, y la verosimilitud de los datos ha sido contrastada mediante la realización de los correspondientes estudios estadísticos.

SUMO

Esta herramienta denominada *Simulation of Urban MObility* (SUMO) (22), altamente portátil, permite la simulación de tráfico para manejar grandes redes de carreteras. Para la parte del generador de tráfico, la ruta puede ser importada de diversas fuentes, así como ser generada por el usuario según sus necesidades. SUMO es también capaz de generar trazas de salida directamente utilizables por los simuladores de red. Con el fin de facilitar la configuración de SUMO, también brevemente de la movilidad vehicular de modelo del generador para Redes.
Para la parte de generación del tráfico, la asignación de rutas se puede o bien generar manualmente o bien ser importadas de los datos de la vida real. Utiliza el modelo Krauss de *car-following* y permite la importación de diversos mapas topológicos provenientes de herramientas como TIGER, GIS Arcview, OpenStretMap o incluso VISSIM. Las trazas de salida pueden ser utilizadas directamente por simuladores de redes, para esto se hace uso der la herramienta MOVE (*Mobility Model Generator for Vehicular Networks*) construida en base a SUMO, que permite transformar la salida del generador de tráfico en trazas legibles por parte de los simuladores de redes de manera inmediata. Además, MOVE proporciona una GUI que permite al usuario construir rápidamente escenarios de simulación realistas sin la molestia de escribir guiones de simulación, así como conocer los detalles internos del simulador. Para más información sobre SUMO se dirige al lector al capítulo 4, de este proyecto.

SmartPath Traffic Simulator

SmartPath (23) proporciona un entorno de simulación para probar diferentes diseños de control, llevar a cabo la evaluación y la observación de la interacción entre los controladores de vehículo y las autovías. Esta herramienta ha sido desarrollada en la UC Berkeley y genera trayectorias de vehículos según modelos validados de *car-following*. Está programada en SHIFT que es un lenguaje de programación con la semántica de simulación, que se utilizó en SmartAHS como un medio para la especificación, simulación y evaluación del modelado y control de sistemas automatizados para la autopista (AHS).

SmartPath sustituye al proyecto anterior siendo la versión 3.0 del mismo.
La restricción más importante de este simulador es su limitación a la modelización de los segmentos de carreteras, la falta de topología modelización en las restricciones de movimiento, y la falta de viaje y bloques de ruta de movimiento.

FreeSim

Se trata de un simulador programado en Java por la Universidad del sur de California (24) que permite la representación y la carga de escenarios como estructuras graficas de múltiples sistemas de carreteras, para llevar a cabo simulación de tipo macroscópica y microscópica. Se pueden también crear algoritmos para el tráfico y éstos pueden ser aplicados bien en toda la red o sólo para los vehículos o nodos indicados. Los datos de entrada que utiliza la herramienta de simulación de tráfico pueden ser generados por los usuarios o bien importados de fuentes que proporcionen datos en tiempo real de alguna organización de transporte. (25)

STRAW

Este simulador ha sido desarrollado por Universidad de Ilinois, Northwestern Evanston (26). Sus siglas corresponden con las *Street Random Waypoint*, y está basado en el simulador de redes *wireless* SWANS, cuya descripción veremos más adelante. Contiene un bloque complejo de constantes de movimiento que se obtienen de las topologías urbanas al ser importadas de la base de datos TIGER. El simulador de tráfico se basa en el modelo de *car-following* de Nagel-Schreckenberg. Y contempla diversas implementaciones de protocolos de transporte, enrutamiento y acceso al medio que no se encuentran originariamente presente en SWANS. El principal inconveniente actual de esta herramienta es que el uso del simulador de redes SWANS no está tan difundido como sus competidores y es por esto que queda un poco en segundo plano a la hora de tener la oportunidad de contener una mayor variedad de protocolos y conceptos de comunicaciones.

VanetMobiSim

El generador de movilidad a continuación fue desarrollado por el Institut Eurécom junto con el Politecnico di Torino (27). VanetMobiSim (28) es una extensión de la herramienta CanuMobiSim, que se centra en la movilidad vehicular, y las características realistas de modelos de movimiento del automóvil, tanto a nivel macro como microscópico. VanetMobiSim está destinado a ampliar la movilidad vehicular apoyando de la herramienta CanuMobiSim, proporcionando un mayor grado de realismo en las simulaciones. En lo que a macro movilidad se refiere se tienen en cuanta la topología y características de las carreteras y la presencia de señales de tráfico. A la hora de definir las topologías se dota a la herramienta con la capacidad de importar mapas de la base de datos TIGER (29), así como la posibilidad de crear mapas mediante los grafos de Voronoi (30). Esta herramienta en el nivel microscópico incluye os modelos de movilidad: *Intelligent Driver Model* (con o sin cambios de carril), junto con IM (*Intersection Management*). VanetMobiSim se basa en Java y puede generar trazas de movimiento en diferentes formatos, para luego ser utilizadas por diferentes simuladores de red como son: ns-2 (31), GloMoSim (32), y QualNet (33) .

CityMob

Esta herramienta ha sido creada por el grupo de redes de computadores de la Universidad Politécnica de Valencia (34) y las trazas generadas pueden alimentar al simulador de redes ns-2 (31). CityMob es un generador de movilidad especialmente diseñado para investigar los diferentes modelos de movilidad en VANETs, y su impacto en el rendimiento de la comunicación intervehicular. CityMob crea escenarios de movilidad urbana y simula los coches dañados utilizando la red para enviar información a otros vehículos, tratando de evitar accidentes o embotellamientos. Existen tres modelos de movilidad que combinan un cierto nivel de aleatoriedad cuando se trata de representar algunos entornos realistas. Los modelos son:

1) El modelo simple (SM): los modelos verticales y horizontales de acuerdo a pautas de movilidad sin cambios de dirección.

2) El modelo de Manhattan (MM): se modela la ciudad como una rejilla estilo Manhattan, con un tamaño de bloque uniforme en toda la simulación de la zona.
3) El modelo de centro de la ciudad (DM): este modelo añade la densidad del tráfico en el modelo de Manhattan. En una ciudad real, el tráfico no se distribuye uniformemente, si no que existen zonas con una densidad de vehículos más alta. Estas zonas se localizan generalmente en el centro de la ciudad, y los vehículos deben moverse más lentamente que en las afueras.

El usuario tiene que establecer el modelo de movilidad, el número total de nodos, el tiempo de simulación, el tamaño del mapa, la velocidad máxima y la distancia entre las calles consecutivas así como el número de nodos dañados (35).

Monarch

El proyecto MONARCH "*MObile Networking ARCHitectures*" (36) ha sido desarrollado en la Rice University, de Houston y propone una herramienta que extrae topologías para carreteras de mapas reales obtenidos de la base de datos TIGER. Se trata básicamente de un modelo de movilidad del tipo *random way point*, el cual está restringido a funcionar en topologías extraídas de ciudades reales. Permite extraer trazas útiles para ser introducidas en ns2. Como dato de investigación comentar que gracias al proyecto Monarch dicho simulador de redes, ns-2, fue extendido para ser usado en redes tipo wireless, convirtiéndose en la referencia para las redes MANET.

MITSIMLab

MITSIMLab desarrollado en el MIT (37), y programado en C++, es una herramienta de simulación que fue desarrollada para evaluar los impactos de los distintos diseños de tráfico del sistema de gestión a nivel operativo y para ayudar en el refinamiento del diseño posterior. MITSMLab representa una amplia gama de diseños de sistema de gestión de tráfico, modelos de la respuesta de los conductores a la información del tráfico en tiempo real de los controles; e incorpora la interacción dinámica entre el sistema de gestión del tráfico y los controladores en la red. Los diferentes elementos del MITSIMLab se organizan en tres módulos:

1. Simulador de Tráfico Microscópico (MITSIM).
2. Simulador de Gestión de Tráfico (TMS).
3. Interfaz gráfica de usuario (GUI).

Esta herramienta, representa una amplia gama de diseños de sistema de gestión de tráfico, modelos de la respuesta de los conductores a la información del tráfico en tiempo real y de control, e incorpora la interacción dinámica entre el sistema de gestión del tráfico y los controladores en la red.

VISSIM

Esta herramienta fue desarrollada en Java en el Imperial College de Londres (38). El proyecto tiene su objetivo en proporcionar una implementación para aproximar el movimiento urbano

de los vehículos bajo una simulación microscópica. Se pueden realizar pruebas *on-line*, siendo la principal desventaja de que se trata sólo de un ejercicio académico y por lo tanto carece de numerosas características y no es escalable.

Mobitools

MobiTools es una *suite* de movilidad que fue desarrollada en la Universidad de UCLA (39), está programada en Java e incorpora el uso de mapas reales para la creación de trazas de movilidad (MobiMap / MobiDense), un visualizador para mapas (MobiView), y un módulo para calcular los efectos de propagación (MobiRay) (40). MobiView es una herramienta de visualización para redes ad hoc vehiculares. MobiView se basa en la NASA World Wind, una aplicación de código abierto que permite acceder a las imágenes Landsat de la NASA. MobiView permite situar gráficamente los vehículos en un mapa y calcular visualmente la conectividad de red y el realismo de movilidad así mismo, es posible hacer zoom y ver el mapa desde distintos ángulos, mientras que la animación está funcionando. Es una herramienta compatible con ns2 y con Qualnet (33). Así como con Google maps (41) y con la base de datos TIGER (29).

Vergilius

VERGILIUS es una herramienta para el estudio de redes *wireless ad-hoc* desarrollada entre las universidades de California (UCLA) (42) y la universidad de Bolonia en Italia. Esta herramienta Vergilius consiste en una *toolbox* diseñada para hacer más eficiente la generación de las trazas de movilidad y el cálculo de las pérdidas de propagación en escenarios de redes vehiculares.

Esta herramienta usa los mapas digitales TIGER para generar escenarios centrados especialmente en ciudades tanto para SUMO como para CORSIM. Además, la herramienta de propagación Vergilius calcula la matriz de atenuación teniendo en cuenta la información cartográfica digital de los mapas ciudad de bloque y la información hoja de ruta extraída de la base de datos Tiger. Sus principales componentes son:

Una herramienta para la extracción de mapas TIGER.

Un generador de escenarios para los simuladores de la movilidad (CORSIM (43) y SUMO).

Un módulo de predicción de pérdidas por trayecto compatible con QualNet.

Un analizador de trazas para identificar las características de los patrones de movimiento generados

Translite

Basada en SUMO, esta herramienta genera trazas de movilidad para ser insertadas en n-s2 (31). Utiliza mapas de tipo TIGER, OpenStreetMap (44) y Google Earth (45). Está programada en Java y fue desarrollada en el Ecole Polytechnique Fédérale de Lausanne (46). Se considera una versión simplificada de la herramienta TraNs (46)que veremos más adelante.

Microsimulation of Road Traffic Flow Tool

Por último mencionaremos la herramienta diseñada por Martin Treiber en la Universidad de Dresden (47), disponible para su descarga así como para ser visualizada *on-line* mediante una aplicación de Java. Dispone de distintas configuraciones de las carreteras así como dos modelos de tráfico el IDM (*Intelligent Drive Model*) y el *lane-changing model*.

III.III.II). Network Simulators

Como alternativa a los simuladores de red comerciales presentamos a continuación los de tipo *freeware*. Hay que tener pues en cuenta que la comunidad que está detrás del uso de un simulador en concreto determina significativamente los distintos protocolos disponibles en el mismo así como sus funcionalidades, a más usuarios mayores funcionalidades tendremos para nuestro uso.

NS-2

El simulador de redes más utilizado y conocido es ns-2 (31), su popularidad no se debe a su simplicidad o eficiencia, sino a su modularidad y su universalidad. Como ya comentamos el proyecto Monarch lo extendió, ampliando su horizonte hacia redes *wireless*. Contiene las principales capas de la pila OSI, los principales protocolos de MANETs y las capas físicas MAC han sido recientemente extendidas y validadas para el modelado de redes vehiculares.

GloMoSim

GloMoSim (32) es una alternativa a ns2, siendo la versión gratuita de QualNet, pero con capacidades reducidas. Tiene las mismas funciones y objetivos que ns-2. El conjunto de protocolos disponibles es muy reducido debido a la menor penetración de esta herramienta en la comunidad.

JIST-SWANS

Es un simulador de red escalable codificado en Java, que ha sido desarrollado en la Universidad de Cornell, USA (48). A pesar de haber sufrido una popularidad limitada en un principio, resurgió y actualmente ha conseguido llegar a una gran comunidad conteniendo los principales protocolos de MANETs. Se define como un simulador de redes escalable basado en la plataforma JiST. JiST es un máquina de altas prestaciones de simulación de eventos discretos que corren en la máquina virtual de Java. Su principal ventaja es que proporciona las mismas prestaciones que ns2 y GlomoSim, pero permite simular redes con una mayor cantidad de nodos.

GTNetS

Desarrollado en el instituto tecnológico de Georgia, el simulador Simulator (GTNetS) (49) constituye un entorno de simulación que permite estudiar el comportamiento de redes de moderada a gran escala. GTNetS se encuentra en estricta conformidad con las distintas capas de los protocolos, lo que lo hace una solución más sencilla a la hora de desarrollar protocolos para MANETs.

OMNeT++

OMNeT++ (50) es una herramienta extensible y modular, basada en componentes de la biblioteca de C ++ y dedicada principalmente a la construcción de simuladores de red. "Red" se entiende en un sentido más amplio ya que incluye no solo redes cableadas e inalámbricas, sino redes dentro del chip, redes de colas, redes de sensores, redes inalámbricas ad-hoc, redes fotónicas, etc. OMNeT++ ofrece un IDE Integrated development environment basado en Eclipse, un entorno de ejecución gráfico, y es anfitrión de otras herramientas. Existen extensiones para simulación en tiempo real, emulación de la red, lenguajes alternativos de programación (Java, C #), la integración de bases de datos, integración de SystemC, y otras funciones. OMNeT++ es libre para uso académico y es ya una plataforma ampliamente utilizada por la comunidad científica mundial.

SNS

SNS (*Staged Network Simulator*), desarrollado en la universidad de Cornell (51). Tradicionalmente los simuladores de redes inalámbricas están limitados en velocidad y en escalabilidad, ya que realizan muchos cálculos redundantes. La técnica de simulación por etapas propone eliminar los cálculos redundantes a través de la función de almacenamiento en caché y reutilizar. La idea central es almacenar en caché los resultados de las operaciones costosas y reutilizarlos siempre que sea posible. SNS está basado en ns-2.

III.III.III) Simuladores integrados VANET

En esta sección veremos las herramientas VANET de tipo integrado que están disponibles de manera abierta a la comunidad investigadora. Tal y como se describe en la Sección III.II.III, estas herramientas integrados contienen en su interior un único modelo de las capacidades de movilidad y de simulación de red.

GrooveSim

Esta herramienta fue desarrollada en la Universidad de Pensilvania junto con la universidad de Carnegie Mellon en Pittsburg y la empresa automovilística General Motors. (52)
GroveNet, programado en Java, es el proyecto central de la modelización y visualización de la movilidad para vehículos, mientras que GrooveSim es el modelo de movilidad en sí. GrooveNet es un simulador híbrido que permite la comunicación entre vehículos simulados, vehículos reales y entre los vehículos reales y simulados. Al modelar la comunicación entre vehículos en una topografía real de la calle basada en mapas se facilita el diseño del protocolo. La arquitectura modular de GrooveNet incorpora la movilidad, el viaje y los modelos de difusión de mensajes sobre una variedad de modelos de comunicación de las capa de enlace y física. Con esta herramienta es sencillo realizar simulaciones de miles de vehículos en cualquier ciudad de los EE.UU., así como agregar nuevos modelos de redes, de seguridad, de aplicaciones de interacciones intervehiculares. GrooveNet soporta múltiples interfaces de red, GPS y eventos disparados desde la computadora a bordo del vehículo. A través de la simulación, permite estudiar la latencia de los mensajes, y la cobertura bajo distintas condiciones para los tipos de tráfico.

NS3

En la universidad de Old Dominion de Norfolk (53) han desarrollado la integración de un modelo de movilidad junto con una aplicación para autovías integrada en la herramienta por ns-3, la próxima generación de la popular ns-2 en red (54). En este simulador la movilidad de vehículos y comunicación en red están integrados a través de eventos. Los manejadores de eventos creados por el usuario pueden enviar mensajes de red o alterar la movilidad de los vehículos cada vez que se recibe un mensaje de red o bien cada vez que la movilidad se ve modificada por el modelo de movilidad.

NCTUns

NCTUns que es un simulador de red extensible así como un emulador capaz de simular distintos protocolos utilizados, tanto en redes cableadas como en inalámbricas (55). NCTUns proporciona muchas ventajas únicas que no pueden ser fácilmente alcanzadas por simuladores de redes tradicionales, como ns-2 y OPNET (56), está desarrollado en la Universidad Chiao Tung de Taiwan. Esta herramienta no se centra específicamente en la movilidad de vehículos, sino que proporciona una gama completa de la pila de herramientas de simulación de red . NCTuns contiene funcionalidades suficientes para el modelado de la movilidad de vehículos tales como patrones de conducción humana, *car-following* y control para las intersecciones. A parte de tener propiedades de simulador también las tiene de emulador lo cual permite una evaluación más precisa de los protocolos de redes. La parte de red contempla la capa de acceso al medio IEEE WAVE 802.11p para entornos vehiculares.

III.III.IV) Simuladores híbridos VANET

Los simuladores integrados tienen el potencial de ofrecer características muy avanzadas para modelar el movimiento de vehículos y capacidades de red, pero también tienen la limitación de una configuración compleja. La comunidad investigadora trató de aunar ya simuladores de red y de movilidad existentes y validados, con el claro objetivo de aprovechar la larga experiencia obtenida anteriormente por separado en ambos campos.

TRANS

Herramienta escrita en Java y desarrollada en el Ecole polytechnique Fédérale Lausanne (46), cuyo enfoque se basa en el generador de movilidad SUMO aunado con el simulador de redes ns-2. Hace uso de una interfaz Interpreter4 de tal manera que las trazas extraídas de SUMO se transmiten a ns-2, y viceversa las instrucciones de ns-2 se envían a SUMO para ajustar el tráfico rodado. Las interacciones entre el tráfico de vehículos y la red pueden ser aplicado en consecuencia. Las trazas obtenidas de SUMO, se consideran de las más realistas entre las disponibles en la actualidad. Si bien se trata de un proyecto sin pensamiento de seguir siendo desarrollado.

MobiREAL

MobiREAL es un simulador novedosos de para la sociedad con dispositivos móviles. Está programado en C++ y ha sido desarrollado en Osaka University en Japón (57). Permite simular la movilidad real de los seres humanos y los automóviles, cambiando su comportamiento en función de un contexto de aplicación dado y así obtener una evaluación detallada de las aplicaciones de red, protocolos de enrutamiento, las infraestructuras, etc. MobiREAL permite simular redes móviles ad-hoc mediante la adición modelos de movilidad a un simulador de redes GTNetS. Es un simulador de red que puede simular la movilidad realista de las personas y vehículos, y permitir el cambio de su comportamiento en función de una determinada aplicación VANET. MobiREAL puede describir la movilidad de nodos utilizando C. MobiREAL Animator visualiza de forma dinámica el movimiento del nodo, la conectividad, y la transmisión de paquetes. Esto mejora la comprensión de los resultados de la simulación de manera intuitiva. Con el uso de MobiReal, es posible simular una mezcla de diferentes modelos de movilidad al mismo tiempo. Para la movilidad vehicular, los autores han modificado un simulador de tráfico llamado NetStream que fue desarrollado por TOYOTA. Dado que es un NetStream software propietario, su principal inconveniente es que los usuarios no pueden acceder y modificar ésta parte del simulador, limitando, por tanto, su uso.

VEINS

Hablaremos más detenidamente de esta herramienta en el Capitulo V de este proyecto si bien daremos una pequeña introducción. Las siglas significan "*Vehicles in Network Simulation*" (58) y se define como un marco de simulación compuesto por un simulador de redes basado en eventos (OMNET++) y un generador de movilidad microscópico (SUMO). Ambos dominios se encuentran bidireccionalmente acoplados y trabajan en tiempo real, de tal manera que la comunicación intervehicular tiene efectos en la movilidad del tráfico rodado.

IV-SUMO, EL SIMULADOR DE TRÁFICO RODADO

En el cuarto capítulo de este proyecto “bucearemos” por el mundo de simulación de tráfico rodado para introducirnos de lleno en una herramienta ampliamente conocida y utilizada por la comunidad académica internacional que se encuentra continuamente en proceso de desarrollo y que promete brindarnos grandes alternativas de trabajo para la modelización vehicular, SUMO.

Simulation of Urban MObilty Tool (SUMO)

La herramienta *Simulation of Urban MObilty Tool* (SUMO) (59) consiste en una herramienta especialmente atractiva para su uso y estudio en el campo de la simulación vehicular ya que se trata de un *software* altamente portable, de tipo *freeware y opensource*. Consta tanto de una consola que permite introducir los comandos correspondientes para ejecutar las simulaciones así como de una interfaz gráfica (GUI) que permite visualizar e interactuar con el *output* de la simulación de tráfico. Se encuentra en continuo desarrollo por parte del instituto de sistemas de transporte perteneciente al centro aeroespacial alemán (60). Este proyecto comenzó en el año 2000, apareciendo en 2001 la primera versión ejecutable continuando hasta el día de hoy en desarrollo siendo una herramienta utilizada en numerosos proyectos de alcance internacional.

IV. I.) ¿Cómo es un simulador de tráfico rodado?

Antes de comenzar a describir SUMO es importante que conozcamos desde dentro los principales componentes que se pueden distinguir en un generador de movilidad (61). Veamos pues, apoyándonos en la siguiente figura, cuales son los componentes básicos de un simulador de tráfico rodado:

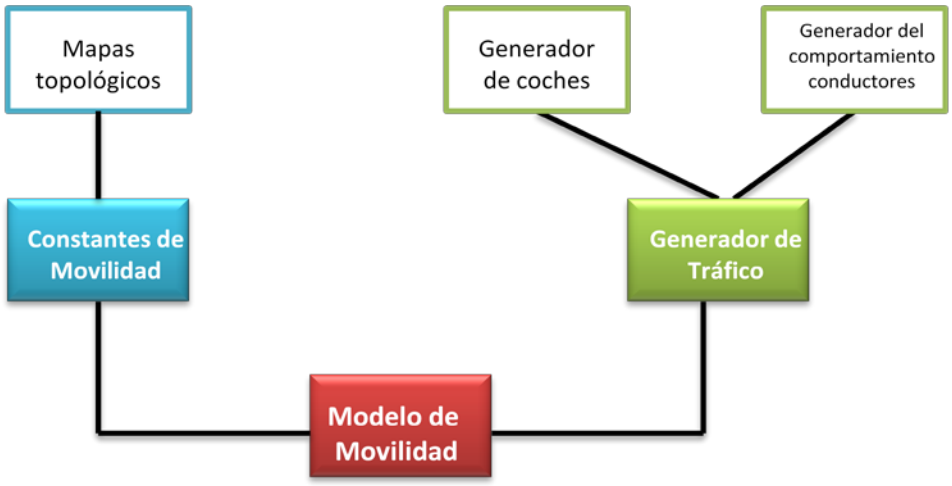


Figura 20 Componentes básicos de un simulador de tráfico

A) CONSTANTES DE MOVIMIENTO: Se entienden como todos los elementos pertenecientes a la topología de las vías que pueden influenciar en los patrones de movilidad de los vehículos. Se definen por tanto las siguientes:

A.1) Gráficos: se refiere a la topología que define el escenario donde ocurren las simulaciones y principalmente se pueden distinguir:

- Los definidos por el usuario de manera artesanal, es decir conectado vías mediantes puntos de unión.
- Los aleatorios generados por la máquina de simulación.
- Los mapas reales importados de diversas fuentes en función a los estándares topológicos.

A.2) Origen y Destino: siendo de tipo aleatorio, definido por el usuario, en función de los mapas o relacionados con sitios de interés.

A.3) Política de intersecciones: de tal manera que se ven modelados los cruces dentro de las topologías definidas con sus correspondiente regulaciones como son semáforos o señales de tráfico.

B) GENERADOR DE TRÁFICO: como su propio nombre indica es el responsable de producir el tráfico y se compone principalmente de:

B.1) Generador del viaje: bien de manera aleatoria o establecida por el usuario.

B.2) Cálculo de caminos: algoritmos necesarios para generar el camino del origen al destino.

B.3) Patrones de movilidad: Macro-Micro-Meso-Submicro. Es necesario que entendamos el concepto más generalista que define estas herramientas, es decir, es los modelos de movilidad vehicular. La clasificación de éstos es como sigue:

Modelos macroscópicos: estos modelos describen grandes cantidades de interés como son: densidad vehicular, y velocidad media aplicando la teoría hidrodinámica de fluidos al comportamiento de los vehículos. En este tipo de modelos la simulación se realiza partiendo el espacio de simulación en secciones, en lugar de tener en cuenta el comportamiento de cada uno de los vehículos como en el caso de los modelos tipo micro; se requieren por tanto menos recursos de computación que para estos últimos si bien se obtiene menor nivel de detalle.

Modelos microscópicos: los modelos micro simulan el movimiento de cada uno de los vehículos existentes en el escenario de simulación asumiendo que el movimiento de éstos depende no sólo de las características físicas y motoras del propio vehículo sino también del control llevado a cabo por el conductor.

En este punto conviene que introduzcamos el llamado concepto de *car-following*, el cual implementa de manera microscópica los patrones de movimiento en los cuales entran en juego los factores humanos. A la hora de generar el tráfico este modelo de *car-following* incluye el realismo dado a la movilidad de vehículos, considerando que el vehículo no sólo se moverá de una lado a otro sino que está sometido a una serie de interacciones con los demás vehículos, así como que se encuentra bajo el control del ser humano a la hora de generar el tráfico. El funcionamiento de este mecanismo es intuitivo; consiste en adaptar la movilidad del coche siguiente en función de una serie de reglas con tal de evitar que se produzca algún choque con el coche a la cabeza de éste. Un esquema ilustrativo de este mecanismo es el que sigue (61):

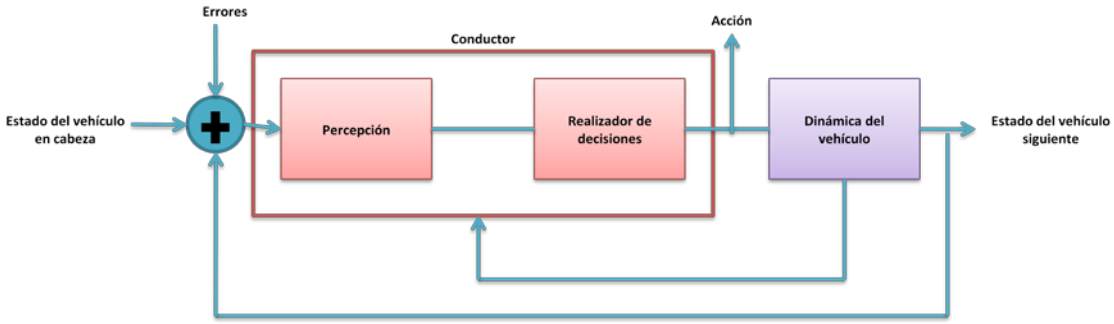


Figura 21 Diagrama de bloques para *car-following*

Es importante diferenciar para las simulaciones entre espacio continuo y discreto. En el primer caso cada vehículo tiene una determinada posición que viene descrita por un número en coma flotante mientras que para el espacio discreto las simulaciones son de carácter celular. Es decir las vías se dividen en celdas de una determinada anchura y los vehículos van saltando de una celda a otra como se muestra en la figura a continuación:

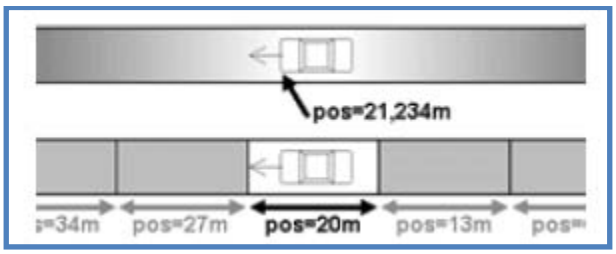


Figura 22 Espacio continuo VS discreto

Modelos mesoscópicos: estos modelos se encuentran en la frontera entre los modelos macro y microscópicos y combinan por tanto propiedades de ambos. La unidad de simulación en este caso es al igual que en los modelos microscópicos la del vehículo de manera individual. Sin embargo, los movimientos de éstos siguen una aproximación de tipo macroscópico. Por ejemplo no se considera la velocidad dinámica individual del vehículo sino que se asume un valor medio ponderado a lo largo del viaje en un determinado tramo.

Modelos sub-microscópicos: por último nos encontramos con los modelos submicroscópicos basados en el modelo micro, pero teniendo en cuenta a su vez las distintas partes del coche, dividiendo el mismo en subestructuras. Se considera por tanto la velocidad de rotación del motor condicionada a la velocidad del coche o a los movimientos del cambio de marchas del conductor. De esta manera se logra un mayor detalle utilizado sobre todo en técnicas de diseño, si bien requiere una gran capacidad computacional.

B.4) Cambios de carril: definiendo los tipos de adelantamientos.

B.5) Manejo de intersecciones: modo de comportamiento en los cruces dado por las constantes de movimiento.

Según la clasificación expuesta y a grandes rasgos, en relación a las constantes de movimiento podemos afirmar que SUMO permite gráficos definidos por el usuario, de tipo *Grid*, *Spider* o aleatorio o bien importados de mapas tipo (62):

- ArcView networks (shapefiles)
- Elmar's networks
- TIGER
- OpenStreetMap
- Robocup Simulation League
- VISUM (Vissim)

Nuestra herramienta bajo estudio contempla diversos carriles, así como limitaciones de velocidad, la opción de generar rutas entre origen y destino de manera aleatoria así como la de indicar puntos especialmente atractivos, además, dentro de las intersecciones hay un política de señales de stop y de semáforos. En lo que respecta al generador de tráfico, las asignaciones de rutas pueden ser bien de manera aleatoria, bien mediante viajes sin indicar concretamente todas la vías por la cuales se deben circular, sino simplemente indicando origen y destino o bien especificando paso a paso las distintas vías que recorrerá el vehículo. SUMO hace uso del modelo de *car-following* denominado Krauss Model (63). SUMO consta a su vez de diversas interfaces que le permiten conectarse con simuladores de redes como por ejemplo: ns-2 (64), QualNet (65) y OMNeT++ (66) y su plataforma de programación es C++, permitiendo la simulación vehicular en entornos urbanos y de autovías. Además SUMO está validado esto es, los patrones de movilidad han sido comparados con las topologías reales midiendo el error que existe entre ambas. Normalmente si este proceso no es posible debido a que las trazas del mundo real no están disponibles se puede delegar dicha validación haciendo uso de trazas de un modelo que si está modelado en lugar de las reales.

IV. II) Instalación y ejecución de SUMO

El primer paso será el de decidir si se quiere solamente instalar SUMO o a su vez también extenderlo mediante programación del mismo. En nuestro caso en principio la meta será sólo hacer uso de la herramienta generadora de tráfico por lo tanto seguimos los siguientes pasos para su instalación. (Nota: es importante tener presente que deben tenerse previamente instalados los paquetes *Python* y *Perl*). Anotar que la última versión de la herramienta SUMO es la 0.12.0 y data del 27 de mayo de 2010 (67), y será la que utilicemos en este proyecto. Una vez hayamos descomprimido los paquetes binarios obtendremos el siguiente ejecutable: `sumo.exe` el cual ejecutaremos en el directorio que fue instalado para asegurarnos así que fue correctamente instalado.

A continuación en la siguiente figura vemos una captura de la línea de comandos donde se observa como SUMO ha sido instalado y ejecutado exitosamente.

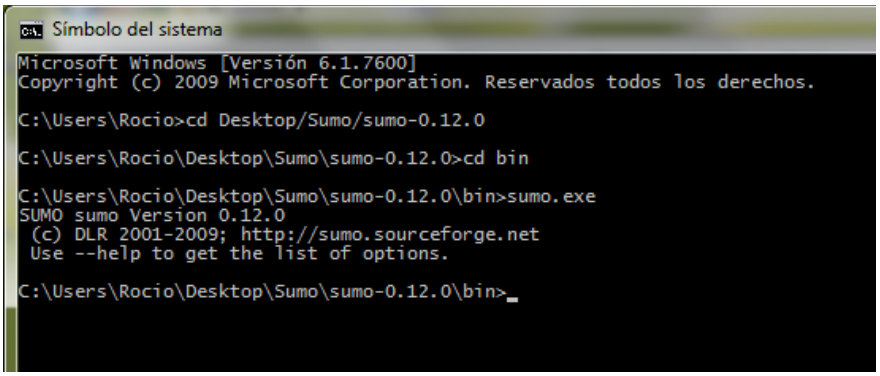


Figura 23 Línea de comandos para SUMO

IV. II.I) Los paradigmas del software

A la hora de diseñar SUMO se tuvieron en cuenta dos pilares principales, por un lado la ejecución rápida y por otro la portabilidad.

Una aproximación que cumple ambas condiciones es que el software se ejecute mediante línea de comandos, característica que hoy sigue manteniéndose si bien se ha añadido una interfaz gráfica para mejorar la visualización de resultados en caso de que sea necesario.

Otra de las condiciones para asegurar velocidad en el cálculo era la de dividir las funcionalidades del programa en diversos bloques encargados de computar de manera independiente cada una de ellas, más adelante veremos dichas funcionalidades para los distintos bloques así como la interdependencia de unos con otros. La ventaja como ya se ha comentado, es que se pueden usar estructuras de datos ajustadas para cada bloque de manera más rápida, si bien esto puede resultar en el inconveniente de convertir a SUMO en una herramienta no tan sencilla y cómoda de utilizar en comparación a otros paquetes de simulación con propósitos similares.

IV. II.II) Aplicaciones incluidas en SUMO

SUMO es un paquete de software que incluye diversas aplicaciones necesarias para preparar la simulación microscópica de tráfico rodado. Este paquete incluye las siguientes aplicaciones mostradas en la tabla a continuación:

APLICACIÓN	DESCRIPCIÓN
SUMO	Aplicación que genera la simulación microscópica sin visualización gráfica.
GUISIM	Aplicación para la interfaz gráfica.
NETCONVERT	Importador y generador de redes, lee las redes viales desde diferentes formatos y las convierte a un formato entendible por SUMO.
NETGEN	Genera redes abstractas para la simulación de SUMO.
DUAROUTER	Calcula las rutas más rápidas a través de la red, importando diversos tipos de demandas.

JTRROUTER	Calcula las rutas usando los porcentajes de giro en las intersecciones.
DFROUTER	Calcula las rutas obtenidas por los bucles de inducción.
OD2TRIPS	Descompone las matrices OD en viajes únicos de vehículos.
POLYCONVERT	Importa los puntos de interés y polígonos de diferentes formatos y los traduce en una descripción visible por GUISIM.

Figura 24 Aplicaciones incluidas en SUMO

El esquema siguiente muestra interrelaciones existentes entre las distintas aplicaciones:

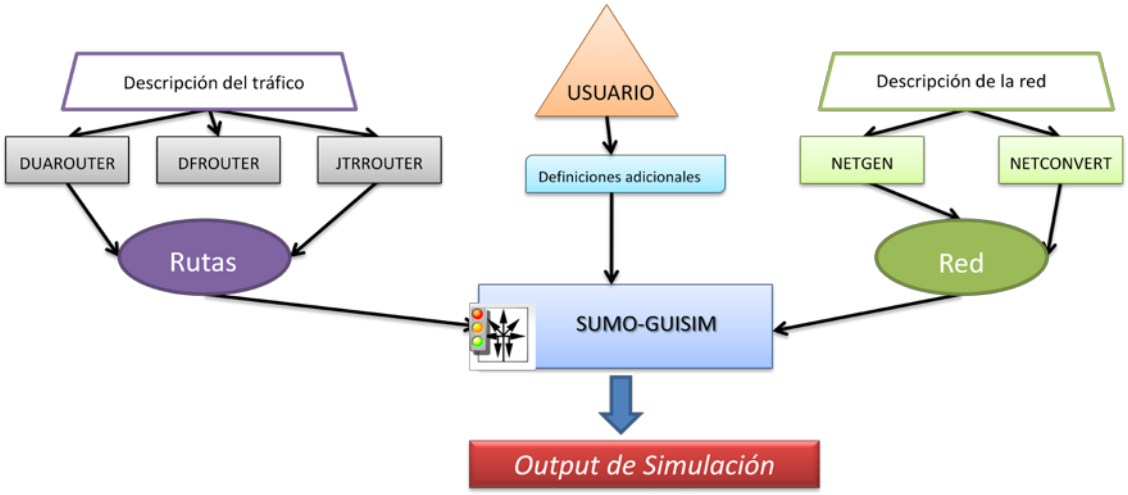


Figura 25 Esquema de las aplicaciones en SUMO

IV. II.III) Convenio de nombres para los archivos de configuración

En función de la aplicación, los archivos de configuración tienen extensiones diversas y es importante tener en cuenta siempre este punto, cómo se muestra seguidamente:

- *.sumo.cfg: Archivo de configuración para SUMO y GUISIM
- *.netc.cfg: Archivo de configuración para NETCONVERT
- *.netg.cfg: Archivo de configuración para NETGEN
- *.rou.cfg: Archivo de configuración para DUAROUTER
- *.jtr.cfg: Archivo de configuración para JTRROUTER
- *.df.cfg: Archivo de configuración para DFROUTER
- *.od2t.cfg: Archivo de configuración para OD2TRIPS

IV. II.IV) Datos necesarios para comenzar una simulación en SUMO

Como es de esperar la primera información sin la cual no podremos empezar una simulación de tráfico rodado será aquélla que ilustre la red vial donde van a suceder los acontecimientos.

Es decir, el primer paso será o bien crear una red de vías definida por el usuario de manera aleatoria o bien importada de otras fuentes. El segundo aspecto a considerar en cuanto a *inputs* se centra en cómo se van a desplazar los nodos por la red vial que planteamos, es decir el tráfico de vehículos. En el caso de SUMO cada vehículo conoce su ruta de manera individual, contiene su lista de origen-destino así como cada tramo que atraviesa en la red. Los pasos a seguir para crear una simulación son los que muestra el siguiente esquema:

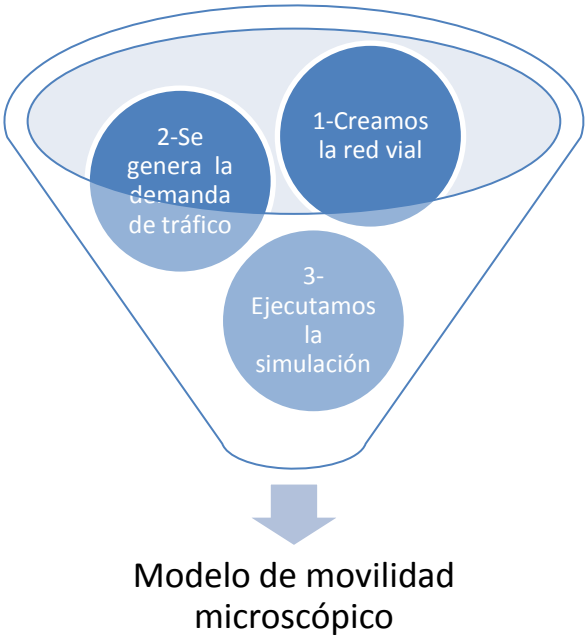


Figura 26 Procedimiento para generar simulaciones

IV.III) Sumo: generando escenarios paso a paso

A partir de esta sección hablaremos en términos de tutorial y veremos paso a paso cómo generar un ejemplo de red a simular. Según lo expuesto anteriormente el primer paso para obtener un modelo de movilidad con SUMO es crear una red vial o *network*.

IV.III.I Generación de *networks*

Haciendo uso del bloque NETGEN podremos obtener las siguientes redes en SUMO:

a.1 Generación automática: la generación automática se basa como su propio nombre indica en la obtención de redes de manera automática por el simulador. Las posibles redes ofrecidas por SUMO son las siguientes:

- **Redes tipo *grid***

A la hora de generar estas redes (de tipo rejilla o entramado) hay que tener en cuenta el número de cruces que queremos en las coordenadas x e y, y la distancia que existe entre dichos cruces en función de las coordenadas, expresadas en metros. Es decir siguiendo los siguientes comandos:

`--grid-x-number`: número de cruces en el eje x

`--grid-y-number`: número de cruces en el eje y

`--grid-x-length`: distancia en metros entre los cruces en la coordenada x

`--grid-y-length`: distancia en metros entre los cruces en la coordenada y

`--grid-number and --grid-length` : si se desean los mismos valores para el eje x y el eje y

En lo que respecta a los cruces existentes dentro de nuestra red se puede elegir el tipo de señalización dentro del cruce y obteniéndose tres tipos distintos que podrán ser asignados mediante el comando `–junction` o bien `–j` en la forma abreviada.
Dos son las opciones disponibles:

1. `priority:` Donde los vehículos tienen que esperar hasta que los vehículos de la derecha hayan atravesado a la intersección
2. `traffic_light:` La intersección es controlada por un semáforo

Existe una opción para añadir calles de una determinada longitud en la frontera de la tabla *grid* de tal manera que todas las intersecciones contarán con 4 brazos de vías. Si bien la longitud en el eje x y en el eje y deben de ser iguales. El comando en línea a introducir es el siguiente: `--attach-length`.

El siguiente paso es el de hacer una llamada a NETGEN para que el simulador interprete mediante los comandos indicados el tipo de red en este caso *grid* que vamos a generar.
Luego para visualizar la red que hemos generado tenemos dos opciones o bien abrimos la interfaz gráfica y cargamos el archivo *.net.xml* generado por NETGEN o bien hacemos una llamada a SUMO directamente a través de un archivo de configuración de la forma `.sumo.cfg` que en este caso contendrá la siguiente información:

```
<configuration>

      <input>
          <net-file value="nombre de la red.net.xml"/>
      </input>
  </configuration>
```

Veamos a grandes rasgos las principales funcionalidades de la interfaz de simulación GUI para SUMO:

Interfaz de Simulación (Sumo-GUI)

Para observar los resultados de manera gráfica podemos hacer uso de la interfaz gráfica o GUI. Para realizar una llamada a ésta tenemos que introducir el siguiente comando:

`Sumo-gui` : para abrir la interfaz gráfica

`Sumo-gui –c *.sumo.cfg`: para abrir la interfaz gráfica con el correspondiente archivo de configuración cargado.

Una vez ejecutados los comandos y si todo funciona correctamente obtendremos nuestra ventana de simulación tal y como se muestra a continuación:

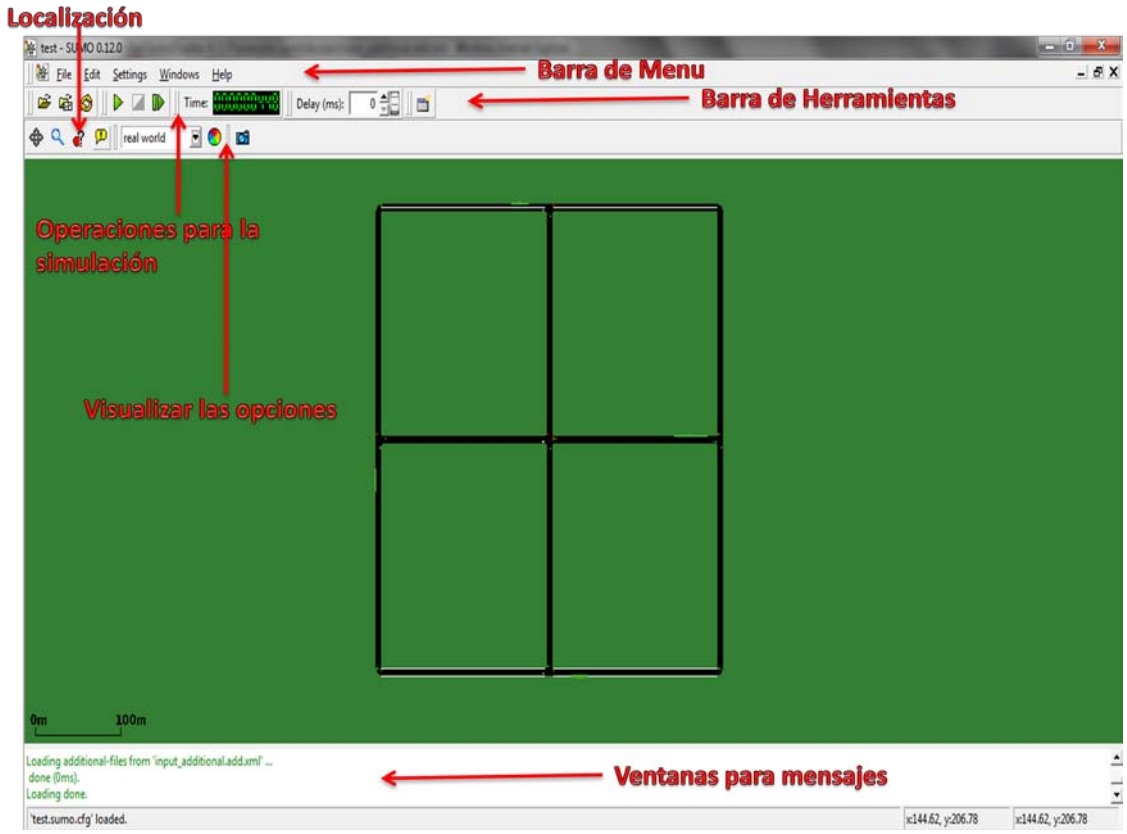


Figura 27 Captura de la GUI para SUMO

Las flechas en la figura anterior muestran parte de las funcionalidades más prácticas de la interfaz gráfica y sus características son las siguientes:

Barra de Menú

Se divide en dos grandes bloques el menú de archivo y el menú de herramientas:
Menú de archivo: en este menú podremos llevar a cabo las siguientes acciones:

- Abrir una simulación, que permite elegir un archivo con la extensión `.sumo.cfg`
- Abrir una red para ser visualizada, permitiendo elegir un archivo con la extensión `.net.xml`
- También proporciona la opción de recargar una simulación previa, para su nueva ejecución.

Menú de ventanas. Es el encargado de:

- Mostrar el estado en línea- Nos indica en qué punto nos encontramos de ejecución.

- Ventana de Mensaje- Permite que aparezcan los mensajes en la parte inferior de la ventana.

- Barra de herramientas- Del mismo modo nos deja ver o no la barra de herramientas en modo gráfico.

- Modo vertical, horizontal o cascada- Sirve para ordenar las ventanas de las distintas simulaciones en la misma pantalla de simulación de manera vertical horizontal o en cascada.

Barra de herramientas

La mayor parte de las acciones definidas en el menú de archivo están disponibles gráficamente mediante iconos en la barra de herramientas y como funcionalidad especialmente interesante tenemos las o*peraciones para la simulación:*
En esta parte de la interfaz disponemos de botones como play, y *stop* para facilitar las ejecuciones de las simulaciones. También se dispone de un botón para poder ejecutar ésta paso a paso. Para controlar la velocidad de la simulación es necesario jugar con el retraso o *delay* que introducimos, es decir el tiempo de espera entre dos pasos de simulación consecutivos. Lógicamente cuanto mayor sea este retraso más lenta irá la simulación. A la hora de visualizar el escenario es importante el menú para **visualizar las opciones** ya que contiene una imagen de nuestro entorno de simulación dado bien por colores, o por etiquetas con los nombres de los objetos allí presentes. Es imprescindible para poder detectar todos los componentes que forman parte de la ejecución ayudando visiblemente a la comprensión del entorno. Existen otros **botones de localización** que centran la vista de la pantalla principal en el elemento que deseamos (intersecciones, vías, vehículos, semáforos o estructura o forma adicional).

Veamos pues ahora algunos ejemplos relacionados con la generación de redes tipo *grid* de manera automática:

A) En este primer ejemplo generamos una red automática tipo *grid*, la cual contiene el mismo número de cruces en el eje x que en el eje y, un total de 5 y todos se encuentran distanciados del resto 100 metros, con lo cual obtendremos una red cuadrada tipo *grid.* El comando a introducir es el siguiente:

```
netgen --grid-net --grid-number=5 --grid-length=100 --output-
file=gridnet1tl.net.xml
```

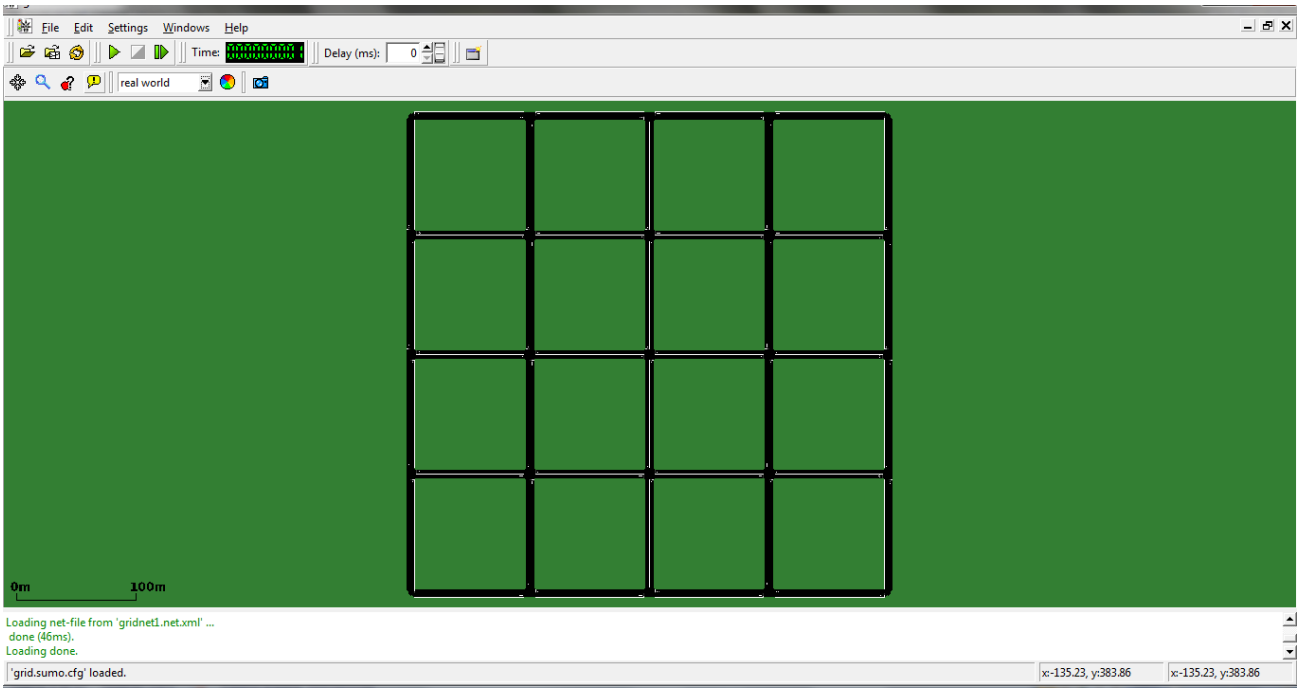


Figura 28 Red *grid* generada

Si ampliamos para ver una de las intersecciones:

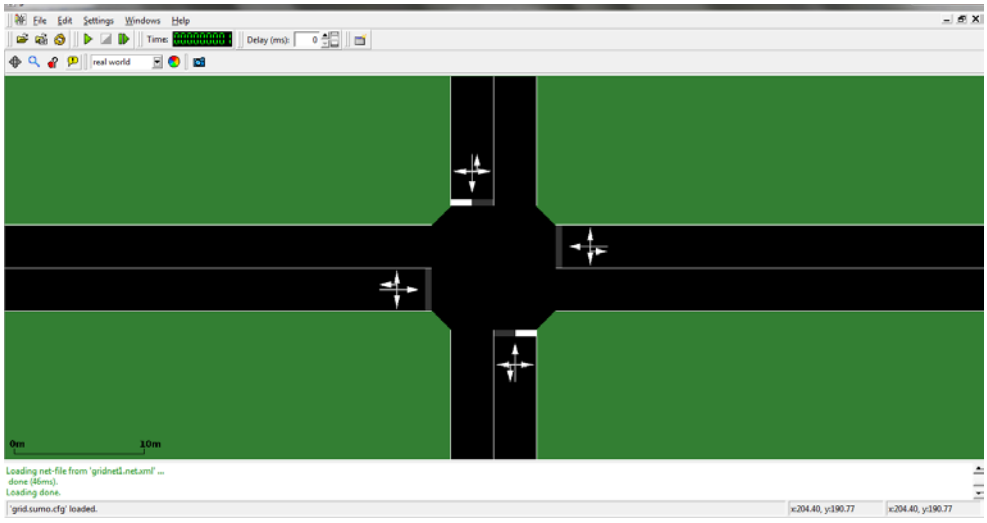


Figura 29 Detalle de intersección de red *grid*

B) En el segundo ejemplo generamos la misma red automática tipo *grid* donde las intersecciones son controladas por semáforos. El comando a introducir es el siguiente:

```
netgen --grid-net --grid-number=10 --grid-length=100 -j traffic_light
--output-file=gridnet1tl.net.xml
```

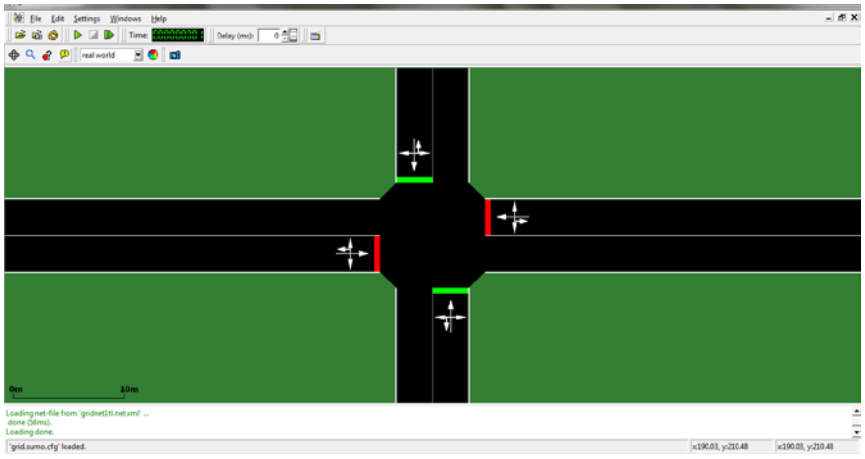


Figura 30 Detalle de intersección controlada por semáforos

C) Para el tercer ejemplo tomamos una red tipo *grid*, la cual contiene 6 cruces en el eje x y 4 cruces en el eje y separados 60 y 40 metros respectivamente unos de otros dentro del mismo eje. Para que todas las intersecciones controladas por semáforos contengan cuatro brazos de tráfico se utiliza el comando `attach-length` con un valor de 100 metros. El comando a introducir es el siguiente:

```
netgen --grid-net --grid-x-number=6 --grid-y-number=4 --grid-y-length=40 --grid-x-length=60 -j traffic_light --attach-length=100 --output-file=gridnet.net.xml
```

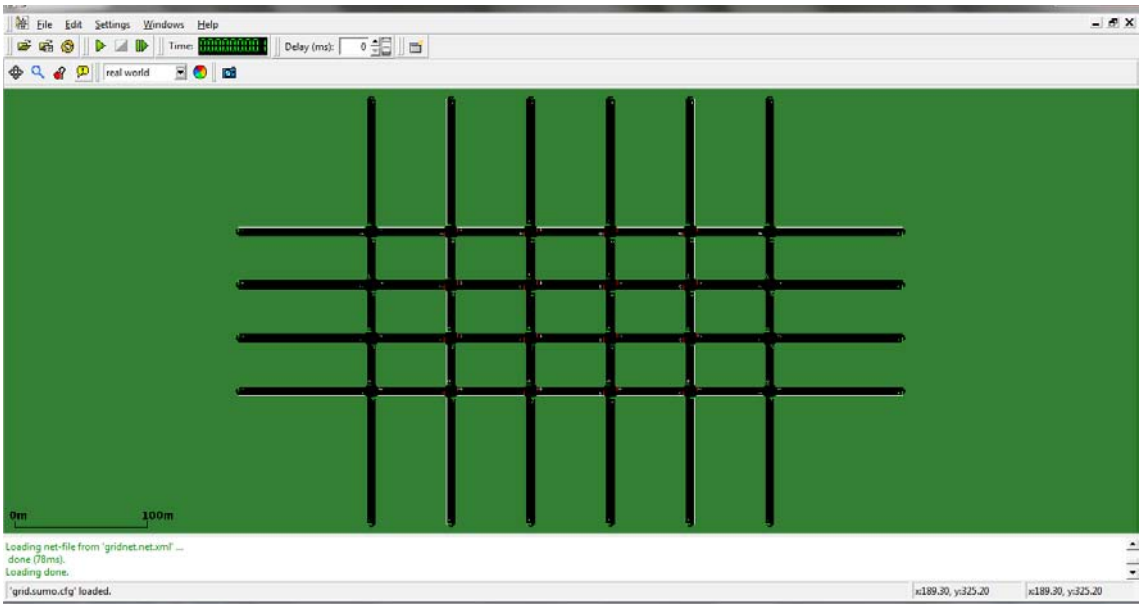


Figura 31 Red *grid* con ampliación en intersecciones

- **Redes tipo *spider***

Estas redes de generación automática se definen por el número de ejes o brazos que las dividen, siendo 13 por defecto, el número de círculos concéntricos (20 por defecto) que las forman, y la distancia entre dichos círculos en metros (100 por defecto). Los comandos asociados para su generación son los que siguen:

```
--spider-armnumber
--spidercircle-number o –circles
-spiderspace- rad o --radius
```

Es importante destacar que el punto central de la red de araña suele ser conflictivo al estar formado por numerosas vías, por lo que se toma como una unión tipo *right-of-way*. Si bien puede eliminarse el centro para obtener de este modo una red circular.

Veamos un par de ejemplos ilustrativo de las redes *spider*:

A) En este primer ejemplo de generación automática de redes *spider* hemos generado una red con un total de 5 brazos y de 5 círculos concéntricos donde la distancia entre dichos círculos es de 100 metros. El comando a introducir es el siguiente:

```
netgen --spider-net --spider-arm-number=5 --spider-circle-number=5 --
spider-space-rad=100 --output-file=spider.net.xml
```

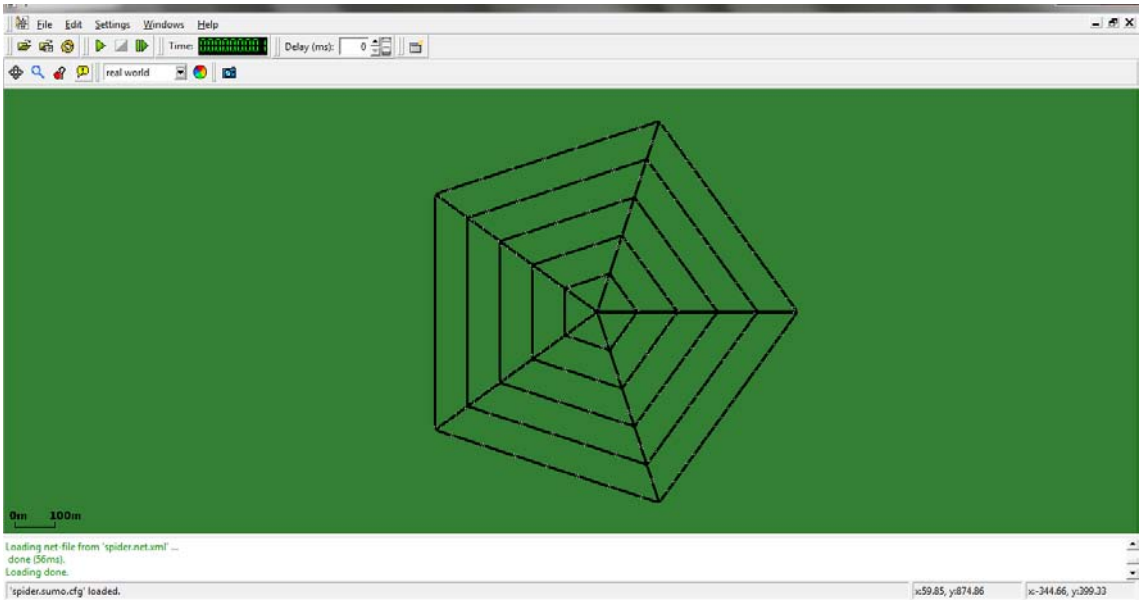


Figura 32 Red *Spider* de generación automática

B) Para el Segundo ejemplo generamos una red *spider* de las mismas características que la anterior, pero omitiendo el centro de la red para obtener un tráfico circular. El comando a introducir es el que sigue:

```
netgen --spider-net --spider-arm-number=5 --spider-circle-number=5 --
spider-space-rad=100--spider-omit-center --output-
file=spidercircular.net.xml
```

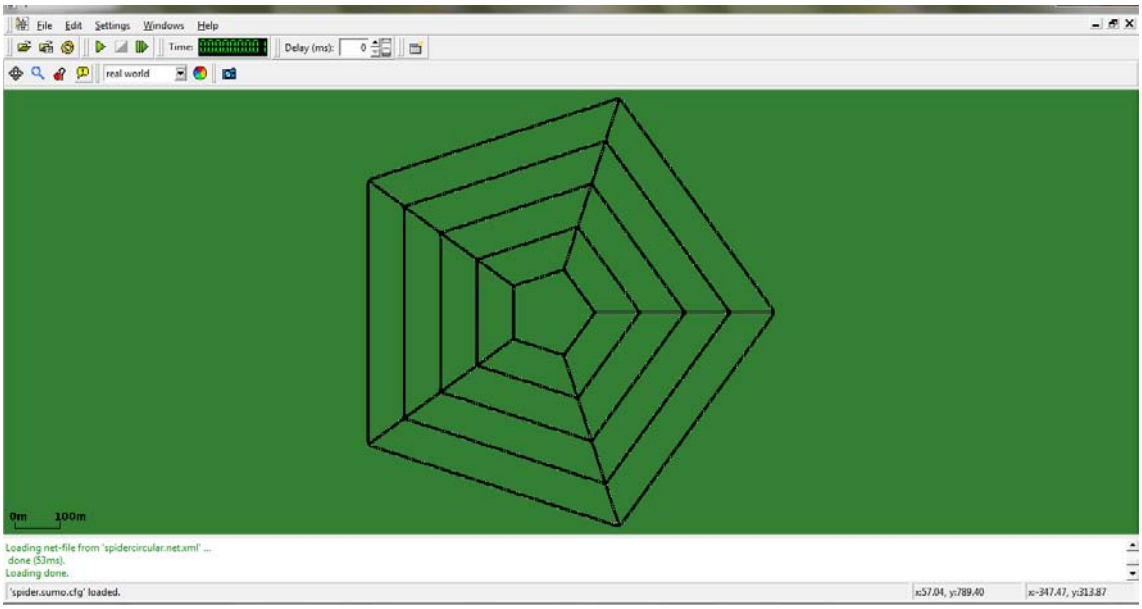


Figura 33 Red *spider* sin centro

- ## Redes aleatorias

Con esta opción se generan redes aleatorias y dependiendo de los parámetros establecidos se obtendrán redes con distintas características:

- `--rand-max-distance <FLOAT>`:
- `--rand-min-distance <FLOAT>`:
- `--rand-min-angle <FLOAT>`:
- `--rand-num-tries <FLOAT>`:
- `--rand-connectivity <FLOAT>`:
- `--rand-neighbor-dist1 <FLOAT>`:
- `--rand-neighbor-dist2 <FLOAT>`:
- `--rand-neighbor-dist3 <FLOAT>`:
- `--rand-neighbor-dist4 <FLOAT>`:
- `--rand-neighbor-dist5 <FLOAT>`:
- `--rand-neighbor-dist6 <FLOAT>`:

Veamos un ejemplo ilustrativo de las redes *random*, donde se ejecuta un total de 500 iteraciones incluyendo una distancia máxima de cada segmento de 150 metros. El comando a introducir es el siguiente:

```
netgen --random-net -o random.net.xml --rand-iterations=500 --rand-
max-distance=150 --rand-min-angle=1
```

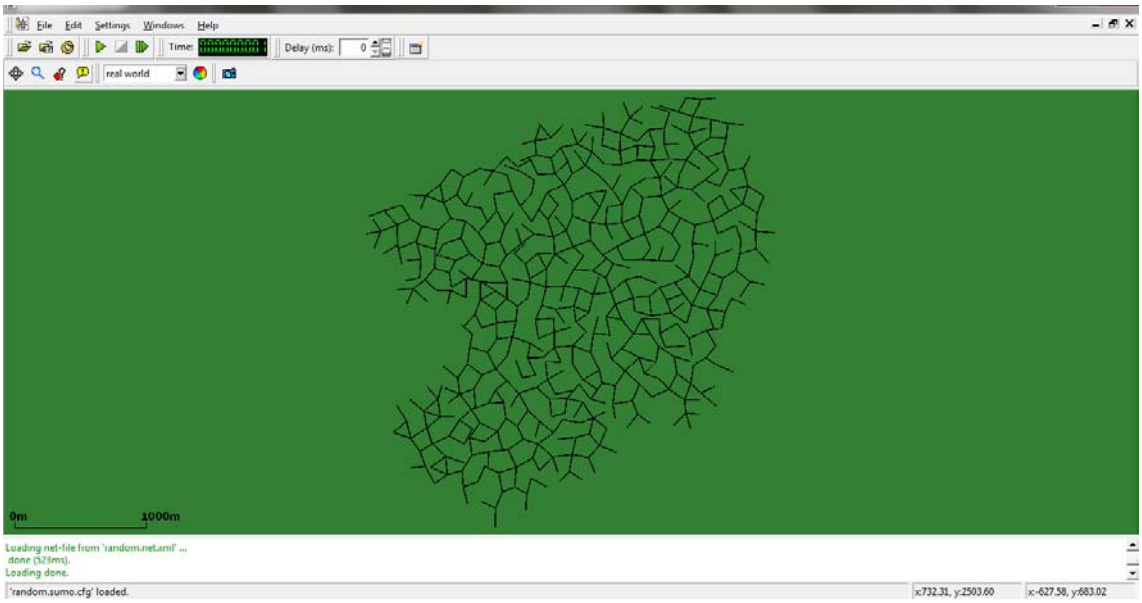


Figura 34 Red aleatoria

a.2) Generación xml: la segunda opción que nos proporciona SUMO a la hora de generar una red es la de que el usuario diseñe paso a paso la red donde se llevarán a cabo las simulaciones y para esto deberá seguir el siguiente esquema:

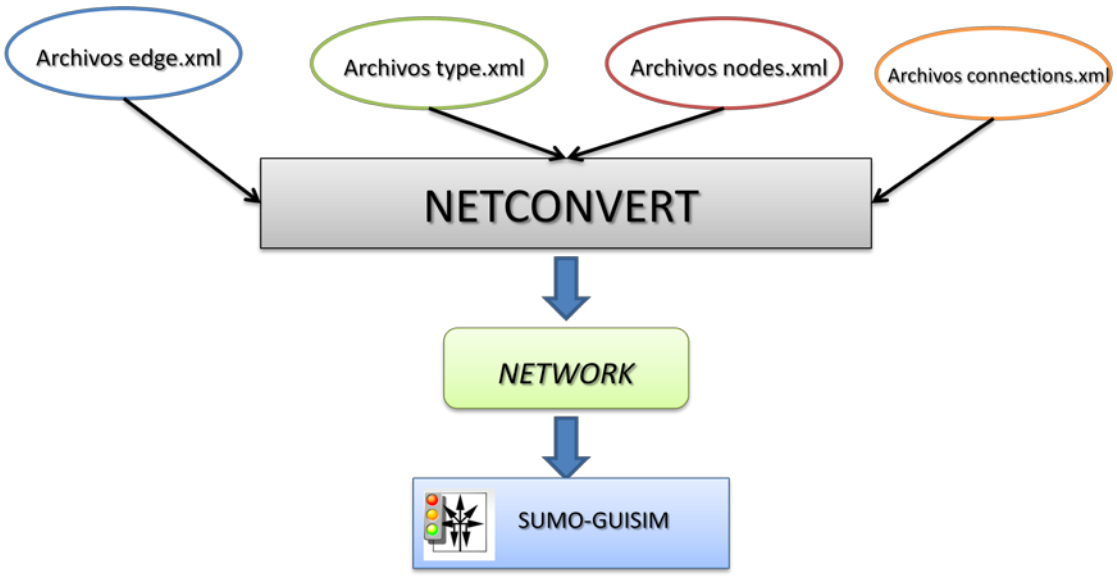


Figura 35 Esquema para la generación de redes manuales

Los conceptos a tener en cuenta antes de comenzar la descripción de nuestra red son los siguientes:

- **Nodo(*node*)**: representa las intersecciones y las uniones existentes en la red.
- ***Edge***: representa las carreteras y las calles de la red, es decir las vías de tráfico.
- **Tipo**: nos permite definir tipos de vías con unas propiedades determinadas como prioridades, número de carriles, velocidad máxima de la vía, etc.
- **Conexiones**: indica cómo se unen los diversos vértices para formar la red final.

Descripción de los nodos

El documento donde se incluirán las descripciones de los nodos tiene extensión `.nod.xml` y utilizaremos la siguiente sintaxis de comandos:

```
<node id="<STRING>" x="<FLOAT>" y="<FLOAT>" [type="<TYPE>"]/>
```

Donde como campos obligatorios tenemos:
• id: Identificador del nodo.
• x: Posición del nodo en el eje de coordenadas x dada en metros.
• y: Posición del nodo en el eje de coordenadas y dada en metros.

Mientras que el tipo será una característica opcional a incluir y se define como sigue:

• type. Permite definir el tipo adicional del nodo. Si se deja en blanco NETCONVERT asignará un tipo de manera aleatoria que puede que no sea el deseado. Los tipos disponibles son los siguientes:

o priority. Los vehículos deben esperar hasta que los vehículos de la derecha hayan atravesado la intersección.
o traffic_light. La unión de nodos se encuentra controlada por semáforos.

El sistema de coordenadas utilizadas por sumo es el siguiente:

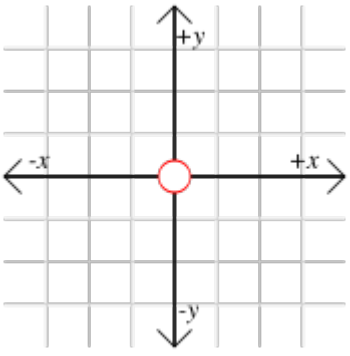


Figura 36 Sistemas de coordenadas para SUMO

De tal manera que este sistema se encuentra centrado en la pantalla del monitor correspondiéndose con las posiciones de los ejes x (posiciones horizontales) e y (posiciones verticales).Para ilustrar mejor la elaboración de las redes de manera manual, veremos paso a paso la creación de los archivos necesarios para construir el escenario que denominaremos red manual. El esquema de dicha red con sus nodos, vértices y conexiones correspondientes se muestra a continuación:

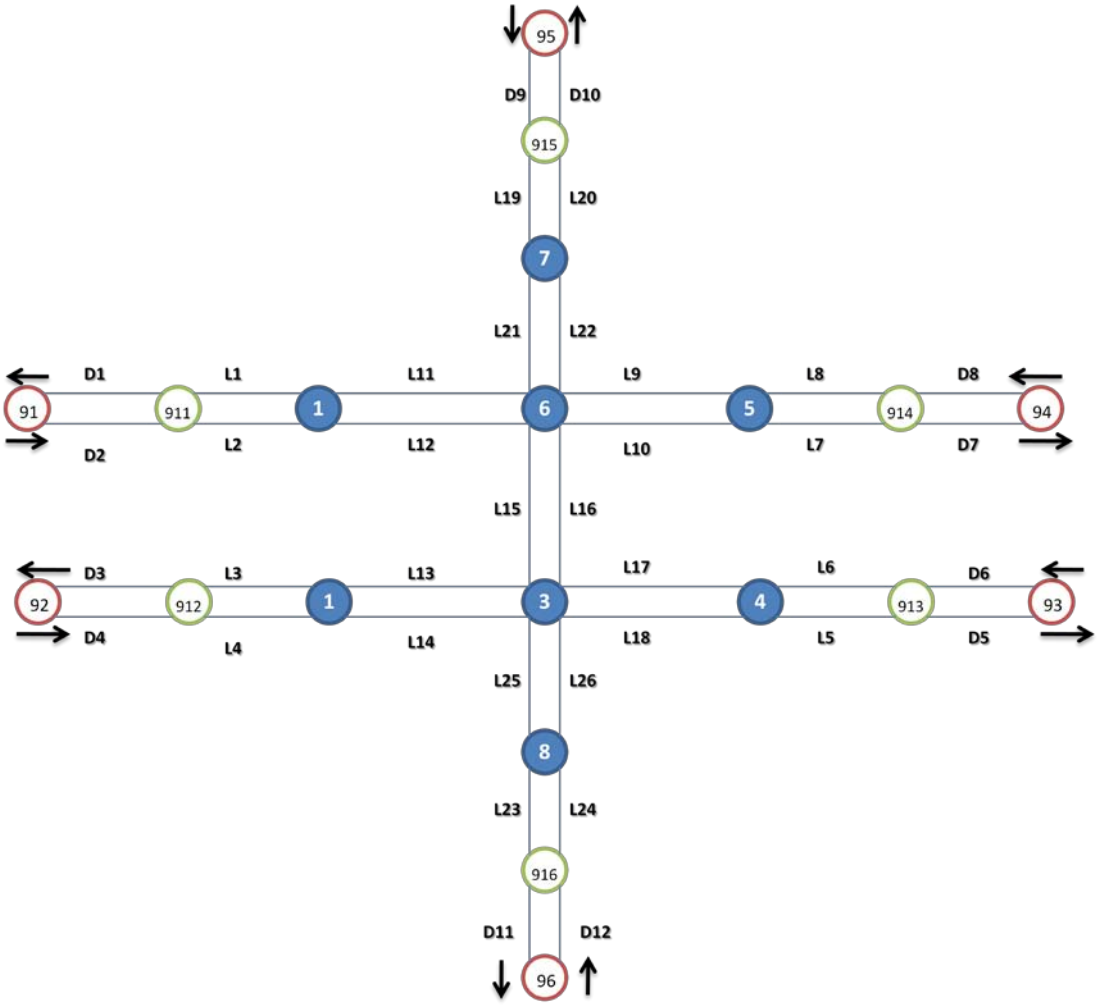


Figura 37 Esquema de red manual

El contenido del archivo *xml* que describe los nodos que forman nuestra red manual se muestra a continuación:

redmanual.nod.xml

```
<nodes>
  <node id="91" x="-1000.0" y="1000.0" />
  <node id="92" x="-1000.0" y="0.0" />
```

```
<node id="93" x="3000.0" y="0.0" />
<node id="94" x="+3000.0" y="+1000.0" />
<node id="95" x="+1000.0" y="+3000.0" />
<node id="96" x="+1000.0" y="-3000.0" />
<node id="911" x="-500.0" y="+1000.0" />
<node id="912" x="-500.0" y="0.0" />
<node id="913" x="+2500.0" y="0.0" />
<node id="914" x="2500.0" y="+1000.0" />
<node id="915" x="+1000.0" y="+2500.0" />
<node id="916" x="+1000.0" y="-2500.0" />
<node id="1" x="0.0" y="+1000.0" />
<node id="2" x="0.0" y="0.0" />
<node id="3" x="+1000.0" y="0.0" type="traffic_light" />
<node id="4" x="+2000.0" y="0.0" />
<node id="5" x="+2000.0" y="+1000.0" />
<node id="6" x="+1000.0" y="+1000.0" type="traffic_light" />
<node id="7" x="+1000.0" y="+2000.0" />
<node id="8" x="+1000.0" y="-1000.0" />
</nodes>
```

- **Descripción de los *edges***

Para los vértices la sintaxis utilizada tendrá dos vertientes, o bien definimos nuestro *edge* desde un nodo (*fromnode*) a otro (*tonode*) :

`<edge id="<STRING>"fromnode="<NODE_ID>" tonode="<NODE_ID>"/>.`

O simplemente indicamos las coordenadas de origen y destino correspondientes tanto del eje x como del eje y de origen (xfrom, yfrom) y de destino (xto, yto), de tal manera que en éstas se crearán los nodos correspondientes si no existían:

`<edge id="<STRING>" xfrom="<FLOAT>" yfrom="<FLOAT>" xto="<FLOAT>" yto="<FLOAT>") />.`

Es importante destacar que los *edges* son siempre unidireccionales e irán orientados de origen "*from*" a destino "*to*". Para definir otras propiedades de los *edges* tendremos dos alternativas bien utilizamos archivos de tipo, *.typ.xml* , cuya función es detallar el tipo de vía en el que nos

encontramos de tal manera que la opción *type* contendrá todos los datos relevantes para la definición de

`type="<STRING>"`

o bien los detallamos en el propio archivo *edg.xml* como sigue:

```
nolanes="<INT>" speed="<FLOAT>" priority="<UINT>" length="<FLOAT>"
shape="<2D_POINT>  <2D_POINT>]* spread_type="center"
```

Es importante notar que aunque se defina un archivo tipo *type*, se pueden sobrescribir siempre estos parámetros, sustituyendo cualquiera de los siguientes parámetros en la línea de comandos anteriormente expuesta: `nolanes`, `speed` and `priority`. En el caso de dejar estos parámetros sin definir los valores por defecto serán los siguientes: un solo carril, velocidad máxima permitida 13.9 m/s lo que corresponde con aproximadamente 50 km/h. La longitud del *edge* se calculará como la distancia entre un punto de inicio y un punto de fin.
El archivo tipo contendrá por tanto el siguiente comando con el siguiente formato:

`<type id="<STRING>" nolanes="<INT>" speed="<FLOAT>"priority="<UINT>"/>.`

Resumiendo los atributos principales para los *edges* son los siguientes:

- `id:` identificador del nodo mediante una cadena de caracteres.
- Para la definición del origen y destino tenemos dos alternativas:

Alternativa A)
- `fromnode:` indicando el nombre del nodo de donde nace el *edge*.
- `tonode:` indicando el nombre del nodo de donde finaliza el *edge*.

Alternativa B)
- `xfrom.` Se refiere a la coordenada x del nodo de donde el *edge* tiene que nacer, y debe ser un número decimal.
- `yfrom.` Se refiere a la coordenada y del nodo de donde el *edge* tiene que nacer, y debe ser un número decimal.
- `xto.` Se refiere a la coordenada x del nodo de donde el *edge* tiene que finalizar, y debe ser un número decimal.
- `yto.` Se refiere a la coordenada y del nodo de donde el *edge* tiene que finalizar, y debe ser un número decimal.

- Para la descripción de las características de los *edges* disponemos de dos opciones:

Opción A)
- `type:` indicando el nombre de un tipo de edge que fue definido previamente en un archivo .typ.xml

Opción B)
- `nolanes:` número de carriles, siempre siendo un valor entero.
- `speed:` máxima velocidad permitida en la vía (edge= definida con un número decimal expresado en m/s.
- `priority:` la prioridad del *edge*, dada por un entero.

Indicar que la prioridad juega un papel crucial durante el cómputo de las reglas para cada nodo. Normalmente la velocidad permitida en un *edge* así como el número de carriles se utilizan para calcular que elemento tiene una mayor prioridad. Con el comando priority se puede forzar la prioridad del nodo.

- `length:` la longitud en metros del *edge*, siendo éste un número decimal.
-

Sobre la forma de los *edges* para el comando `shape`, se indica una lista de posiciones a visitar por el *edge*, dadas por coordenadas x e y (metros). Pongamos el siguiente ejemplo:

```
<edge id="e1" fromnode="0" tonode="1" shape="0,0 100,0"/>
```

La interpretación es como sigue: *el edge con identificador e1 nace en el nodo 0, visitando la posición 0,0 para avanzar 100 hasta que finalmente alcana el nodo sumidero 1.*

- `spread_type:` la función de este comando es la de distribuir los carriles a ambos lados de la forma propia del *edge*. Es decir, si utilizamos la opción "*center*" los carriles se distribuyen a ambos lados de la forma, mientras que cualquier otro valor se interpreta como "*right*", es decir distribuyendo los carriles hacia la derecha.

- **Descripciones de las conexiones**

Las conexiones describen como se conectan específicamente los distintos carriles entre sí. Es decir cómo se conectan entre sí carriles de diferentes *edge*s a través de los distintos puntos de unión. Para continuar con la creación de nuestra red el siguiente paso será el de definir el archivo `type`, que nos dará los tipos de vías a los que haremos referencia en el archivo `.edg.xml` a continuación:

redmanual.typ.xml

```
<types>
  <type id="a" priority="3" nolanes="3" speed="13.889" />
  <type id="b" priority="3" nolanes="2" speed="13.889" />
  <type id="c" priority="2" nolanes="3" speed="13.889" />
  <type id="d" priority="1" nolanes="2" speed="19.444" />
    </types>
```

redmanual.edg.xml

```
<edges>
  <edge id="D1" fromnode="911" tonode="91" type="a" />
  <edge id="D2" fromnode="91" tonode="911" type="b" />
```

```
<edge id="D3" fromnode="912" tonode="92" type="a" />
<edge id="D4" fromnode="92" tonode="912" type="b" />
<edge id="D5" fromnode="913" tonode="93" type="a" />
<edge id="D6" fromnode="93" tonode="913" type="b" />
<edge id="D7" fromnode="914" tonode="94" type="a" />
<edge id="D8" fromnode="94" tonode="914" type="b" />
<edge id="D9" fromnode="95" tonode="915" type="b" />
<edge id="D10" fromnode="915" tonode="95" type="a" />
<edge id="D12" fromnode="96" tonode="916" type="d" />
<edge id="D11" fromnode="916" tonode="96" type="a" />
<edge id="L1" fromnode="1" tonode="911" type="a" />
<edge id="L2" fromnode="911" tonode="1" type="b" />
<edge id="L3" fromnode="2" tonode="912" type="a" />
<edge id="L4" fromnode="912" tonode="2" type="b" />
<edge id="L5" fromnode="4" tonode="913" type="a" />
<edge id="L6" fromnode="913" tonode="4" type="b" />
<edge id="L7" fromnode="5" tonode="914" type="a" />
<edge id="L8" fromnode="914" tonode="5" type="b" />
<edge id="L9" fromnode="5" tonode="6" type="a" />
<edge id="L10" fromnode="6" tonode="5" type="a" />
<edge id="L11" fromnode="6" tonode="1" type="a" />
<edge id="L12" fromnode="1" tonode="6" type="a" />
<edge id="L13" fromnode="3" tonode="2" type="a" />
<edge id="L14" fromnode="2" tonode="3" type="a" />
<edge id="L15" fromnode="6" tonode="3" type="c" />
<edge id="L16" fromnode="3" tonode="6" type="c" />
<edge id="L17" fromnode="4" tonode="3" type="a" />
<edge id="L18" fromnode="3" tonode="4" type="a" />
<edge id="L19" fromnode="915" tonode="7" type="b" />
<edge id="L20" fromnode="7" tonode="915" type="a" />
```

```
    <edge id="L21" fromnode="7" tonode="6" type="a" />
    <edge id="L22" fromnode="6" tonode="7" type="a" />
    <edge id="L23" fromnode="8" tonode="916" type="a" />
    <edge id="L24" fromnode="916" tonode="8" type="b" />
    <edge id="L25" fromnode="3" tonode="8" type="a" />
    <edge id="L26" fromnode="8" tonode="3" type="a" />
    </edges>
```

Para especificar los movimientos de tráfico y las conexiones de los carriles se requiere un archivo como el que sigue, si bien por defecto el carril situado más a la derecha de cada Link, estará alineado con el carril de más da la derecha del link correlativo.
Analicemos pues el archivo `redmanual.con.xml`:

redmanual.con.xml

```
<connections>
  <connection from="L2" to="L12" lane="0:0" />
  <connection from="L2" to="L12" lane="0:1" />
  <connection from="L2" to="L12" lane="1:2" />
  <connection from="L4" to="L14" lane="0:0" />
  <connection from="L4" to="L14" lane="1:1" />
  <connection from="L4" to="L14" lane="1:2" />
  <connection from="L19" to="L21" lane="0:0" />
  <connection from="L19" to="L21" lane="0:1" />
  <connection from="L19" to="L21" lane="1:2" />
  <connection from="L24" to="L26" lane="0:0" />
  <connection from="L24" to="L26" lane="0:1" />
  <connection from="L24" to="L26" lane="1:2" />
  <connection from="L9" to="L11" lane="0:0" />
  <connection from="L9" to="L11" lane="1:1" />
  <connection from="L9" to="L11" lane="1:2" />
  <connection from="L9" to="L15" lane="1:2" />
  <connection from="L9" to="L15" lane="2:2" />
```

```
<connection from="L9" to="L22" lane="0:0" />
<connection from="L9" to="L22" lane="1:2" />
<connection from="L16" to="L10" lane="0:0" />
<connection from="L16" to="L10" lane="1:1" />
<connection from="L16" to="L10" lane="1:2" />
<connection from="L16" to="L11" lane="2:2" />
<connection from="L12" to="L15" lane="0:0" />
<connection from="L12" to="L15" lane="1:1" />
<connection from="L12" to="L10" lane="1:0" />
<connection from="L12" to="L10" lane="1:1" />
<connection from="L12" to="L10" lane="2:2" />
<connection from="L14" to="L16" lane="1:1" />
<connection from="L14" to="L16" lane="1:0" />
<connection from="L14" to="L16" lane="2:2" />
<connection from="L14" to="L18" lane="0:0" />
<connection from="L14" to="L18" lane="1:1" />
<connection from="L14" to="L18" lane="1:2" />
<connection from="L17" to="L16" lane="0:0" />
<connection from="L17" to="L16" lane="1:1" />
<connection from="L17" to="L16" lane="1:2" />
<connection from="L17" to="L13" lane="1:0" />
<connection from="L17" to="L13" lane="1:1" />
<connection from="L17" to="L13" lane="2:2" />
<connection from="L17" to="L25" lane="1:2" />
<connection from="L17" to="L25" lane="2:2" />
</connections>
```

Una vez hemos definido todos los componentes necesarios para la creación de un red vial pasamos a su agrupación en un archivo de configuración, de tal manera que lo procesaremos mediante NETCONVERT para generar un archivo final con extensión `.net.xml` para finalmente alimentar a SUMO en la generación de la simulación de tráfico microscópico.

Finalmente nuestra red quedará de la siguiente manera:

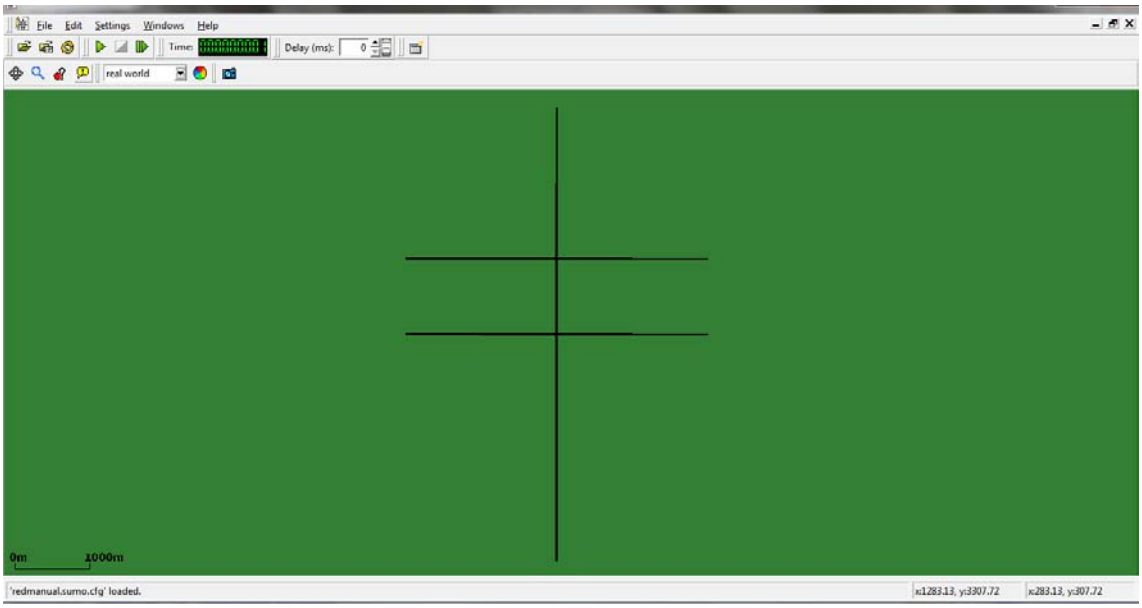


Figura 38 Visualización de la red en GUI

Ampliando para ver la zona de intersección, controlada por semáforos tal y como indicamos para los nodos 6 y 3:

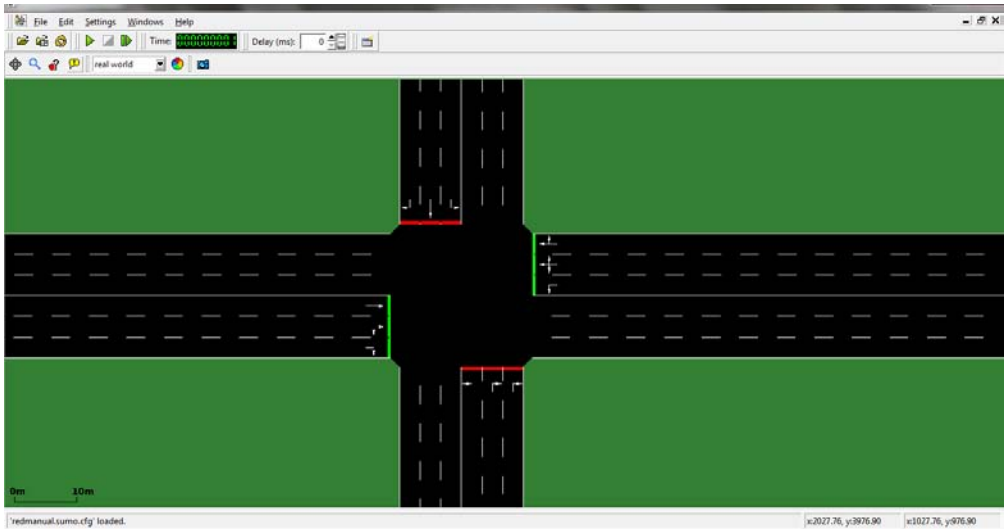


Figura 39 Detalle de intersección con semáforos para redmanual.net

Las incorporaciones ó paso de 3 carriles a 2 que se han simulado son las siguientes en detalle ampliadas:

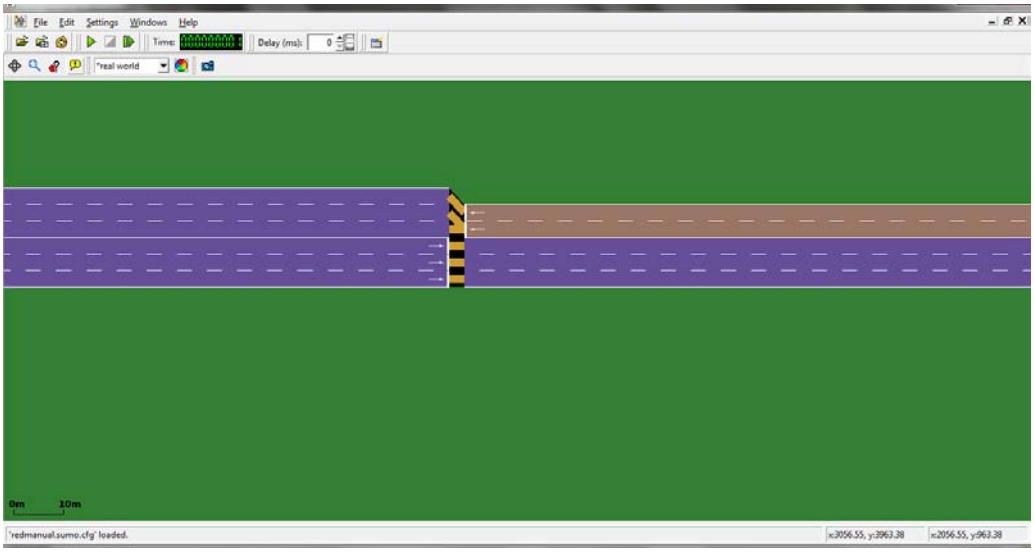


Figura 40 Detalle de estrechamiento de carriles

a.3) Generación mediante importación

Hacer uso de las fuentes topográficas disponibles a los usuarios es muy útil siempre y cuando se sepa tratar la información para los objetivos requeridos. Como indicamos en la primera parte de este apartado, SUMO es capaz de importar mapas de diversas fuentes si bien nosotros nos centraremos en *OpenStreetMap.* Esta base de datos contiene mapas de distribución gratuita de todo el mundo y no sólo se pueden visualizar mapas, sino que también se pueden importar y editar como veremos seguidamente. Una vez tengamos nuestro mapa con un nivel de detalle deseado pasaremos a utilizar la herramienta NETCONVERT, es decir el traductor de SUMO para entender fuentes de datos distintas a el mismo.

OpenStreetmap: el mundo en un *click*

La motivación para el uso de esta base de datos se basa en primer lugar en que resultó muy amigable tanto en lo que refiere a su interfaz como al método de uso, siendo también innovador para la nueva versión de sumo 0.12.0. Pero lo que la hace más interesante si cabe, es que se trata de una herramienta totalmente *Opensource*, sin problemas de licencias legales. Pongamos como ejemplo el mapa de la zona universitaria de Antigones en Cartagena:

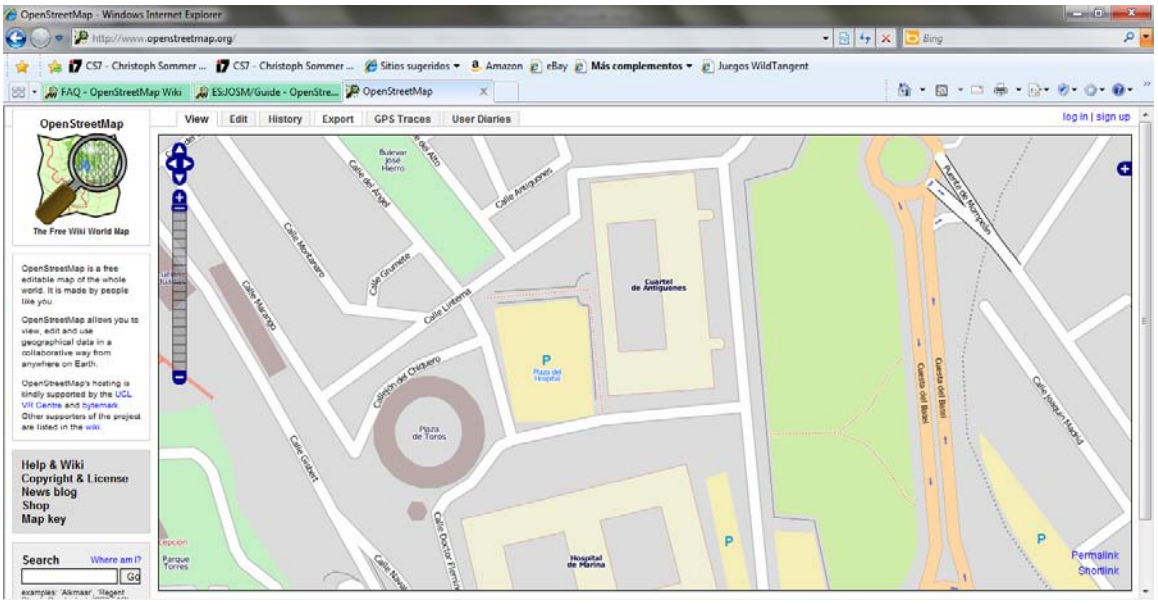


Figura 41 Zona de Antigones con *OpenStreetMap*

Haciendo click en la pestaña de Exportar, una vez hayamos ampliado mediante zoom la zona deseada nos encontramos con el siguiente menú el cual nos permitirá la exportación de los datos topográficos en distintos formatos, en nuestra caso para que exista compatibilidad con SUMO debemos de seleccionar la opción de Datos formato *OpenStreetMap XML* lo que resultará en un mapa en formato **.osm** que luego podremos convertir con nuestro paquete de simulación. En nuestro caso lo guardaremos bajo el nombre: `universidad.osm`.

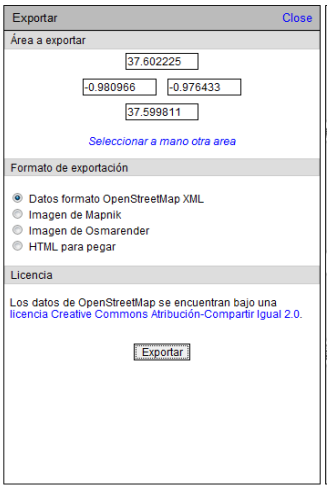


Figura 42 Menú para exportación de mapas

Veamos cómo se visualizaría en SUMO nuestro mapa:

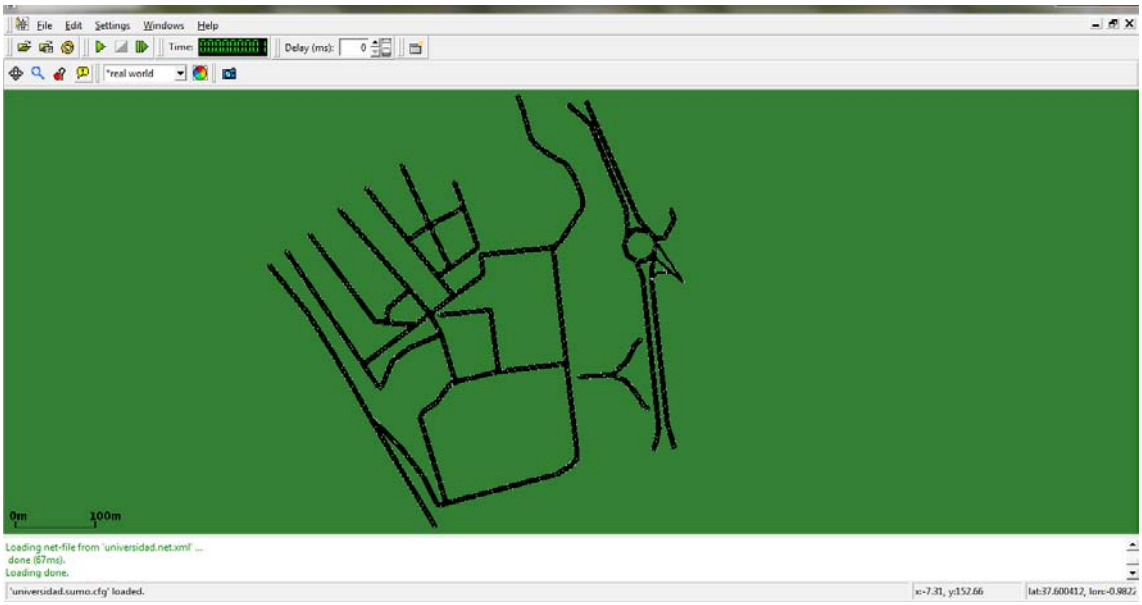


Figura 43 Mapa Antigones en SUMO-GUI

El cual o bien nos sirve perfectamente para alcanzar nuestros objetivos de simulación o bien decidimos retocarlo, para simplificar la zona de estudio, para ello haremos uso del programa en java JOSM (68).

Con el programa JOSM se podrá modificar el mapa importado y una vez modificado según nuestras preferencias podremos cargarlo haciendo uso de NETCONVERT que nos dará el formato adecuado de nuestro mapa para llevar a cabo las simulaciones.

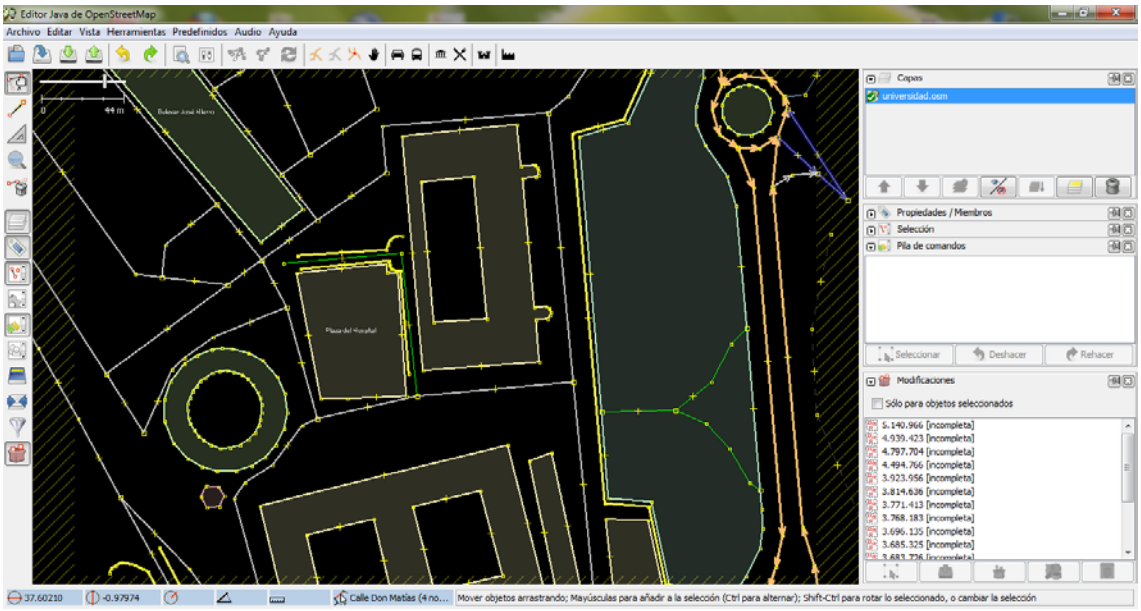


Figura 44 Vista de mapa Antigones en JOSM

Una vez tengamos nuestro mapa con las características topográfica deseadas podremos seguir trabajando en SUMO introduciendo en la línea de comandos lo siguiente;

```
netconvert --osm universidad.osm.xml -o universidad.net.xml
```

y por tanto consiguiendo una red entendible para nuestro simulador de tráfico.

(Nota: acerca de las coordenadas, los datos de entrada se encuentran en coordenadas WGS84 y SUMO se encargará de transformarlas a UTM de manera automática.)

IV.III.II Generación de rutas

El siguiente paso en nuestra simulación es la de generar rutas, es decir la demanda de tráfico para que los vehículos se desplacen por la red que hemos creado y para esto disponemos de distintos mecanismos. Es importante conocer los términos por los cuales definiremos los movimientos vehiculares dentro de la red y son principalmente:

- *Trip* (viaje): movimiento del vehículo definido por un origen de partida, o *edge* y un punto de llegada, otro *edge* junto con el tiempo de simulación de salida.
- *Route* (Ruta): se basa en un itinerario, recorrido, viaje o *trip* en el que además se definen explícitamente los distintos *edges* que atravesará nuestro vehículo.

Para generar los movimientos vehiculares disponemos de los siguientes mecanismos:

a) Diseño de viajes.
b) Diseño de flujos (*flow*), del miso modo que los *trips* se define un *edge* de origen y otro de destino si bien se pueden unir diversos vehículos que tengan el mismo origen y el mismo destino.
c) Diseño de flujos junto con porcentaje (o probabilidad) de giro en intersecciones.
   En este caso se omiten el *edge* destino y se utilizan porcentaje de giro en las intersecciones.
d) Utilizando rutas aleatorias.
   Mediante un archivo *python* diseñado para tal efecto, si bien tiene que ver poco con la realidad.
e) Importando rutas existentes de la realidad.
f) A mano.
   Es decir definiendo “manualmente” con *xml* desde el inicio, los archivos `.rou.xml`

Tipos de vehículos

Antes de comenzar a definir las rutas tenemos que aprender a definir los tipos de vehículos que podremos simular en nuestro software. La sintaxis para definir un tipo vehículo es la que sigue:

```
<vtype id="type1" accel="0.8" decel="4.5" sigma="0.5" length="7.5" maxspeed="70"/>
```

En este caso se han elegido los valores más típicos para el modelo de *car-following* de Stefan Krauss, que es el utilizado por SUMO como se indicó al comienzo del capítulo. Si bien actualmente se encuentran en desarrollo otros mecanismos como el IDM (69), pero al estar en

versión beta nos centraremos en el modelo ya consolidado de Krauss y los parámetros correspondientes a éste. En la definición de un tipo de vehículo se tienen en cuenta parámetros físicos y comportamientos mentales de los conductores, y los principales son los siguientes:

- id: cadena de caracteres para definir el identificador del vehículo.
- type: tipo de vehículo a usar para el mismo.
- accel: capacidad de aceleración para los vehículos de este tipo (en m/s^2).
- decel: capacidad de deceleración para los vehículos de este tipo (en m/s^2).
- sigma: grado de imperfección del conductor desde 0 hasta 1. Donde el valor unitario representa el comportamiento de un conductor perfecto, hasta 0 con el que se modela un conductor totalmente distraído.
- tau: tiempo de reacción del conductor en segundos.
- length: longitud del vehículo en metros.
- maxspeed: máxima velocidad del vehículo (en m/s).
- depart: tiempo de la simulación en el que el vehículo se genera para ser introducido en la red.
- color: parámetro definitorio del color del vehículo, dado por tres valores entre 0 y 1 para rojo verde y azul, separados por coma y sin espacios en blanco entre sí.

Clases de vehículos abstractos

Los vehículos en SUMO pueden ser asignados como vehículos de clase abstracta, definida haciendo uso de una clase *vclass* abstracta. Este tipo de clases se utiliza en determinados carriles para permitir o no la entrada de determinados vehículos mediante los comandos presentados anteriormente, `allow o disallow`.

Las clases existentes son las siguientes:

- "private"
- "public_transport"
- "public_emergency"
- "public_authority"
- "public_army"
- "vip"
- "ignoring"
- "passenger"
- "hov"
- "taxi"
- "bus"
- "delivery"
- "transport"
- "lightrail"
- "cityrail"
- "rail_slow"
- "rail_fast"
- "motorcycle"
- "bicycle"
- "pedestrian"

Un ejemplo ilustrativo de lo comentado hasta el momento sobre vehículos y rutas:

```
<routes>
<vtype id="type1" accel="0.8" decel="4.5" sigma="0.5" length="5"
maxspeed="70"/>
<vehicle id="0" type="type1" depart="0" color="1,0,0">
<route edges="beg middle end rend"/>
</vehicle>
</routes>
```

En este caso se genera un tipo de vehículo type1 y con las determinadas constantes de aceleración, deceleración, sigma, longitud del vehículo y velocidad máxima. A continuación se genera un vehículo de tipo uno, con salida en el tiempo 0 segundos y color rojo, siguiendo una ruta dada por el nombre de los *edges*. Una vez alcance el *edge* con nombre *rend* se eliminará de la simulación. Para definir varios vehículos que utilizan la misma ruta haremos lo siguiente: definir una ruta en el siguiente ejemplo *ruta 0*, que nos servirá para ser reutilizada por los diversos vehículos, en este caso los vehículos *0* y *1*.

```
<routes>
<vtype id="type1" accel="0.8" decel="4.5" sigma="0.5" length="5"
maxspeed="70"/>
<route id="route0" color="1,1,0" edges="beg middle end rend"/>
<vehicle id="0" type="type1" route="route0" depart="0" color="1,0,0"/>
<vehicle id="1" type="type1" route="route0" depart="0" color="0,1,0"/>
</routes>
```

Rutas y distribuciones vehiculares

Otra opción es dejar que sumo elija tanto la ruta como el tipo vehicular durante el tiempo de ejecución dadas unas determinadas distribuciones de probabilidad, como sigue en el ejemplo:

```
<routes>
<vtypeDistribution id="typedist1">
<vtype id="type1" accel="0.8" length="5" maxspeed="70"
probability="0.9"/>
<vtype id="type2" accel="1.8" length="15" maxspeed="50"
probability="0.1"/>
</vtypeDistribution>
</routes>
<routes>
<routeDistribution id="routedist1">
<route id="route0" color="1,1,0" edges="beg middle end rend"
probability="0.9"/>
<route id="route1" color="1,2,0" edges="beg middle end"
probability="0.1"/>
</routeDistribution>
</routes>
```

Donde la suma de las probabilidades tanto para el tipo de vehículo como par a las rutas a elegir debe de ser igual a 1.

Definiciones de viajes (*trip*)

La sintaxis básica para la definición de un viaje es como sigue:

```
<tripdef id="<ID>" depart="<TIME>" from="<ORIGIN_EDGE_ID>"
to="<DESTINATION_EDGE_ID>" [type="<VEHICLE_TYPE>"] [period="<INT>"
repno="<INT>"] [color="<COLOR>"]/>.
```

El parámetro `period`, nuevo para nosotros, nos indicará cada que cierta frecuencia (n segundos) un vehículo será emitido haciendo uso de la ruta definida en este comando. El parámetro `repno` nos permite indicar que cantidad de vehículos son emitidos cada vez. Este archivo se introduce en la parte del simulador DUAROUTER con el siguiente comando en línea: `"--trip-defs"` or `"-t"`, como sigue, y nos dará un archivo de salida del tipo `.rou.xml` que podrá utilizarse como archivo de rutas para llevar a cabo la simulación deseada:

```
duarouter --trip-defs=<TRIP_DEFS> --net=<SUMO_NET> \
   --output-file=MySUMORoutes.rou.xml -b <UINT> -e <UINT>
```

Definiciones de flujo (flow)

Para el caso del flujo lo que tenemos es:

```
<flow  id="<ID>"  from="<ORIGIN_EDGE_ID>"  to="<DESTINATION_EDGE_ID>"
begin="<INTERVAL_BEGIN>" end="<INTERVAL_END>" no="<VEHICLES_TO_EMIT>"
[type="<VEHICLE_TYPE>"] [color="<COLOR>"]/>.
```

Es importante destacar que en este caso el tiempo de salida no viene dado como tal ya que una serie de vehículos serán creados en función del valor que se le dé al parámetro “no”. Los tiempos de salida se verán distribuidos uniformemente dentro del intervalo dado por `begin-end`, siendo siempre números enteros. Existen dos posibilidades a la hora de definir los flujos que son las que siguen:

```
<flow id="0" from="edge0" to="edge1" begin="0" end="3600" no="100"/>
<interval begin="0" end="3600">
<flow id="0" from="edge0" to="edge1" no="100"/>
</interval>
```

En este caso tendremos que utilizar la siguiente línea de comandos para que DUAROUTER realice la traducción necesaria al crear el `archiv .rou.xml`:

```
duarouter --flows=<FLOW_DEFS> --net=<SUMO_NET> \
--output-file=MySUMORoutes.rou.xml -b <UINT> -e <UINT>
```

Tomando el siguiente escenario hagamos una comparativa entre el comando `trips` y el comando `flows`:

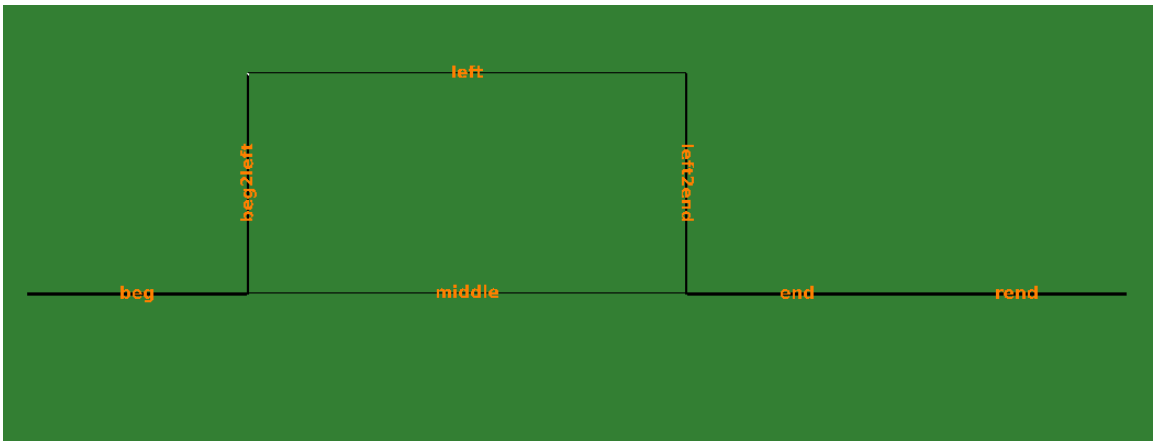


Figura 45 Escenario para generar *trips* vs. *flows*

Viajes vs. *flows*

En ambos casos el código genera el mismo archivo .rou.xml que alimentará a sumo.cfg:

- Para *flows* tenemos:

```
<flowdefs>

  <flow id="0" from="beg" to="rend" begin="0" end="1" no="20" />

    </flowdefs>
```

- En el caso de *trips*:

```
<tripdefs>
  <tripdef id="0" depart="0" from="beg" to="rend" />
  <tripdef id="1" depart="0" from="beg" to="rend" />
  <tripdef id="2" depart="0" from="beg" to="rend" />
  <tripdef id="3" depart="0" from="beg" to="rend" />
  <tripdef id="4" depart="0" from="beg" to="rend" />
  <tripdef id="5" depart="0" from="beg" to="rend" />
  <tripdef id="6" depart="0" from="beg" to="rend" />
  <tripdef id="7" depart="0" from="beg" to="rend" />
  <tripdef id="8" depart="0" from="beg" to="rend" />
  <tripdef id="9" depart="0" from="beg" to="rend" />
  <tripdef id="10" depart="0" from="beg" to="rend" />
  <tripdef id="11" depart="0" from="beg" to="rend" />
  <tripdef id="12" depart="0" from="beg" to="rend" />
  <tripdef id="13" depart="0" from="beg" to="rend" />
  <tripdef id="14" depart="0" from="beg" to="rend" />
  <tripdef id="15" depart="0" from="beg" to="rend" />
  <tripdef id="16" depart="0" from="beg" to="rend" />
  <tripdef id="17" depart="0" from="beg" to="rend" />
```

```
<tripdef id="18" depart="0" from="beg" to="rend" />
<tripdef id="19" depart="0" from="beg" to="rend" />
</tripdefs>
```

Obteniendo en ambos casos un total de 20 coches generados en toda la simulación:

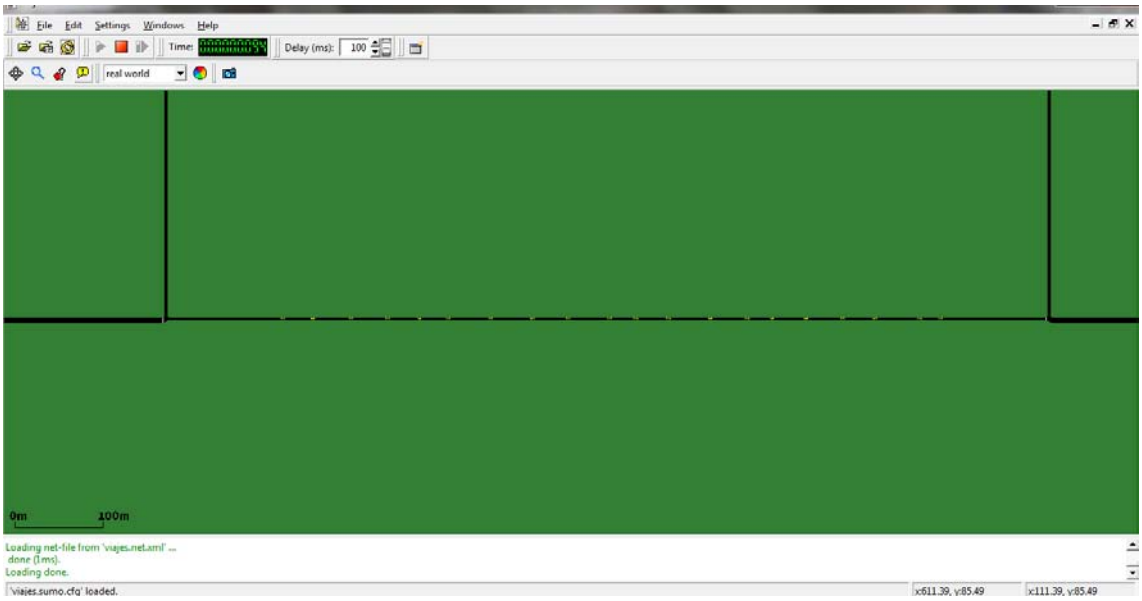


Figura 46 Momento de la simulación con los 10 vehículos en la red

b.1) Generación de rutas aleatorias

El archivo de python *randomTrips.py*, genera una serie de viajes aleatorios para una red dada, si bien debemos afirmar que a veces genera viajes que **no son del todo realistas.**

Pongamos el siguiente ejemplo:

```
python ..\tools\trip\randomTrips.py -n viajes.net.xml --length -p 2 -b
10 --end=100 –lanes
```

Donde los posibles parámetros para definir los viajes generados son los siguientes:

- -n incluiremos la red donde queremos generar los viajes.
- --length la elección de la fuente y el destino del viaje puede ser de manera aleatoria o ponderada en función de la longitud de los *edges* o del número de carriles.
- El intervalo de creación de los viajes vienen definidos entre dos parámetros –b (begin) y –e (end) que por defecto es igual a 3600 segundos.
- Mediante –o especificaremos en que archivo escribiremos los viajes generados

b.2) Porcentaje de giro en las uniones

La aplicación JTRROUTER utiliza flujos y porcentajes de giro en las uniones como entrada.

Los parámetros necesarios para hacer uso de esta opción son los siguientes:

- La red a través de la cual se enrutan los vehículos.
- La descripción de los porcentajes de giro en las uniones mediante el archivo con extensión *.turns.xml*.
- La descripción de los *flows*.

La sintaxis del comando en línea es como sigue:

```
jtrrouter --flows=<FLOW_DEFS> --turns=<TURN_DEFINITIONS> --
net=<SUMO_NET> \ --output-file=MySUMORoutes.rou.xml -b <UINT> -e
<UINT>
```

Para la creación del archivo con extensión `.turns.xml` se necesita incluir para cada intervalo y para cada pareja origen destino de los *edge* una problabilidad que en conjunto sumará 1. Veamos un ejemplo a continuación:

```
<turn-defs>
<interval begin="0" end="3600">
<fromedge id="myEdge0">
<toedge id="myEdge1" probability="0.2"/>
<toedge id="myEdge2" probability="0.7"/>
<toedge id="myEdge3" probability="0.1"/>
</fromedge>
... any other edges ...
</interval>
... some further intervals ...
</turn-defs>
```

En este caso la principal diferencia a la hora de definir los *flows* es que no será necesario incluir un nodo destino ya que éste se calcula de manera aleatoria según va siguiendo los giros definidos.

b.3) A mano

La última forma que presentaremos en esta sección para definir las posibles rutas es la de generar un ruta "manualmente", o desde cero mediante el lenguaje *xml*. Para nuestra red denominada *redmanual* nos quedaría el siguiente código que define las rutas a seguir por nuestros vehículos:

redmanual.rou.xml

```
<routes>
 <vtype accel="3.0" decel="6.0" id="CarA" length="5.0" maxspeed="50.0" sigma="0.5" />
 <vtype accel="2.0" decel="6.0" id="CarB" length="7.5" maxspeed="50.0" sigma="0.5" />
 <vtype accel="1.0" decel="5.0" id="CarC" length="5.0" maxspeed="40.0" sigma="0.5" />
```

```
<vtype accel="1.0" decel="5.0" id="CarD" length="7.5" maxspeed="30.0" sigma="0.5" />
<route id="route01" edges="D2 L2 L12 L10 L7 D7" />
<route id="route02" edges="D2 L2 L12 L15 L18 L5 D5" />
<route id="route03" edges="D2 L2 L12 L15 L13 L3 D3" />
<route id="route04" edges="D4 L4 L14 L18 L5 D5" />
<route id="route05" edges="D4 L4 L14 L16 L10 L7 D7" />
<route id="route06" edges="D4 L4 L14 L16 L11 L1 D1" />
<route id="route07" edges="D6 L6 L17 L13 L3 D3" />
<route id="route08" edges="D6 L6 L17 L16 L11 L1 D1" />
<route id="route09" edges="D6 L6 L17 L16 L10 L7 D7" />
<route id="route10" edges="D8 L8 L9 L11 L1 D1" />
 <route id="route11" edges="D8 L8 L9 L15 L13 L3 D3" />
 <route id="route12" edges="D8 L8 L9 L15 L18 L5 D5" />
 <vehicle depart="10" id="veh0" route="route01" type="CarA" color="1,0,0" />
 <vehicle depart="15" id="veh1" route="route02" type="CarA" color="1,0,0" />
 <vehicle depart="20" id="veh2" route="route03" type="CarA" color="1,0,0" />
 <vehicle depart="25" id="veh3" route="route04" type="CarA" color="1,0,0" />
 <vehicle depart="30" id="veh4" route="route05" type="CarA" color="1,0,0" />
 <vehicle depart="35" id="veh5" route="route06" type="CarA" color="1,0,0" />
 <vehicle depart="40" id="veh6" route="route07" type="CarA" color="1,0,0" />
 <vehicle depart="45" id="veh7" route="route08" type="CarA" color="1,0,0" />
 <vehicle depart="50" id="veh8" route="route09" type="CarA" color="1,0,0" />
 <vehicle depart="55" id="veh9" route="route10" type="CarA" color="1,0,0" />
 <vehicle depart="60" id="veh10" route="route11" type="CarA" color="1,0,0" />
 <vehicle depart="65" id="veh11" route="route12" type="CarA" color="1,0,0" />
 <vehicle depart="70" id="veh12" route="route01" type="CarB" color="0,1,0" />
 <vehicle depart="75" id="veh13" route="route02" type="CarB" color="0,1,0" />
 <vehicle depart="80" id="veh14" route="route03" type="CarB" color="0,1,0" />
 <vehicle depart="85" id="veh15" route="route04" type="CarB" color="0,1,0" />
 <vehicle depart="90" id="veh16" route="route05" type="CarB" color="0,1,0" />
```

```
  <vehicle depart="95" id="veh17" route="route06" type="CarB" color="0,1,0" />
  <vehicle depart="100" id="veh18" route="route07" type="CarC" color="0,1,0" />
  <vehicle depart="105" id="veh19" route="route08" type="CarC" color="1,0,0" />
  <vehicle depart="110" id="veh20" route="route09" type="CarC" color="1,0,0" />
  <vehicle depart="115" id="veh21" route="route10" type="CarC" color="1,0,0" />
  <vehicle depart="120" id="veh22" route="route11" type="CarC" color="1,0,0" />
  <vehicle depart="125" id="veh23" route="route12" type="CarC" color="1,0,0" />
</routes>
```

IV.III.III Medidas de simulación

Las medidas de simulación que se pueden obtener con SUMO son muy amplias, a continuación presentaremos aquéllas más representativas para nosotros y de mayor utilidad para este proyecto mediante ejemplos creados sobre nuestra red manual:

- Información sobre el vehículo:

Posiciones de la fila de vehículos: nos da todas las posiciones de los vehículos a lo largo del tiempo y contiene tanto posiciones como velocidades para todos los vehículos durante todo el tiempo de simulación. Fue la primera medida implementada en SUMO y genera una gran cantidad de información en poco espacio de tiempo ya que para cada *step* (tiempo de simulación en segundos) se obtiene una salida como sigue: para cada *edge* de la red, y para cada carril de este *edge* se obtienen todos los vehículos que podemos encontrar con su identificador, posiciones en metros y la velocidad del mismo (m/s). La línea de comandos a introducir es la siguiente:

```
--netstate-dump <FILE> (or --ndump <FILE> or --netstate <FILE>)
```

El principal inconveniente es que ralentiza mucho la simulación debido a la cantidad de de información que hay que generar en tiempo real.

Sonda para el tipo de vehículos: con este detector obtendremos las posiciones de los vehículos a lo largo del tiempo de simulación para un tipo de vehículo en concreto o para todos por defecto.

Para generar este tipo de detector o sonda será necesario generar un archivo adicional con extensión `.add.xml` y luego incluirlo en el correspondiente archivo de `sumo.cfg` en la sección de *inputs* como sigue:

```
<input>
```

```
<additional-files value="vtypeprobe.add.xml"/>

</input>
```

Los comandos que describen nuestro fichero `vtyprobe.add.xml` son los siguientes:

```
<vtypeprobe id="<ID>" [ type="<VEHICLE_TYPE>" ]
freq="<OUTPUT_FREQUENCY>" file="<OUTPUT_FILE>"/>
```

En donde si indicamos el tipo de vehículos, sólo obtendremos información acerca de estos tipos de vehículos, en cambio al dejarlo en blanco y por defecto se genera información de todos los tipos de vehículos. Y la frecuencia se refiere a la periodicidad para tomar medidas y escribirlas.

La salida es como sigue:

```
<vehicle-type-probes>
  <timestep time="0.00" id="probe1" vtype="CarA" />
  <timestep time="10.00" id="probe1" vtype="CarA">
  <vehicle id="veh0" lane="D2_0" pos="0.00" x="0.00" y="3995.05" speed="0.00" />
  </timestep>
 <timestep time="20.00" id="probe1" vtype="CarA">
  <vehicle id="veh0" lane="D2_0" pos="96.01" x="96.01" y="3995.05" speed="13.28" />
  <vehicle id="veh1" lane="D2_0" pos="35.34" x="35.34" y="3995.05" speed="10.87" />
  <vehicle id="veh2" lane="D2_0" pos="0.00" x="0.00" y="3995.05" speed="0.00" />
  </timestep>
```

…… etc. hasta alcanzar el tiempo final de simulación

Información de viaje: nos aporta información para cada viaje realizado por el vehículo. El fichero de salida contiene información acerca del tiempo de salida del vehículo, el tiempo real en el cual el vehículo quiere comenzar (que puede que sea inferior al real debido al tiempo de procesado), y el tiempo de llegada de dicho vehículo; en este momento al llegar al destino se genera toda la información correspondiente al viaje antes de que el coche abandone la simulación.

El comando que fuerza esta generación es el siguiente:

```
--tripinfo-output <FILE> or --tripinfo <FILE>
```

Obteniendo para nuestra redmanual la siguiente salida:

(Nota: se recomienda ejecutarlo desde comando sin incluirlo en un archivo de configuración y ejecutar SUMO sin la opción gráfica.)

```
<tripinfos>
<tripinfo id="veh3" depart="25.00" departLane="D4_0" departPos="0.00"
departSpeed="0.00" departDelay="0.00" arrival="295.00" arrivalLane="D5_0"
arrivalPos="1.68" arrivalSpeed="13.17" duration="270.00"
routeLength="3472.98" waitSteps="0" rerouteNo="0" devices="" vtype="CarA"
vaporized="" />
```

La siguiente tabla nos muestra las posibles salidas que se obtienen con el comando `–tripinfo`:

Atributo	Unidad	Descripción
id	(vehículo) id	Identificador del vehículo
depart	(simulación) s	Momento de la simulación en la que el vehículo se emite
departLane	(carril) id	Identificador del carril donde el coche inicia su viaje
departPos	m	Posición dentro del carril donde el coche inicia su viaje
departSpeed	m/s	Velocidad del vehículo en el momento de partida
departDelay	(simulación) s	Tiempo que el vehículo tiene que esperar hasta poder empezar su viaje
arrival	(simulación) s	Tiempo en el cual el vehiculo llego a su destino
arrivalLane	(carril) id	Identificador del carril donde el coche finaliza su viaje
arrivalPos	m	Posición dentro del carril donde el coche finaliza su viaje
arrivalSpeed	m/s	Velocidad del vehículo en el momento de llegada
duration	(simulación) s	Tiempo que el vehículo necesitó para completar su ruta
routeLength	m	Longitud de la ruta del vehículo
waitSteps	Pasos de simulación	Cantidad de pasos de simulación en la que el vehículo se mantuvo por debajo de 0.1m/s
rerouteNo	#	Cantidad de vehículos que han sido reenrutados

devices	[ID]*	Lista de dispositivos que contiene el vehículo
vtype	ID	Tipo de vehículo

Tabla 4 *Outputs* de información de viaje

Información de las rutas: nos aporta información para cada ruta realizada por el vehículo, así como si se produjeron reenrutamientos durante el tiempo de ejecución. En principio es interesante para los casos en los cuales se produzca un cambio en las rutas, sino la información obtenida se puede extraer de los ficheros predefinidos `.rou.xml` y `.trips.xml`

El comando que fuerza esta generación es el siguiente:

```
--vehroute-output <FILE> or --vehroutes <FILE>
```

Medidas para los edges-carriles: con estas medidas obtendremos indicadores macroscópicos como la velocidad media del vehículo, la densidad media, etc. Esta medida se puede generar bien para carriles o para *edge*s. Necesitaremos definir un archivo adicional que contenga las siguientes directivas, en primer lugar si nos interesa medir en un *edge* y en segundo lugar si la medida se realiza para un carril:

```
<meandata-edge id="<DETECTOR_ID>" file="<OUTPUT_FILE>"/>.

<meandata-lane id="<DETECTOR_ID>" file="<OUTPUT_FILE>"/>.
```

Para nuestro ejemplo de red manual, podremos incluir un fichero que nos dé las medidas *edges* cada cierta frecuencia, donde excluiremos aquellos *edges* vacíos para evitar la redundancia y el exceso de generación de datos:

```
<add>

   <meandata-edge id="D1" freq="10" excludeEmpty="1"
   file="meanedge10.xml"/>

   <meandata-edge id="D8" freq="25" excludeEmpty="1"
   file="meanedge20.xml"/>

   <meandata-edge id="D10" freq="50" excludeEmpty="1"
   file="meanedge50.xml"/>
   <meandata-edge id="D11" freq="90" excludeEmpty="1"
   file="meanedge90.xml"/>

</add>
```

Obtendremos por tanto tres archivos y cada uno contendrá las medidas asociadas a cada *edge* con la frecuencia determinada y en el cual obviamos aquellos *edges* que se encuentran vacios para el momento de medida. Echemos un vistazo a un extracto de la salida *meanedge50.xml*, cuando la simulación está a mitad de su duración:

```
<netstats>
    <interval begin="0.00" end="50.00" id="D10">
        <edge   id="D2"   sampledSeconds="102.00"   traveltime="40.59"
density="4.08"   occupancy="1.02"   waitingTime="0.00"   speed="12.32"
departed="3"  arrived="0"  entered="0"  left="0"  laneChangedFrom="0"
laneChangedTo="0"/>

        <edge   id="D4"   sampledSeconds="57.00"   traveltime="42.99"
density="2.28"   occupancy="0.57"   waitingTime="0.00"   speed="11.63"
departed="3"  arrived="0"  entered="0"  left="0"  laneChangedFrom="0"
laneChangedTo="0"/>
<edge id="D6" sampledSeconds="13.00" traveltime="59.91" density="0.52"
occupancy="0.13"    waitingTime="0.00"    speed="8.35"    departed="2"
arrived="0"       entered="0"       left="0"       laneChangedFrom="0"
laneChangedTo="0"/>
</interval>

<interval begin="50.00" end="100.00" id="D10">
<edge id="D2" sampledSeconds="90.16" traveltime="43.47" density="3.61"
occupancy="1.26"    waitingTime="0.00"    speed="11.50"    departed="3"
arrived="0"       entered="0"       left="3"       laneChangedFrom="0"
laneChangedTo="0"/>
<edge id="D4" sampledSeconds="89.70" traveltime="43.13" density="3.59"
occupancy="1.03"    waitingTime="0.00"    speed="11.59"    departed="3"
arrived="0"       entered="0"       left="3"       laneChangedFrom="0"
laneChangedTo="0"/>

      …

</interval>
```

Los valores de la salida tanto para las medias en los *edges* como en los carriles, se interpretan como sigue.

Nombre	Tipo	Descripción
begin	(simulación) s	Primer paso de la simulación donde se recolectan valores
end	(simulación) s	Último paso de la simulación donde se recolectan valores
edge@id	(*edge*) id	Nombre del *edge* sobre el que se reporta
lane@id	(carril) id	Nombre del carril sobre el que se reporta

sampledSeconds	s	Cantidad de segundos durante los cuales se midieron vehículos, siendo este valor la suma del tiempo total de media para los vehículos.
traveltime	s	Tiempo estimado en atravesar un determinado *egde* o carril basado en la velocidad media.
density	#veh/km	Densidad vehicular
occupancy	%	Ocupación en %
waitingTime	s	Cantidad de segundos durante los cuales los vehículos estaban detenidos
speed	m/s	Velocidad media en el intervalo correspondiente
departed	#veh	Cantidad de vehículos que han sido emitidos en un carril/edge durante un determinado intervalo
arrived	#veh	Número de vehículos que han terminado su ruta
entered	#veh	Número de vehículos que han entrado en el *edge* o carril
left	#veh	Número de vehículos que han abandonado el *edge* o carril
laneChangedFrom	#veh	Cantidad de vehículos que salen del carril cambiando su ruta
laneChangedTo	#veh	Cantidad de vehículos que entran al carril cambiando su ruta

Tabla 5 Outputs para medidas medias en *edges*/carriles

Detectores simulados: a continuación veremos la forma de generar una serie de simuladores como son los lazos de inducción y las cámaras detectoras.

e1-detectors: lazos de inducción

El principal fin de este tipo de detectores es detectar los vehículos que los atraviesan una vez situados en un determinado carril. No toman medidas de superficie, sino que se consideran un plano recto que corta el carril en un determinado punto.

Será necesario definir un archivo adicional que contenga la siguiente información:

```
<additional>       <e1-detector       id="<ID>"       lane="<LANE_ID>"
pos="<POSITION_ON_LANE>"                        freq="<AGGREGATION_TIME>"
file="<OUTPUT_FILE>" [</additional>
```

Donde el id es un cadena de caracteres, *lane* determina en que carril se posicionará el `detector` y *pos* hace referencia a la posición exacta dentro de dicho carril. De nuevo el comando `freq` nos indica cada cuanto tiempo se irán tomando los datos. En nuestro caso definiremos un detector tipo E1 para cada carril de salida de los distintos *edges*, cercanos a los nodos intersección 6 y 3.

```
<sumo-detectors>
  <detector id="l100" lane="L10_0" pos="500" freq="10" file="E1redmanual.xml" />
  <detector id="l101" lane="L10_1" pos="500" freq="10" file="E1redmanual.xml" />
  <detector id="l102" lane="L10_2" pos="500" freq="10" file="E1redmanual.xml" />
  <detector id="l110" lane="L11_0" pos="500" freq="10" file="E1redmanual.xml" />
  <detector id="l111" lane="L11_1" pos="500" freq="10" file="E1redmanual.xml" />
  <detector id="l112" lane="L11_2" pos="500" freq="10" file="E1redmanual.xml" />
  <detector id="l220" lane="L22_0" pos="500" freq="10" file="E1redmanual.xml" />
  <detector id="l221" lane="L22_1" pos="500" freq="10" file="E1redmanual.xml" />
  <detector id="l222" lane="L22_2" pos="500" freq="10" file="E1redmanual.xml" />
  <detector id="l130" lane="L13_0" pos="500" freq="10" file="E1redmanual.xml" />
  <detector id="l131" lane="L13_1" pos="500" freq="10" file="E1redmanual.xml" />
  <detector id="l132" lane="L13_2" pos="500" freq="10" file="E1redmanual.xml" />
  <detector id="l180" lane="L18_0" pos="500" freq="10" file="E1redmanual.xml" />
  <detector id="l181" lane="L18_1" pos="500" freq="10" file="E1redmanual.xml" />
  <detector id="l182" lane="L18_2" pos="500" freq="10" file="E1redmanual.xml" />
  <detector id="l250" lane="L25_0" pos="500" freq="10" file="E1redmanual.xml" />
```

```
  <detector id="l251" lane="L25_1" pos="500" freq="10" file="E1redmanual.xml" />

  <detector id="l252" lane="L25_2" pos="500" freq="10" file="E1redmanual.xml" />

</sumo-detectors>
```

Las opciones de salida y su interpretación se muestran a continuación:

```
<interval begin="220.00" end="230.00" id="l180" nVehContrib="1"
flow="360.00" occupancy="3.87" speed="12.90" length="5.00"
nVehEntered="1" />
```

Nombre	Tipo	Descripción
begin	(simulación) s	Primer paso de la simulación donde se recolectan valores
end	(simulación) s	Último paso de la simulación donde se recolectan valores
id	id	Identificador del detector
nVehContrib	vehículos	Números de vehículos que pasan completamente por el detector y contribuyen a la medida
flow	vehículos /hora	Cantidad de vehículos que contribuyen a las medidas extrapolados en una hora
occupancy	%	Porcentaje de tiempo que un vehículo estuvo bajo la influencia de un detector.
speed	m/s	Velocidad media de todos los vehículos que contribuyen a las medidas
length	m	Longitud media de todos los vehículos que contribuyen a las medidas
nVehEntered	vehículos	Todos los vehículos que o bien pasan completamente por el detector y contribuyen a la medida o bien sólo lo alcanzan sin contribuir

Tabla 6 *Outputs* para los lazos de inducción

Estado de la simulación: con esta medida obtendremos información acerca del estado de la simulación. Esta salida contiene la cantidad de vehículos que son cargados, emitidos, que circulan, esperando ser emitidos, que han llegado a su lugar de destino y el tiempo que les queda para concluir con su ruta.

El comando para obtener estos datos es el siguiente:

```
  --emissions-output <FILE>   o bien -emissions <FILE>.
```

Veamos para nuestra red manual que fichero de emisiones obtendremos para un tiempo de simulación de hasta 10 segundos que será cuando el primer vehículo será emitido según la configuración de `redmanual.rou.xml`:

```
<emissions>
```

```
  <emission-state time="0.00" loaded="1" emitted="0" running="0" waiting="0"
ended="0" meanWaitingTime="-1.00" meanTravelTime="-1.00" duration="0" />
  <emission-state time="1.00" loaded="1" emitted="0" running="0" waiting="0"
ended="0" meanWaitingTime="-1.00" meanTravelTime="-1.00" duration="1" />
  <emission-state time="2.00" loaded="1" emitted="0" running="0" waiting="0"
ended="0" meanWaitingTime="-1.00" meanTravelTime="-1.00" duration="0" />
  <emission-state time="3.00" loaded="1" emitted="0" running="0" waiting="0"
ended="0" meanWaitingTime="-1.00" meanTravelTime="-1.00" duration="0" />
  <emission-state time="4.00" loaded="1" emitted="0" running="0" waiting="0"
ended="0" meanWaitingTime="-1.00" meanTravelTime="-1.00" duration="0" />
  <emission-state time="5.00" loaded="1" emitted="0" running="0" waiting="0"
ended="0" meanWaitingTime="-1.00" meanTravelTime="-1.00" duration="0" />
  <emission-state time="6.00" loaded="1" emitted="0" running="0" waiting="0"
ended="0" meanWaitingTime="-1.00" meanTravelTime="-1.00" duration="1" />
  <emission-state time="7.00" loaded="1" emitted="0" running="0" waiting="0"
ended="0" meanWaitingTime="-1.00" meanTravelTime="-1.00" duration="0" />
  <emission-state time="8.00" loaded="1" emitted="0" running="0" waiting="0"
ended="0" meanWaitingTime="-1.00" meanTravelTime="-1.00" duration="0" />
  <emission-state time="9.00" loaded="2" emitted="0" running="0" waiting="0"
ended="0" meanWaitingTime="-1.00" meanTravelTime="-1.00" duration="0" />
  <emission-state time="10.00" loaded="2" emitted="1" running="1" waiting="0"
ended="0" meanWaitingTime="0.00" meanTravelTime="-1.00" duration="0" />
```

La interpretación de los resultados es la siguiente:

Nombre	Tipo	Descripción
time	(simulacion) s	Pasos de simulación definidos en la entrada
loaded	#	Número de vehículos cargados hasta el momento
emitted	#	Número de vehículos emitidos hasta el momento
running	#	Número de vehículos rodando hasta el momento
waiting	#	Número de vehículos esperando a ser emitidos
ended	#	Número de vehículos que han alcanzado su destino
meanWaitingTime	s	Tiempo medio de espera de los vehículos para ser emitidos
meanTravelTime	s	Tiempo medio de viaje de los vehículos que han abandonado la simulación

Tabla 7 *Outputs* para estado de simulación

IV.IV) Conclusiones

En este capítulo hemos revisado el mundo de la simulación de tráfico rodado, centrándonos en una herramienta como SUMO e intentando enmarcarla dentro de las líneas principales que definen a un generador de movilidad. SUMO ofrece grandes posibilidades para modelar escenarios de diversas índoles, así como el amplio abanico de situaciones que se dan en el ambiente vial del mundo real. Es una herramienta que tiene mucho que ofrecer, ya que se encuentra en continuo desarrollo y está siendo utilizada en numerosos proyectos a nivel mundial. Por todo esto concluyo este capítulo indicando el gran interés suscitado para que SUMO se convierta en un candidato a seguir siendo explorado y para modelar los diversos escenarios que sean de interés dentro del estudio de las redes vehiculares.

V-ANÁLISIS DE UN SIMULADOR PARA VANETS

El quinto capítulo de este proyecto se centra en el análisis de la herramienta de simulación para la comunicación intervehicular. En particular, se trata de un software de tipo híbrido que conecta por un lado al programa encargado de la simulación de tráfico vehicular SUMO y por otro al simulador de redes OMNeT, mediante una interfaz desarrollada para tales efectos. Veremos pues su arquitectura, instalación, detalles de ejecución así como alguno de los resultados que pueden obtenerse.

En el capítulo anterior hemos analizado a fondo el funcionamiento de la herramienta SUMO como generador de movilidad a nivel microscópico, a continuación uniremos el concepto de simulador de tráfico rodado con el de redes de telecomunicación para dar lugar a una herramienta de análisis para redes vehiculares conocida como VEINS.

IV. I) *Vehicles in Network Simulation*, Veins

La herramienta Veins (70) cuyo nombre deriva de las siglas "*Vehicles in Network Simulation*" consiste, como se mencionó anteriormente, en un marco de simulación compuesto de un simulador a nivel microscópico de tráfico SUMO "*Simulation of Urban Mobility*" (71) y un simulador de redes basado en eventos, OMNeT++4 (72).

Estas dos herramientas se encuentran conectadas bidireccionalmente en todo momento mediante una interfaz de comunicaciones denominada TraCi que permite la simulación *on line* y retroalimentada. (73)

Sabemos que OMNeT++ es un simulador basado en eventos de tal manera que maneja la movilidad realizando una planificación de los movimientos de los nodos en intervalos regulares. SUMO también simula basándose en intervalos discretos luego ambas herramientas son susceptibles a un acoplamiento pudiéndose lograr una sincronización en tiempo ejecución para cada intervalo.

IV.II) OMNeT++ 4 y su papel como simulador de redes

El simulador de redes juega un papel principal a la hora de modelar la configuración de redes antes de ser instaladas. Haciendo uso de esta herramienta se podrán comparar diferentes instalaciones, haciendo posible la detección de problemas y consecuentemente la resolución de los mismo sin tener que invertir un amplio capital en experimentos de campo. Siendo a su vez, una pieza fundamental a la hora desarrollar nuevos protocolos de comunicaciones.

Veins hace uso del simulador basado en eventos OMNeT++ junto con INET *framework*, concretamente la versión 4.0 donde los diversos módulos reutilizables son escritos en C++ y componen una determinada jerarquía que se almacena junto con los *links* de comunicaciones en archivos del tipo .ned (*network description*). En la versión utilizada para VEINS estos módulos contiene una interfaz gráfica que permite sean modelados de una manera sencilla.

El marco de trabajo INET Framework (74) consiste en una paquete de tipo *opensource* para las simulaciones ejecutadas en OMNeT++ que contiene modelos para diversos protocolos tanto cableados como *wireless* incluyendo entre otros: UDP, TCP, SCTP, IP, IPv6, Ethernet, PPP, 802.11, MPLS, OSP. En nuestro caso nos interesan los módulos que permiten el modelado de las relaciones entre los nodos móviles y las transmisiones mediante IEEE 802.11.
En lo que a movilidad se refiere, ésta se encuentra bastante limitada, ya que por ejemplo en OMNeT++ se tienen disponibles modelos como *Random Waypoint or mass-based mobility* aunque se sabe que estos modelos no se adaptan a la realidad de las redes vehiculares. Es entonces donde aparece la idea de hacer uso de simulaciones para trazas dadas obtenidas bien de la observación del tráfico real o bien utilizando programas diseñados para tal efecto, como herramientas de generación de trazas microscópicas.

IV. III) Hacia la bidireccionalidad de las herramientas

Según lo expuesto hasta el momento tenemos por un lado un simulador de tráfico vial microscópico y por otro un simulador de redes con paquetes disponibles para simular las comunicaciones, pero no existe una retroalimentación es decir no tenemos una comunicación bidireccional entre el simulador de tráfico y el simulador de redes

Los protocolos de comunicaciones simulados en OMNeT++ no afectaran en el comportamiento de movilidad de los vehículos dispuestos segun SUMO y esto limita en todo momento la simulación de escenarios vehiculares donde las aplicaciones de seguridad como el CCA (75) son de vital importancia a la hora de evitar posibles accidentes. Pongamos por ejemplo que existe una advertencia de peligro en una vía y de manera aislada genera un mensaje para que los vehículos se desvíen de la ruta inicial, haciendo uso de las trazas esta opción no estará disponible.
Es justo en este punto, a la hora de simular redes vehiculares con aplicaciones concretas, donde aparece claramente la necesidad de la existencia de bidireccionalidad de las herramientas. La manera de conseguir esta bidireccionalidad es haciendo uso de un módulo de comunicaciones que extenderá ambas herramientas parar permitir la conexión bidireccional entre ambas y precisamente la herramienta bajo estudio en este proyecto, Veins, proporciona la retroalimentación necesaria entre ambas herramientas pudiendo así simular situaciones que incluyan VANETS.

En Veins, se han extendido ambos simuladores mediante unos módulos de comunicaciones de tal modo que mientras que la simulación ejecuta estos módulos de comunicaciones se intercambian comandos y trazas de movilidad mediante conexiones TCP. OMNeT++ es un simulador basado en eventos por lo que maneja la movilidad mediante la organización de los nodos en intervalos regulares. Esto casa a la perfección con la aproximación de SUMO que también hace avanzar la simulación en pasos discretos.

El modo de funcionamiento es como sigue:

Las unidades de comunicaciones en ambos simuladores almacenaran todos aquellos comandos que lleguen entre los correspondientes pasos de simulación garantizándose así la ejecución síncrona antes mencionada. Para cada unidad de tiempo OMNeT++ enviará los comandos almacenados a SUMO lanzando el paso de simulación para la herramienta de tráfico. Una vez termine dicho paso de simulación para SUMO éste enviará una serie de comandos y la posición de todos los vehículos de vuelta a OMNeT++. Cuando el simulador de redes recibe esta información se dispondrá el movimiento de los distintos nodos en función de la información de movilidad y una vez procesada, se procede a ejecutar el siguiente paso de la simulación. Ante las nuevas condiciones de movilidad los nodos reaccionarán a estas condiciones de entorno generando de nuevo comandos que serán enviados a SUMO y el proceso prosigue sucesivamente con la recepción y el tratamiento de dicha información por SUMO.

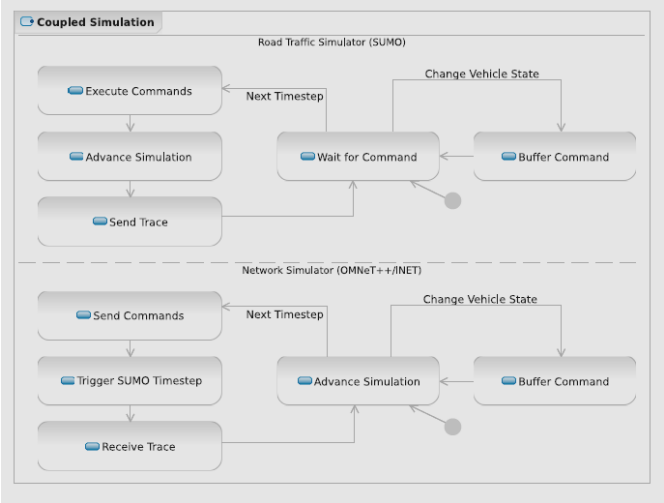


Figura 47 Esquema del *framework* de simulación. Máquinas de estado para los módulos de comunicación de los simuladores de tráfico rodado y de redes (73)

IV. III.I) Interfaz TraCi

El concepto de la interfaz de control de tráfico, TraCi (71) (76), consiste en dotar de acceso a un simulador de tráfico rodado en tiempo real, en nuestro caso proporcionar comunicación con SUMO. Basándose en una arquitectura de tipo TCP de cliente-servidor se podrá acceder a dicha herramienta, la cual actúa como un servidor. El cliente para nosotros, OMNet++4 enviará distintos comandos a SUMO para controlar el estado de la simulación, influenciando así de este modo el comportamiento individualista de los vehículos o bien recolectará información necesaria. Es decir que haciendo uso de un protocolo muy sencillo de petición-respuesta, el tráfico rodado en SUMO puede verse influenciado por OMNeT++ en una gran variedad de formas.

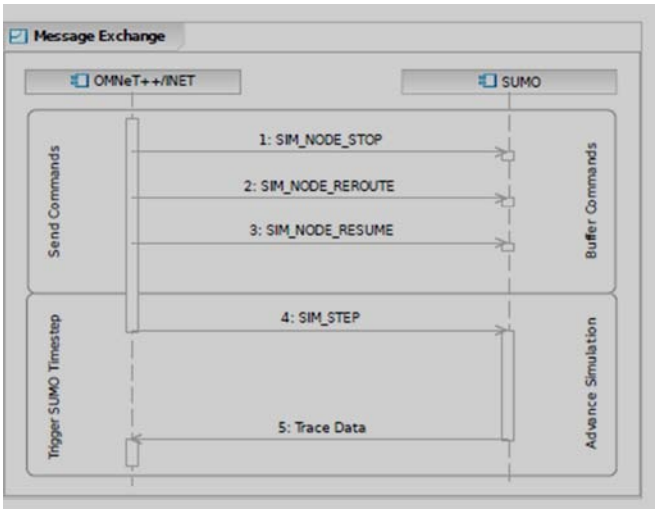


Figura 48 Intercambio de mensajes entre los simuladores (73)

En la figura anterior se muestra un diagrama de secuencia de los mensajes intercambiados mediante los módulos de comunicaciones entre los simuladores de redes y el de tráfico rodado. Encontramos las dos fases alternas de la simulación acoplada. En la primera fase los comandos son enviados a SUMO y en la segunda fase se ejecutan dichos comandos, paso que se refleja en la recepción de la correspondiente traza de movilidad enviada por este software de tráfico rodado. De esta manera ambos simuladores se encuentran unidos y SUMO sólo será capaz de ejecutar un paso de simulación después de que todos los eventos dentro de un paso de simulación hayan sido procesados en OMNeT++.

IV. III.II) Los mensajes de intercambio

Un mensaje TCP consiste de una cabecera pequeña indicando el tamaño total del mensaje y una lista de mensajes ya sean comandos o resultados a continuación de la misma. La longitud y el identificador de cada comando se sitúan al comenzar dicho comando como sigue:

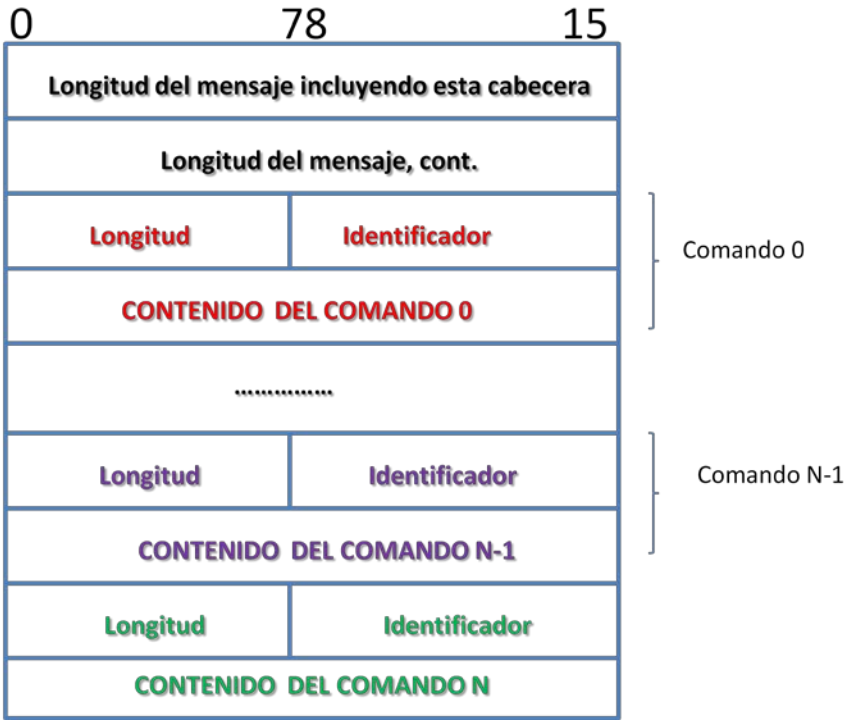


Figura 49 Estructura de mensajes TCP

SUMO responde con una trama tipo Status a cada comando y con los resultados adicionales que dependerán de lo indicado en el comando enviado. El formato de la respuesta es el siguiente:

Ubyte=8bits String=32bit

Result	Description

Donde el campo de *Description* se refiere al identificador del comando al que se le realiza el *aknowledgment* y el campo *Result* puede tomar diversos valores:

- 0x00 en caso de éxito.
- 0xFF si el comando requerido tuvo un fallo.
- 0x01si el comando requerido no se encuentra implementado en el simulador de redes.

IV. III.II.i) Estructura de los comandos

Los comandos de petición acerca de un valor determinado para un objeto tienen la siguiente estructura:

ubyte string

variable	SUMO ID

Donde la variable determina el comando en sí y el campo de SUMO ID se refiere al identificador del objeto sobre el cual pedimos la información determinada.

La respuesta de SUMO vendrá dada por la siguiente trama de información:

ubyte string ubyte <return_type>

variable	SUMO ID	return type of the variable	<VARIABLE_VALUE>

Las dos primeras posiciones de la trama se refieren a la información recibida por parte del servidor mientras que los dos últimos campos indican el tipo de variable a devolver y el valor de la misma.

Para los comandos que fijan cambio de estado nos encontramos con las siguientes estructuras:

ubyte string ubyte <value_type>

variable	SUMO ID	type of the value	new value

Los campos que se diferencian, en este caso, serán los dos últimos haciendo referencia al tipo de valor a establecer así como el valor en sí mismo correspondiendo al último campo en la trama de información. La respuesta en el caso de fijación de estado vendrá dada por los mensajes tipo *Status* definidos anteriormente.

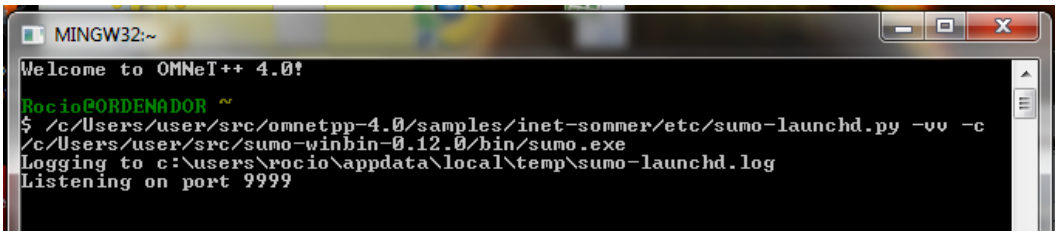
Existen números tipos de comandos manejados por TraCi sobre los cuales no entraremos en más detalle en este proyecto si bien hay que mencionar su interés para posibles modificaciones de los simuladores que se basan en SUMO.

IV. IV) La arquitectura Veins

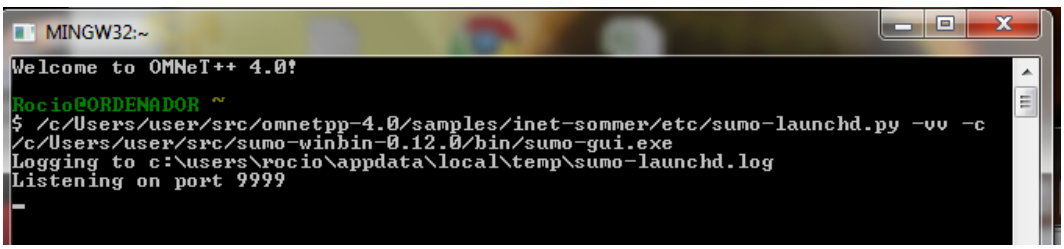
Si bien como indicábamos mediante la interfaz TraCi se realiza la interconexión entre ambas herramientas de simulación en el modo TCP de cliente (SUMO) y servidor (OMNeT++) para facilitar la ejecución de este proceso de comunicación, el módulo cliente hace uso de un dominio desarrollado en Python (77) denominado *sumo-launchd*, es importante tener instalado en la máquina el correspondiente intérprete de Python (78) el cual se dedica a "escuchar" los requerimientos del servidor, actuando así como proxy entre ambos. Evitando así tener que ejecutar SUMO manualmente antes de cada simulación de OMNeT++. Este dominio, a grandes rasgos, se encarga del manejo de los puertos y de los archivos temporales, simplificando la ejecución de la simulación, creando y eliminando según se necesiten por ejemplo las instancias relativas a SUMO. Concretamente comienza una nueva copia de la simulación de SUMO para conexión requerida por el simulador OMNeT++.

Para ejecutarlo será necesario introducir en la ventana MinGW de OMNeT++ el siguiente comando:

```
/c/Users/user/src/omnetpp-4.0/samples/inet-sommer/etc/sumo-launchd.py -vv -c
/c/Users/user/src/sumo-winbin-0.12.0/bin/sumo.exe
```



O bien para abrir la interfaz gráfica de SUMO tenemos:



Luego finalmente y haciendo uso del archivo python sumo-launchd.py la arquitectura de Veins viene representada por el siguiente esquema:

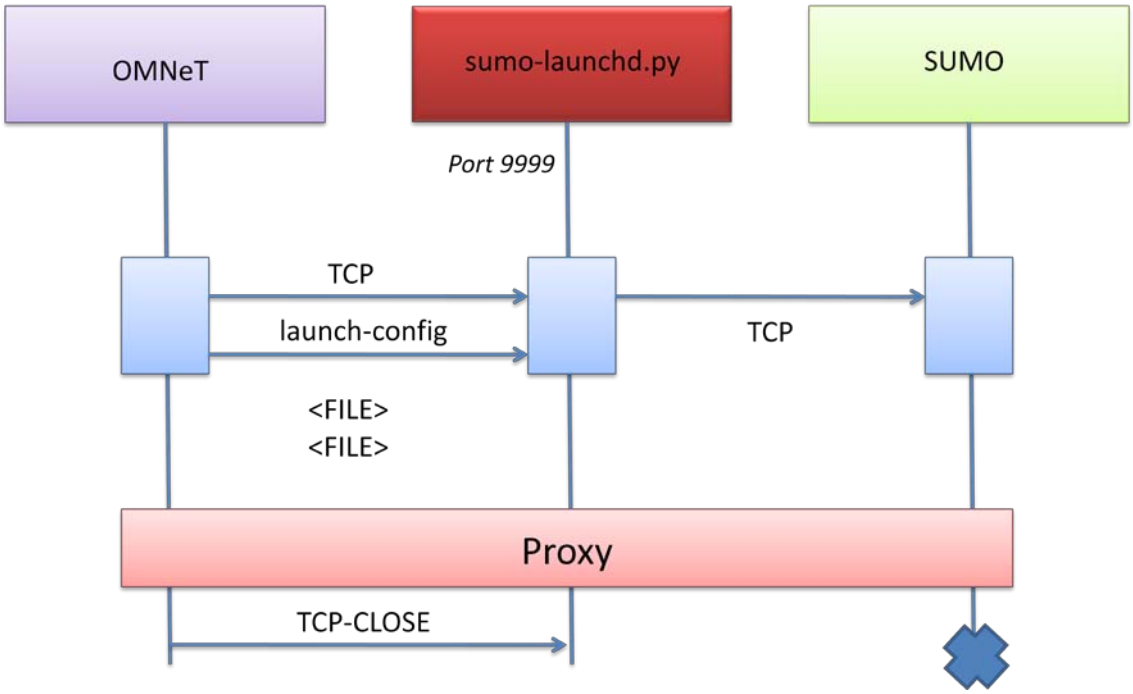


Figura 50 Arquitectura Veins con sumo-launchd.py

El siguiente paso es abrir el simulador de redes OMNeT++4.

V.V) Veins en marcha

Una vez tengamos las herramientas listas para la ejecución pasaremos a llevar a cabo la simulación vehicular propiamente dicha. Para esto tendremos que seguir los siguientes pasos:

(Nota, es muy importante vigilar el estado del *firewall* en Windows ya que éste puede bloquear el funcionamiento de SUMO impidiendo que se realice la comunicación entre ambas herramientas de manera idónea dando problemas evidentes a la interfaz TraCI).

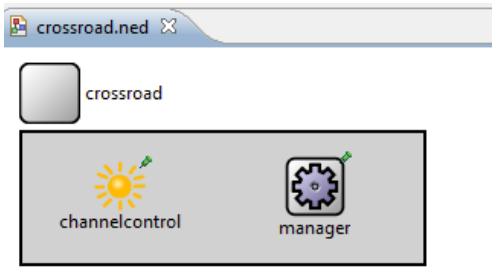
En primer lugar definiremos todos los pasos necesarios para crear nuestra simulación en Veins haciendo uso de un escenario de cruce donde dicho cruce se encuentra controlado por semáforos.

I. Lo primero será elegir la ubicación de la carpeta deseada donde ubicaremos nuestro ejemplo, en nuestro caso la denominaremos *crossroad*. Una vez creada nuestra

carpeta añadiremos en primer lugar un nuevo archivo de descripción de redes, es decir:

II. *New Network Description File (.ned*) de un archivo vacío (*empty file*) y arrastramos de manera gráfica el módulo *Network*, de las subventana *Type*, es necesarios renombrarla con el nombre de la red correspondiente, así como tener cuidado en el sentido de que es sensible a mayúsculas y minúsculas.

III. Una vez esté situado en el tapiz de diseño arrastramos gráficamente dentro del módulo 2 submódulos tal y como se muestra en la siguiente figura.



IV. Ahora tendremos que indicarle a OMNET cuáles son los archivos de SUMO que componen la red y las distintas rutas de los vehículos. Es decir tendremos que crear el archivo *launch.xml*, luego creamos un nuevo archivo en OMNeT donde se incluye la información relativa a nuestra red SUMO y contiene la siguiente información:

```
<?xml version="1.0"?>
<launch>
<copy file="net.net.xml" />
<copy file="input_routes.rou.xml" />
<copy file="test.sumo.cfg" type="config" />
</launch>
```

V. El siguiente paso, es el de crear un archivo de inicialización .ini denominando *omnetpp.ini*. La cabecera de este archivo está etiquetada en función del nombre de la red a simular y a continuación se le pasan los parámetros para las distintas capas de comunicación:

```
[General]
network = crossroad
sim-time-limit = 3500s
**.manager.launchConfig = xmldoc("crossroad.launch.xml")
**.manager.moduleType = "inet.examples.crossroad.Car"

**.channelcontrol.playgroundSizeX = 10000
```

```
**.channelcontrol.playgroundSizeY = 10000

# Network layer
**.networkLayer.ip.procDelay = 10us
**.networkLayer.arp.retryTimeout = 1s
**.networkLayer.arp.retryCount = 3
**.networkLayer.arp.cacheTimeout = 100s

# WiFi link layer
**.wlan.mgmt.frameCapacity = 10          # "maximum queuelength"

**.wlan.mac.address = "auto"       # ""auto" values will be
replaced by a generated MAC address"
**.wlan.mac.maxQueueSize = 14      # "maximum length in frames"
**.wlan.mac.bitrate = 11Mbps
**.wlan.mac.rtsThresholdBytes = 2346B    # "longer messages will
be sent using RTS/CTS; use 2346 for default" (30 header, 2312
data, 4FCS)
**.wlan.mac.retryLimit = -1        # "maximum number of retries
per message, -1 means default"
**.wlan.mac.cwMinData = -1         # "contention window for
normal data frames, -1 means default"
**.wlan.mac.cwMinBroadcast = -1          # "contention window
for broadcast messages, -1 means default"
**.wlan.radio.channelNumber = 0
**.wlan.radio.transmitterPower = 2mW
**.wlan.radio.bitrate = 11Mbps
**.wlan.radio.thermalNoise = -110dBm
**.wlan.radio.pathLossAlpha = 1.9
**.wlan.radio.snirThreshold = 3dB
**.wlan.radio.sensitivity = -85mW

# Channel Control
*.channelcontrol.carrierFrequency = 2.4GHz
*.channelcontrol.pMax = 2mW
*.channelcontrol.sat = -80dBm
*.channelcontrol.alpha = 1.9
*.channelcontrol.numChannels = 1

**.udpapp.*.vector-recording = true
**.vector-recording = true

[Config config1]
*.host[10].mobility.accidentCount = 1
*.host[10].mobility.accidentStart = 115s
*.host[10].mobility.accidentDuration = 240s
```

El siguiente paso es el de crear un módulo .ned cuyo nombre es **Car.ned** y que será definido gráficamente como sigue:

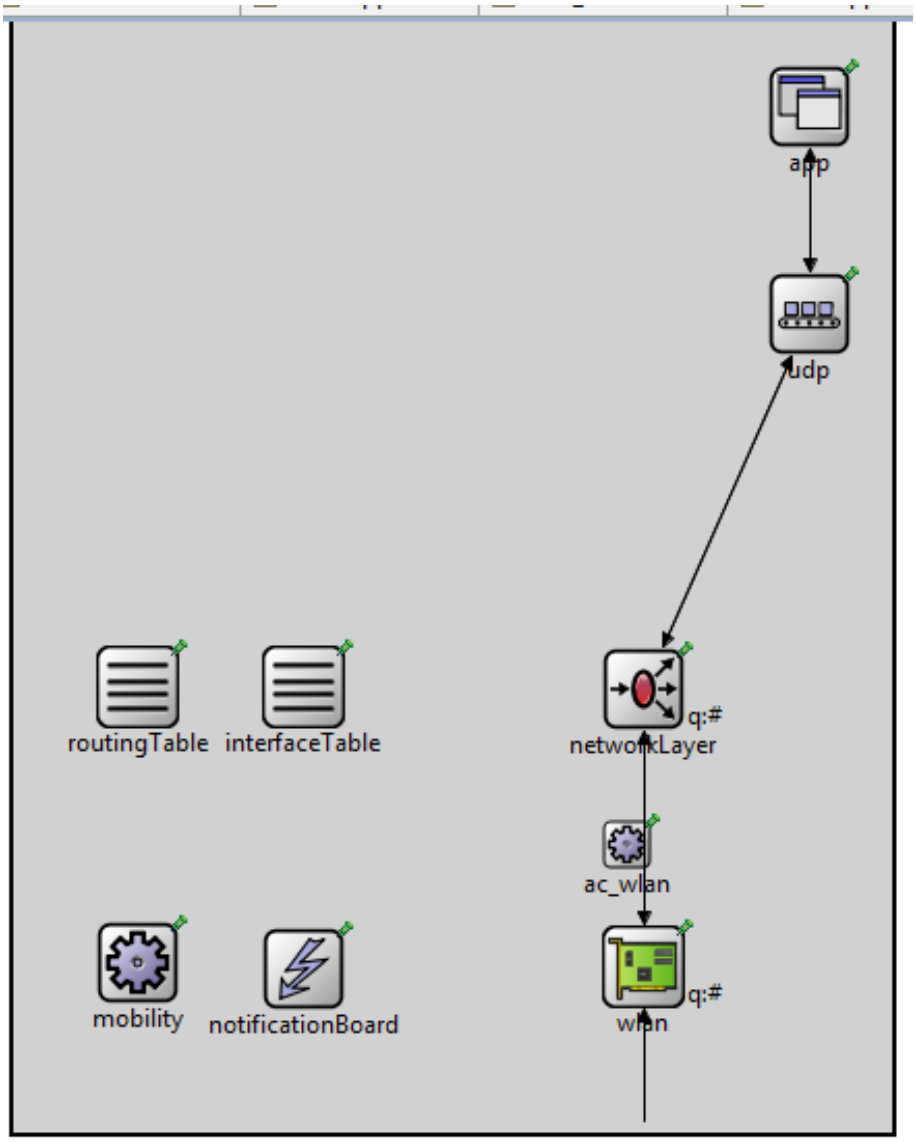


Figura 51 Car.ned

El código propiamente dicho de este módulo Car.ned que se genera según el esquemático gráfico es el siguiente:

Car.ned

```
package inet.examples.crossroad;

import inet.networklayer.autorouting.HostAutoConfigurator;
import inet.transport.udp.UDP;
import inet.nodes.inet.NetworkLayer;
import inet.networklayer.ipv4.RoutingTable;
import inet.networklayer.common.InterfaceTable;
import inet.mobility.traci.TraCIMobility;
```

```
import inet.linklayer.ieee80211.Ieee80211NicAdhoc;
import inet.base.NotificationBoard;
import inet.applications.traci.TraCIDemo;

module Car
{
    parameters:

        @display("bgb=424,541");
    gates:
        input radioIn;

    submodules:
        notificationBoard: NotificationBoard {
            parameters:
                @display("p=140,462;i=block/control");
        }
        ac_wlan: HostAutoConfigurator {
            @display("p=296,402");
        }
        interfaceTable: InterfaceTable {
            parameters:
                @display("p=140,326;i=block/table");
        }
        mobility: TraCIMobility {
            parameters:
                @display("p=60,459;i=block/cogwheel");
        }
        routingTable: RoutingTable {
            parameters:
                IPForward = true;
                routerId = "";
                routingFile = "";
                @display("p=60,326;i=block/table");
        }
        udp: UDP {
            parameters:
                @display("p=384,146;i=block/transport");
        }
        networkLayer: NetworkLayer {
            parameters:
                proxyARP = false;
                @display("p=304,327;i=block/fork;q=queue");
            gates:
                ifIn[1];
                ifOut[1];
        }
        wlan: Ieee80211NicAdhoc {
            parameters:
                @display("p=304,461;q=queue;i=block/ifcard");
        }
        app: TraCIDemo {
            parameters:
                @display("p=384,46;i=block/app");
        }
    connections allowunconnected:
        udp.appOut++ --> app.udp$i;
```

```
        udp.appIn++ <-- app.udp$o;

        udp.ipOut --> networkLayer.udpIn;
        udp.ipIn <-- networkLayer.udpOut;

        wlan.uppergateOut --> networkLayer.ifIn[0];
        wlan.uppergateIn <-- networkLayer.ifOut[0];

        radioIn --> wlan.radioIn;
}
```

Analicemos los distintos submódulos que lo forman, siendo los que siguen y su diseño se puede ejecutar de manera gráfica, de los bloques gráficos se emanara un código en C++ que también presentamos a continuación:

routingTable : RoutingTable

Tabla de enrutamiento. Almacena la tabla de enrutamiento y no contiene puertas, todas la funcionalidades se pueden acceder a través de funciones miembros de las clases de los módulos C++. El código que lo define es el que sigue:

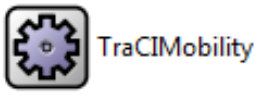
```
package inet.networklayer.ipv4;
simple RoutingTable
{
parameters:
string routerId = default("auto"); // for routers, the router id using
IP address dotted
// notation; specify "auto" to select the highest
// interface address; should be left empty ("") for hosts
bool IPForward = default(true);  // turns IP forwarding on/off
string routingFile = default("");  // routing table file name
@display("i=block/table");
}
```

interfaceTable : InterfaceTable

Almacena la tabla de las interfaces de red. Esta tabla sólo contiene propiedades independientes de los protocolos para las interfaces. Todas las funcionalidades se pueden acceder a través de funciones miembros de las clases de los módulos C++. El código que lo define es el que sigue:

```
package inet.networklayer.common;

simple InterfaceTable
{
    parameters:
        @display("i=block/table");
}
```

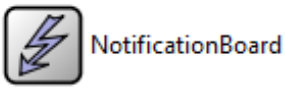
mobility : TraCIMobility



TraCIMobility es un módulo de movilidad para los *hosts* controlados por el *TraCIScenarioManager*. Recibe las actualizaciones de posición y el estado de un módulo externo y actualiza el módulo raíz de consecuentemente. El código que lo define es el que sigue:

```
package inet.mobility.traci;

simple TraCIMobility
{
parameters:
@display("i=block/cogwheel");
bool debug = default(false);  // debug switch
int accidentCount = default(0);  // number of accidents
double accidentStart @unit("s") = default(uniform(30s,60s));  // time until
first accident, relative to departure time
volatile double accidentDuration @unit("s") = default(uniform(30s,60s));
// duration of accident
volatile double accidentInterval @unit("s") = default(uniform(30s,60s));
// time between accidents
}
```

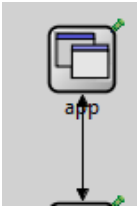
notificationBoard : NotificationBoard



Haciendo uso del módulo *NotificationBoard* los módulos se pueden comunicar unos con otros acerca de los eventos, como por ejemplo los cambios en las tablas de enrutamiento, cambios en las configuraciones de interfaz, *wireless handovers*, los cambios en el canal *wireless* de comunicaciones. Posiciones de los nodos móviles.
Éste actúa de intermediario entre los módulos que pueden sufrir algún cambio en el estado y aquellos módulos interesados en aprender acerca de esos cambios. La manera de acceder a este módulo es vía directa mediante llamadas a métodos C++. Los módulos pueden subscribirse a las categorías de los cambios, de tal manera que cuando ocurre un cambio el módulo correspondiente se lo hará saber al *NotificationBoard*, el cual diseminará la información entre todos los módulos interesados. El código que lo define es el que sigue:

```
package inet.base;

simple NotificationBoard
{
    parameters:
        @display("i=block/control");
}
```

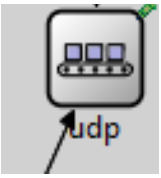
TraCIDemo



Encargado del nivel de aplicación para las comunicaciones intervehiculares, se encuentra conectado con el siguiente nivel de la capa de comunicaciones, UDP. Veremos más adelante el protocolo que define las IVC para este ejemplo. El código que lo define es el que sigue:

```
package inet.applications.traci;

simple TraCIDemo
{
parameters:
bool debug = default(false);  // output debugging information
@display("i=block/app2");
gates:
inout udp;
}
```

udp : UDP



Implementación del protocolo UDP para IPv4 y IPv6. El código que lo define es el que sigue:

```
package inet.transport.udp;
simple UDP
{
parameters:
@display("i=block/transport");
gates:
input appIn[] @labels(UDPControlInfo/down);
input ipIn @labels(UDPPacket,IPControlInfo/up);
input ipv6In @labels(UDPPacket,IPv6ControlInfo/up);
output appOut[] @labels(UDPControlInfo/up);
output ipOut @labels(UDPPacket,IPControlInfo/down);
output ipv6Out @labels(UDPPacket,IPv6ControlInfo/down);
}
```

networkLayer : NetworkLayer

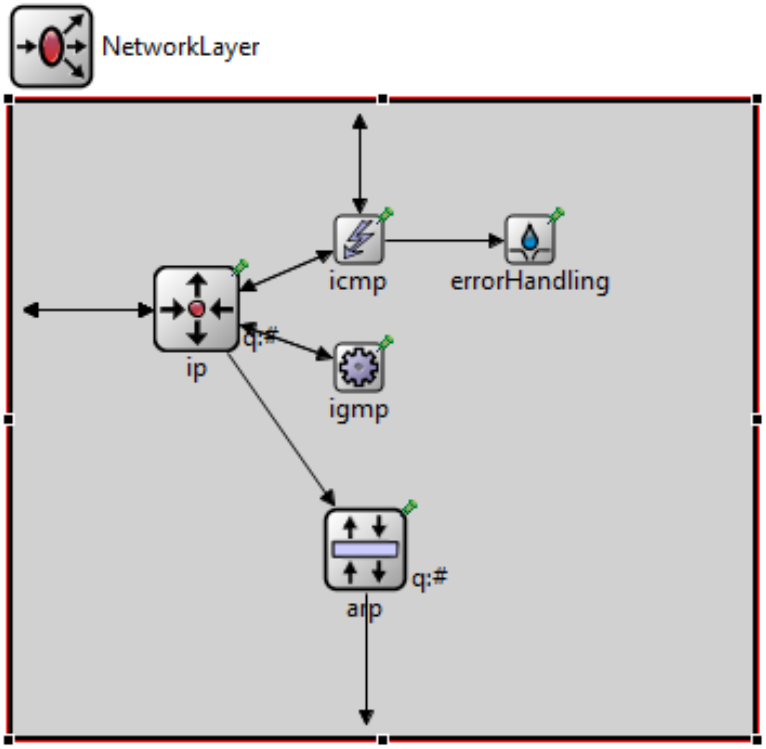


Figura 52 Esquema de *NetworkLayer*

Capa de red para un nodo IP. Las interfaces existentes para las capas de transporte son: TCP, UDP, echo/ping, RSVP. El código que lo define es el que sigue:

```
package inet.nodes.inet;

import inet.networklayer.arp.ARP;
import inet.networklayer.ipv4.ErrorHandling;
import inet.networklayer.ipv4.ICMP;
import inet.networklayer.ipv4.IGMP;
import inet.networklayer.ipv4.IP;


module NetworkLayer
{
parameters:
bool proxyARP = default(true);
@display("i=block/fork");
gates:
input ifIn[] @labels(IPDatagram);
input tcpIn @labels(TCPSegment,IPControlInfo/down);
input udpIn @labels(UDPPacket,IPControlInfo/down);
input sctpIn @labels(IPControlInfo/down,SCTPPacket);   //I.R.
input rsvpIn @labels(IPControlInfo/down);
input ospfIn @labels(IPControlInfo/down);
input pingIn;
output ifOut[];
output tcpOut @labels(TCPSegment,IPControlInfo/up);
output udpOut @labels(UDPPacket,IPControlInfo/up);
output sctpOut @labels(IPControlInfo/up,SCTPPacket); //I.R.
output rsvpOut @labels(IPControlInfo/up);
output ospfOut @labels(IPControlInfo/up);
output pingOut;

submodules:
ip: IP {
parameters:
                timeToLive = 32;
                multicastTimeToLive = 32;
                fragmentTimeout = 60s;
                protocolMapping = "6:0,17:1,1:2,2:3,46:4,89:5,132:6";
                @display("p=85,95;q=queue");
gates:
                transportIn[7];
                transportOut[7];
                queueIn[sizeof(ifIn)];
        }
        arp: ARP {
            parameters:
                proxyARP = proxyARP;
                @display("p=163,206;q=pendingQueue");
            gates:
                nicOut[sizeof(ifOut)];
        }
        icmp: ICMP {
            parameters:
                @display("p=160,63");
        }
        igmp: IGMP {
            parameters:
                @display("p=160,122");
```

```
        }
        errorHandling: ErrorHandling {
            parameters:
                @display("p=239,63");
        }
    connections allowunconnected:
        // transport Layer
        ip.transportOut[0] --> tcpOut;
        ip.transportIn[0] <-- tcpIn;

        ip.transportOut[1] --> udpOut;
        ip.transportIn[1] <-- udpIn;

        ip.transportOut[2] --> icmp.localIn;
        ip.transportIn[2] <-- icmp.sendOut;

        ip.transportOut[3] --> igmp.localIn;
        ip.transportIn[3] <-- igmp.sendOut;

        ip.transportOut[4] --> rsvpOut;
        ip.transportIn[4] <-- rsvpIn;

        ip.transportOut[5] --> ospfOut;
        ip.transportIn[5] <-- ospfIn;

        ip.transportOut[6] --> sctpOut;     //I.R.
        ip.transportIn[6] <-- sctpIn;

        icmp.pingOut --> pingOut;
        icmp.pingIn <-- pingIn;

        icmp.errorOut --> errorHandling.in;

        ip.queueOut --> arp.ipIn;

        // L2 interfaces to IP and from ARP
        for i=0..sizeof(ifOut)-1 {
            ifIn[i] --> { @display("m=s"); } --> ip.queueIn[i];
            ifOut[i] <-- { @display("m=s"); } <-- arp.nicOut[i];
        }
}
```

El módulo de red está formado a su vez por los submódulos siguientes:

1. ip : IP

Implementa el protocolo IP . El código que lo define es el que sigue:

```
package inet.networklayer.ipv4;
simple IP
{
parameters:
double procDelay @unit("s") = default(0s);
int timeToLive = default(32);
int multicastTimeToLive;
string protocolMapping;
double fragmentTimeout @unit("s") = default(60s);
@display("i=block/routing");
gates:
input transportIn[] @labels(IPControlInfo/down,TCPSegment,UDPPacket);
output transportOut[] @labels(IPControlInfo/up,TCPSegment,UDPPacket);
input queueIn[] @labels(IPDatagram);
output queueOut @labels(IPDatagram);
}
```

2. icmp : ICMP

Implementación para *Internet Control Message Protocol*. El código que lo define es el que sigue:

```
simple ICMP
{
parameters:
@display("i=block/control_s");
gates:
input localIn @labels(IPControlInfo/up);  // delivered ICMP packets
input pingIn;  // ping requests from app
output pingOut;  // result of ping
output sendOut @labels(IPControlInfo/down);  // towards network
output errorOut;  // errors
}
```

3. igmp : IGMP

Implementación de *Internet Group Management Protocol*. El código que lo define es el que sigue:

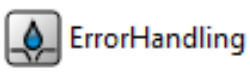
```
package inet.networklayer.ipv4;

module IGMP
{
parameters:
@display("i=block/cogwheel_s");
gates:
input localIn @labels(IPControlInfo/up);  // delivered IGMP packets
output sendOut @labels(IPControlInfo/down); // to IP
connections allowunconnected:
}
```

4. arp : ARP

Implementación del protocolo: *Address Resolution Protocol* para IPv4 Y y direcciones MAC IEEE 802 de 6 bytes. El código que lo define es el que sigue:

```
package inet.networklayer.arp;

simple ARP
{
parameters:
double retryTimeout @unit("s") = default(1s); // number seconds ARP
waits between retries to resolve an \IP address
int retryCount = default(3);   // number of times ARP will attempt to
resolve an \IP address
double cacheTimeout @unit("s") = default(120s); // number seconds
unused entries in the cache will time out
bool proxyARP = default(true);        // sets proxy \ARP mode
(replying to \ARP requests for the addresses for which a routing table
entry exists)
@display("i=block/layer");
gates:
input ipIn @labels(ARPPacket,IPDatagram);
output nicOut[] @labels(ARPPacket);
}
```

5. errorHandling : ErrorHandling



Este módulo maneja las notificaciones de error que llegan por parte de otros módulos de protocolos. El código que lo define es el que sigue:

```
package inet.networklayer.ipv4;

simple ErrorHandling
{
parameters:
@display("i=block/sink_s");
gates:
input in;
}
```

Ya fuera del modulo de red nos encontramos con los siguientes módulos:

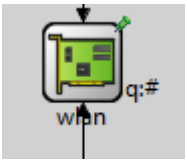
ac_wlan : HostAutoConfigurator

Este módulo, *HostAutoConfigurator* automáticamente asigna direcciones IP y establece las tablas de enrutamiento.

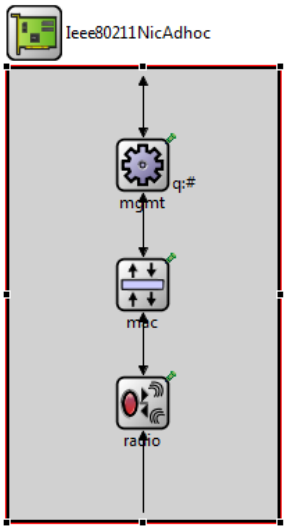
El código que lo define es el que sigue:

```
package inet.networklayer.autorouting;

simple HostAutoConfigurator
{
parameters:
bool debug = default(false);// emit debug messages?
string interfaces = default("wlan");// list of interfaces to
autoassign addresses to, separated by a single space characters
string addressBase = default("10.0.0.0");    // start of address range
from which to automatically assign an address to the
autoassignInterfaces
string netmask = default("255.0.0.0");// netmask of subnet in which to
automatically assign an address to the autoassignInterfaces
string mcastGroups = default("");// list of IP addresses of multicast
groups to join, separated by a single space characters
@display("i=block/cogwheel_s");
}
```

wlan : Ieee80211NicAdhoc



Este módulo implementa la interfaz de la tarjeta para las comunicaciones 802.11 en modo *ad-hoc*. A su vez, este módulo está compuesto por los siguientes sub-módulos:



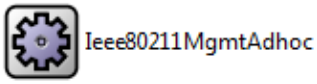
El código que lo define es el que sigue:

```
package inet.linklayer.ieee80211;

import inet.linklayer.ieee80211.mac.Ieee80211Mac;
import inet.linklayer.ieee80211.mgmt.Ieee80211MgmtAdhoc;
import inet.linklayer.radio.Ieee80211Radio;


module Ieee80211NicAdhoc
{
parameters:
@display("i=block/ifcard");
gates:
        input uppergateIn; // to upper layers
        output uppergateOut; // from upper layers
        input radioIn @labels(AirFrame); // to receive AirFrames
submodules:

mgmt: Ieee80211MgmtAdhoc {
            parameters:
                @display("p=96,69;q=wlanDataQueue");
        }
        mac: Ieee80211Mac {
            parameters:
                queueModule = "mgmt";
                @display("p=96,155");
        }
        radio: Ieee80211Radio {
            parameters:
                @display("p=96,240");
        }
    connections:
        radioIn --> radio.radioIn;
        radio.uppergateIn <-- mac.lowergateOut;
```

```
        radio.uppergateOut --> mac.lowergateIn;

        mac.uppergateOut --> mgmt.macIn;
        mac.uppergateIn <-- mgmt.macOut;

        mgmt.uppergateOut --> uppergateOut;
        mgmt.uppergateIn <-- uppergateIn;

}
```

Éste módulo está formado a su vez por los siguientes submódulos:

1. mgmt : Ieee80211MgmtAdhoc



Éste es el módulo de gestión utilizado para el modo 802.11 *ad hoc*. Se basa en la capa MAC (Ieee80211Mac) para la recepción y transmisión de tramas. Esta aplicación no envía tramas de control o gestión, y descarta cualquier trama de este estilo recibida.

```
simple Ieee80211MgmtAdhoc like Ieee80211Mgmt
{
    parameters:
        int frameCapacity = default(100);
        @display("i=block/cogwheel");
    gates:
        input uppergateIn;
        output uppergateOut;
        input macIn @labels(Ieee80211Frame);
        output macOut @labels(Ieee80211Frame);
}
```

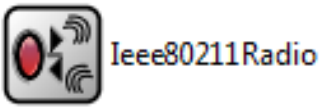
2. mac : Ieee80211Mac

Implementación de la aplicación del protocolo 802.11b MAC. Este módulo está destinado a ser utilizado en combinación con el módulo Ieee80211Radio como capa física.

```
package inet.linklayer.ieee80211.mac;

simple Ieee80211Mac
{
 parameters:
```

```
string address = default("auto"); // MAC address as hex string (12 hex
digits), or// "auto". "auto" values will be replaced by
                                        // a generated MAC address
in init stage 0.
string queueModule = default("");    // name of optional external
queue module
int maxQueueSize; // max queue length in frames; only used if
queueModule==""
double bitrate @unit("bps");
int rtsThresholdBytes @unit("B") = default(2346B); // longer messages
will be sent using RTS/CTS
int retryLimit = default(-1); // maximum number of retries per
message, -1 means default
int cwMinData = default(-1); // contention window for normal data
frames, -1 means default
int cwMinBroadcast = default(-1); // contention window for broadcast
messages, -1 means default
int mtu = default(1500);
@display("i=block/layer");
gates:
      input uppergateIn @labels(Ieee80211Frame);
      output uppergateOut @labels(Ieee80211Frame);
      input lowergateIn @labels(Ieee80211Frame);
      output lowergateOut @labels(Ieee80211Frame);
}
```

3. radio : Ieee80211Radio



Capa física para los modelos IEEE 802.11.

See also: Radio

```
package inet.linklayer.radio;

simple Ieee80211Radio like Radio
{
parameters:
int channelNumber = default(0); // channel identifier
double transmitterPower @unit("mW") = default(20mW); // power used for
transmission of messages (in mW)
double bitrate @unit("bps"); // (in bits/s)
double thermalNoise @unit("dBm") = default(-110dBm); // base noise
level (dBm)
double pathLossAlpha = default(2); // used by the path loss
calculation
double snirThreshold @unit("dB") = default(4dB); // if signal-noise
ratio is below this threshold, frame is considered noise (in dB)
```

```
double sensitivity @unit("mW"); // received signals with power below
sensitivity are ignored
@display("i=block/wrxtx");

gates:
input uppergateIn @labels(PhyControlInfo/down,Ieee80211Frame);    //
from higher layer protocol (MAC)
output uppergateOut @labels(PhyControlInfo/up,Ieee80211Frame); // to
decider (decider connects to higher layer protocol, i.e. the MAC)
input radioIn @labels(AirFrame); // to receive frames (AirFrame) on
the radio channel
}
```

Hasta aquí llegamos con el estudio del módulo *Car.Ned*, donde se define de que manera a nivel comunicación será traducido cada nodo móvil o automóvil. Para la capa de aplicación necesitaremos incluir los siguientes ficheros, donde definiremos los protocolos de IVC (*Intervehicles commuications*) o comunicación intervehicular:

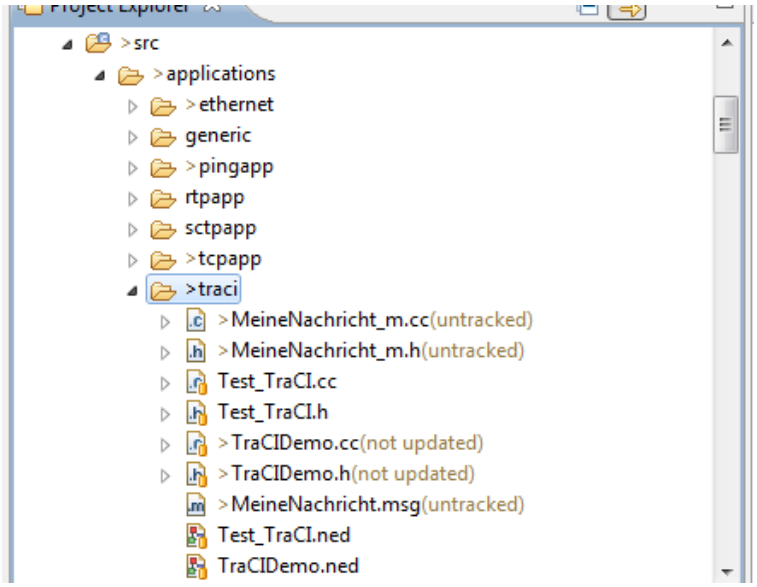


Figura 53 Árbol de ficheros para traci (OMNeT)

Si analizamos el módulo TraciDemo.cc:

TraciDemo.cc

Es donde se encuentra el protocolo de *flooding*:

```
#include <algorithm>
#include <numeric>
```

```
#include "applications/traci/TraCIDemo.h"
#include "NotificationBoard.h"
#include "MeineNachricht_m.h"

Define_Module(TraCIDemo);

TraCIDemo::~TraCIDemo() {
}

void TraCIDemo::initialize(int aStage) {

BasicModule::initialize(aStage);

if (0 == aStage) {
debug = par("debug");
traci = TraCIMobilityAccess().get();
triggeredFlooding = false;

NotificationBoard* nb = NotificationBoardAccess().get();
nb->subscribe(this, NF_HOSTPOSITION_UPDATED);

meinVektor.setName("DasIstMeinVektor");

setupLowerLayer();

empfangeneIds.clear();
      }
}


void TraCIDemo::setupLowerLayer() {
cMessage *msg = new cMessage("UDP_C_BIND", UDP_C_BIND);
UDPControlInfo *ctrl = new UDPControlInfo();
ctrl->setSrcPort(12345);
ctrl->setSockId(UDPSocket::generateSocketId());
msg->setControlInfo(ctrl);
send(msg, "udp$o");
}

void TraCIDemo::finish() {
recordScalar("numMessagesSent", 42);
}

void TraCIDemo::receiveChangeNotification(int category, const
cPolymorphic *details) {
Enter_Method("receiveChangeNotification()");
if (category == NF_HOSTPOSITION_UPDATED) {
handlePositionUpdate();
      } else {
error("should only be subscribed to NF_HOSTPOSITION_UPDATED, but
received notification of category %d", category);
      }
}

void TraCIDemo::handleMessage(cMessage* apMsg) {
      if (apMsg->isSelfMessage()) {
            handleSelfMsg(apMsg);
```

```
      } else {
            handleLowerMsg(apMsg);
      }
}

void TraCIDemo::handleSelfMsg(cMessage* apMsg) {
}


void TraCIDemo::handleLowerMsg(cMessage* apMsg) {
if (cPacket* m = dynamic_cast<cPacket*>(apMsg)) {
if (MeineNachricht* mm = dynamic_cast<MeineNachricht*>(m)) {
std::cout << "empfange " << mm->getId() << std::endl;

if (empfangeneIds.find(mm->getId()) == empfangeneIds.end() ) {
std::cout << "noch nicht bekommen " << mm->getId() << std::endl;
sendMessage(mm->getId());
empfangeneIds.insert(mm->getId());
                  }
            }
      }

      delete apMsg;
}

void TraCIDemo::handlePositionUpdate() {
if ((traci->getPosition().x < 7350) && (!triggeredFlooding)) {
triggeredFlooding = true;
sendMessage(-1);
      }
}

void TraCIDemo::sendMessage(int id) {
static int vergebeneIds = 0;
MeineNachricht* newMessage = new MeineNachricht();
std::cout << "sendMessage " << id << std::endl;
meinVektor.record(id);
if (id == -1) id = vergebeneIds++;
newMessage->setId(id);

newMessage->setKind(UDP_C_DATA);
UDPControlInfo *ctrl = new UDPControlInfo();
ctrl->setSrcPort(12345);
ctrl->setDestAddr(IPAddress::ALL_HOSTS_MCAST);
ctrl->setDestPort(12345);
delete(newMessage->removeControlInfo());
newMessage->setControlInfo(ctrl);

sendDelayed(newMessage, 0.010, "udp$o");
std::cout << "gesendet " << newMessage->getId() << std::endl;
}
```

En este simple protocolo de *flooding*, cada vez que un nodo recibe un mensaje del tipo mm, hará correr el siguiente código de tal manera que cada mensaje se enviará exactamente una vez a sus nodos vecinos.

```
if (empfangeneIds.find(mm->getId()) == empfangeneIds.end() ) {
sendMessage(mm->getId());
empfangeneIds.insert(mm->getId());
}
```

V.VI) Escenarios y resultados

Como escenario a simular tendremos un cruce creado en SUMO denominando *crossroad.xml.* En el momento que la herramienta comienza a funcionar SUMO que se encontraba esperando a OMNET recibirá los primeros mensajes de sincronización a través de la TraCI:

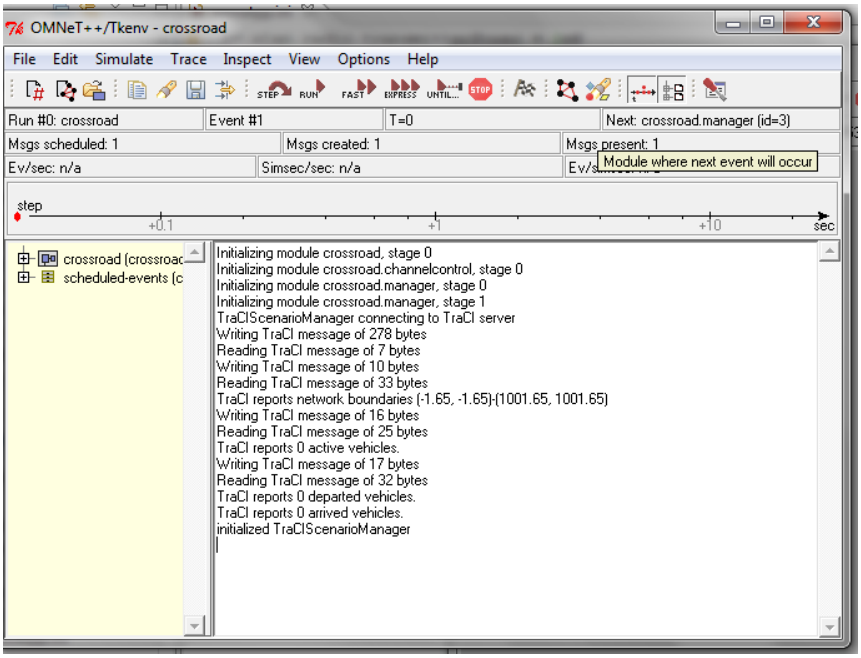


Figura 54 Mensajes TraCI en OMNeT

Durante la simulación podremos observar a través de la interfaz gráfica como ambas herramientas se encuentran sincronizadas:

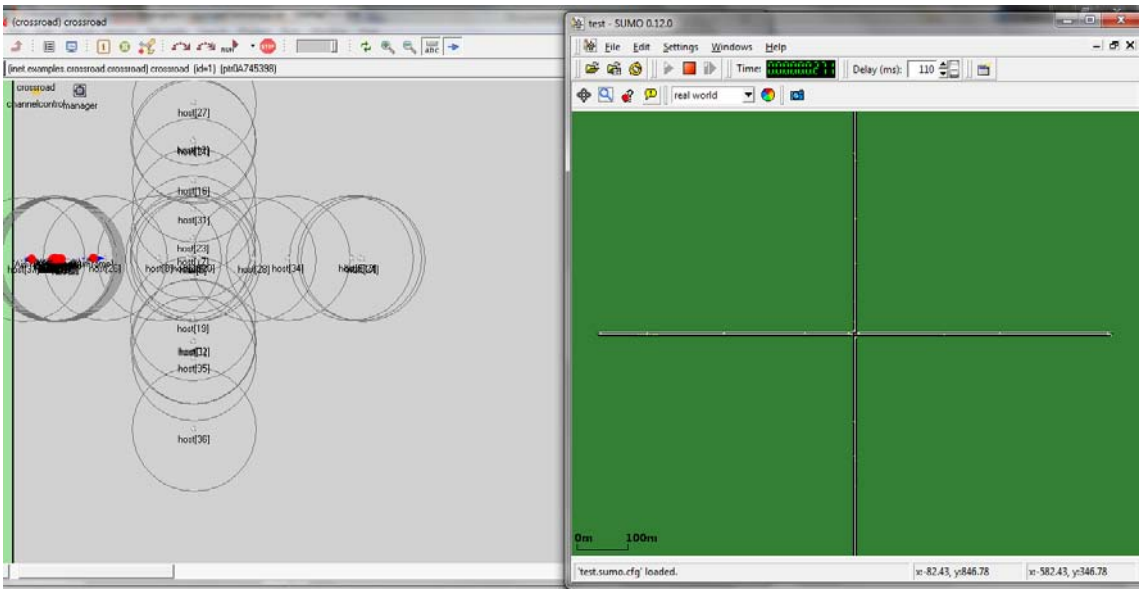


Figura 55 Veins en funcionamiento: OMNeT y SUMO acopladas

Detalles de la simulación para ambas herramientas:

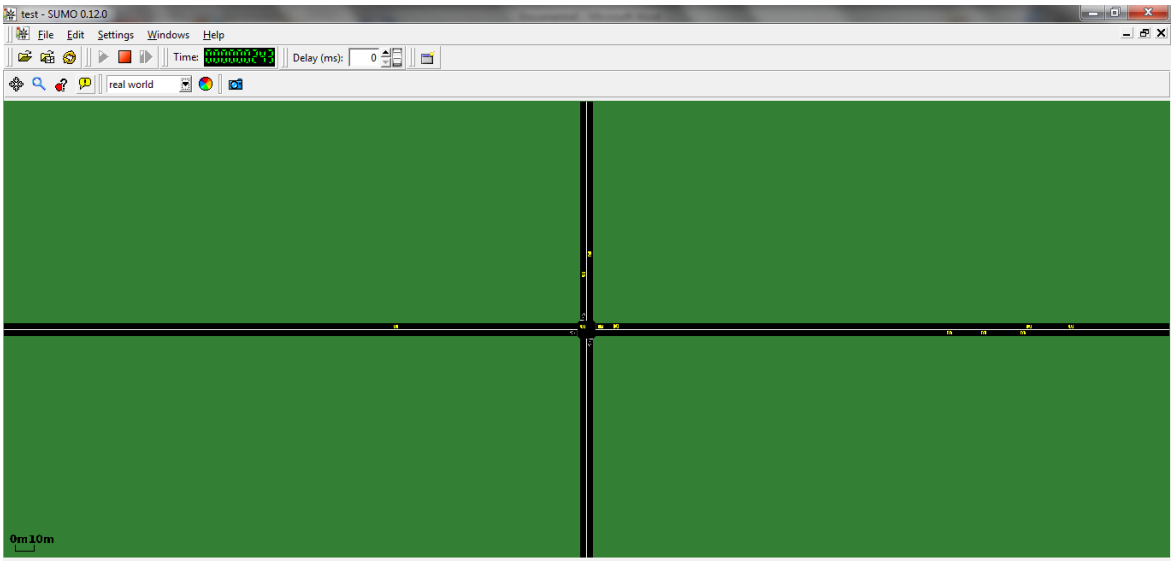


Figura 56 Ampliación de la intersección en SUMO

Figura 57 Ampliación donde se detectan vehículos en SUMO

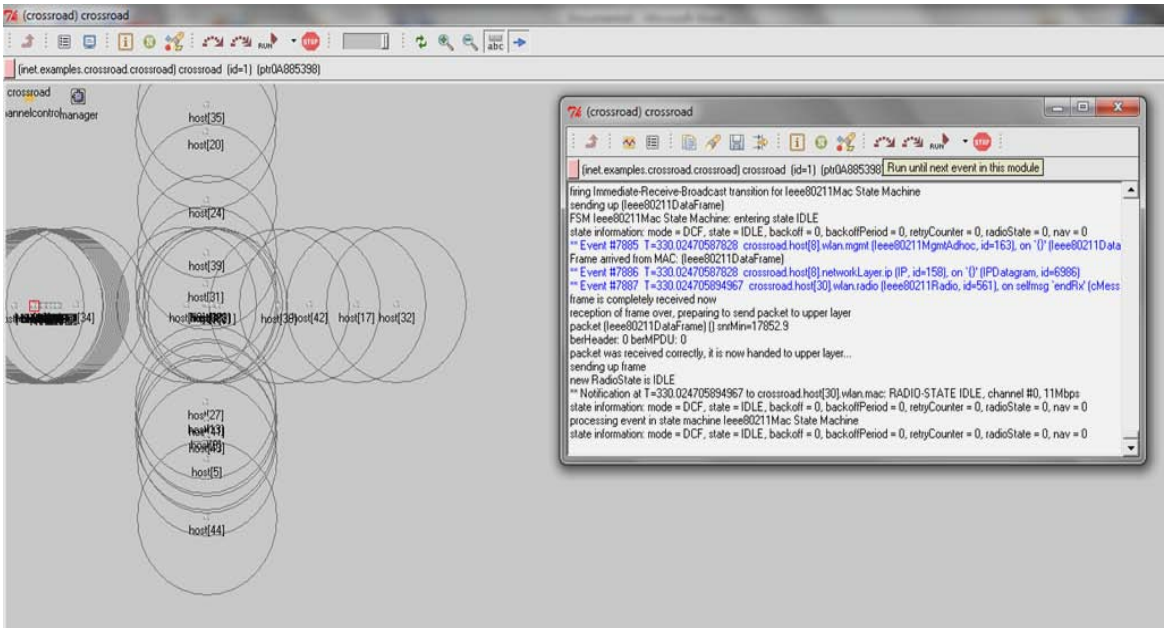


Figura 58 Nodos enviando mensajes y detalle de eventos en OMNeT

Al final de la simulación decidiremos si los resultados que podemos obtener se presentarán en medias escalares o medidas vectoriales, obtenidas con OMNeT:

Medidas escalares: a continuación presentamos algunas de las medidas escalares que se pueden obtener tras realizar la simulación:

A) Distancia recorrida por cada uno de los nodos.

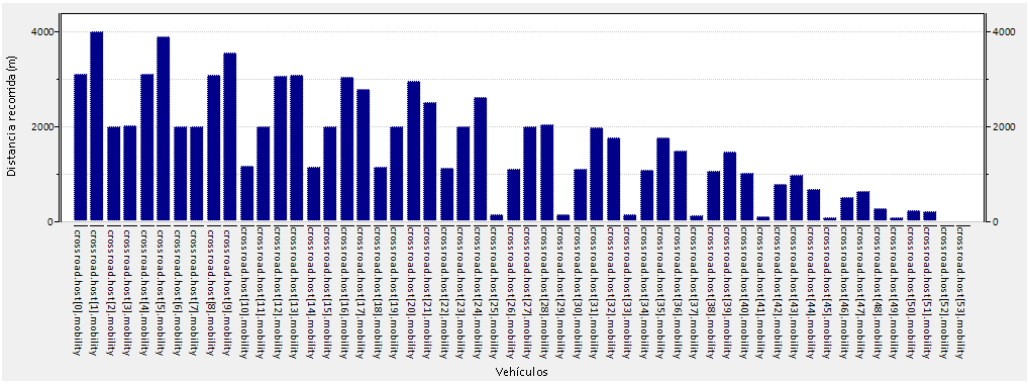


Figura 59 Medida escalar para las distancias recorridas por los nodos

B) velocidad máxima alcanzada por cada uno de los nodos.

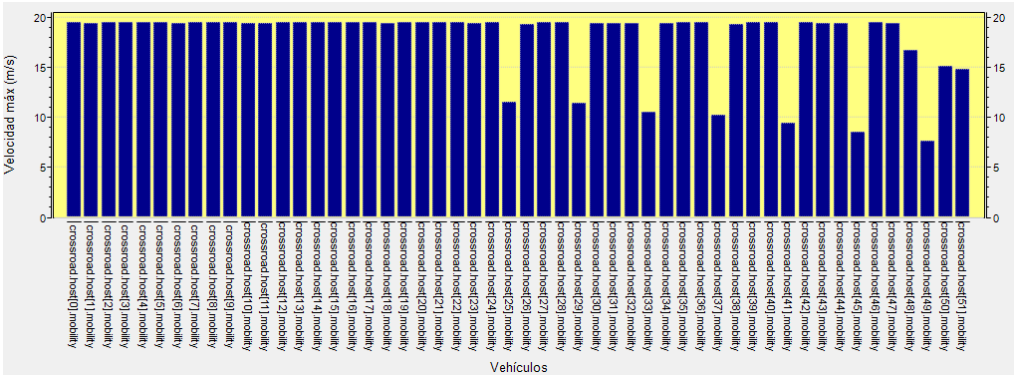


Figura 60 Medida escalar para las velocidades máximas de los nodos

C) Momento en el que los nodos terminan la simulación

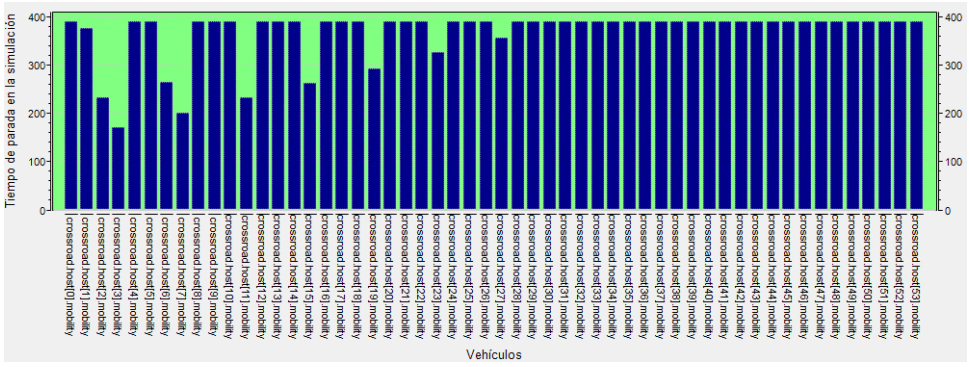


Figura 61 Medida escalar para los tiempos en los que los nodos terminan la simulación

Medidas vectoriales

Analicemos en primer lugar las medidas relativas a la movilidad pertenecientes al siguiente módulo: **Crossroad.host[*].mobility.**

Por ejemplo para el host, también denominado nodo "0" calcularemos la velocidad que dicho vehículo lleva en todo el trayecto de simulación para luego calcular su velocidad media:

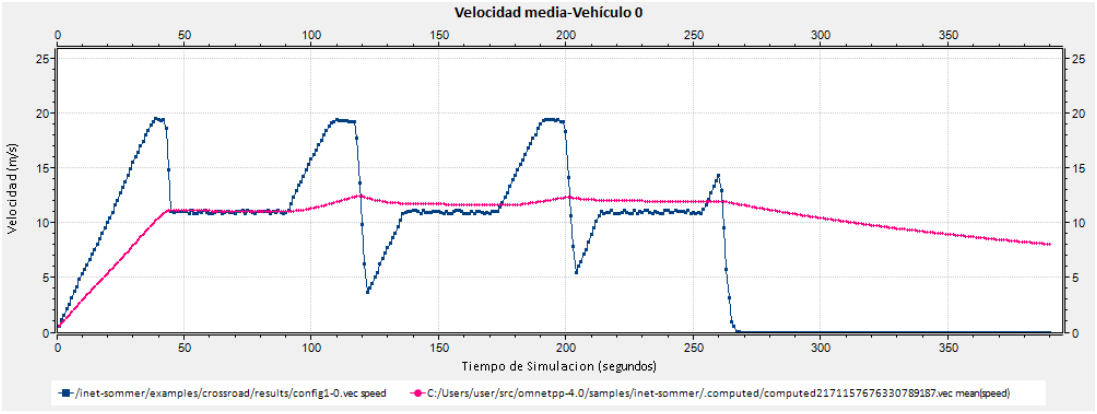


Figura 62 Medidas de velocidad junto con velocidad media para Nodo 0

Así mismo se puede obtener los valores de aceleración y comparándolos con la velocidad se observa como los picos de bajada corresponden a frenadas y/o disminución de la velocidad de dicho vehículo:

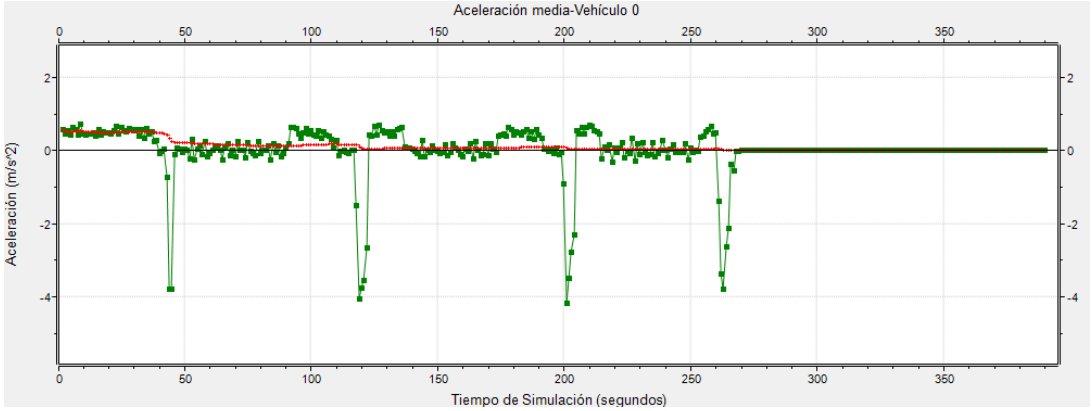


Figura 63 Medidas de aceleración junto con aceleración media para Nodo 0

Analicemos en primer lugar las medidas relativas a la capa de aplicación pertenecientes al siguiente módulo: **Crossroad.host[*].app.**

La siguiente gráfica muestra los mensajes enviados por cada uno de los nodos, según indicamos en el módulo *tracidemo.cc* para que se fueran almacenando en un vector, los mensajes enviados por cada nodo y su identificador correspondiente.

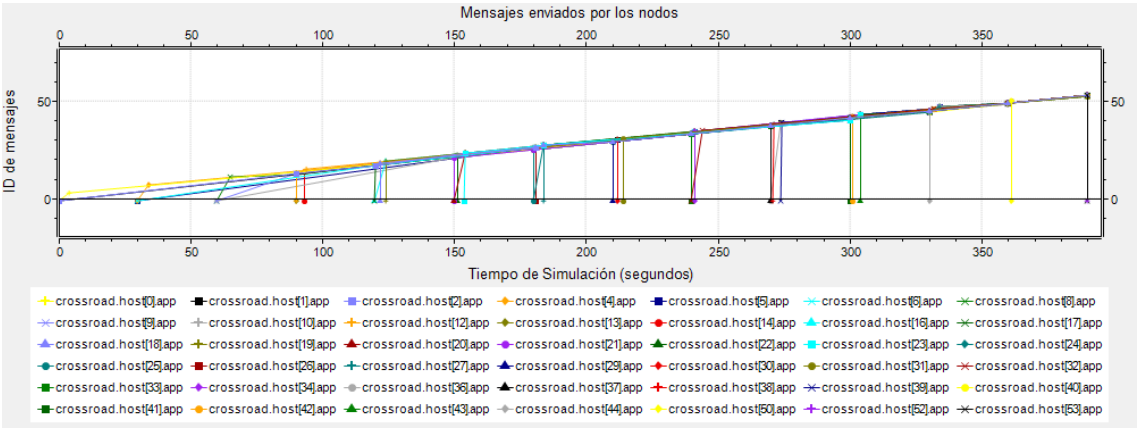


Figura 64 Mensajes con id enviados por los 54 nodos

Si representamos el total de los mensajes enviados por cada uno de los nodos tenemos la siguiente gráfica:

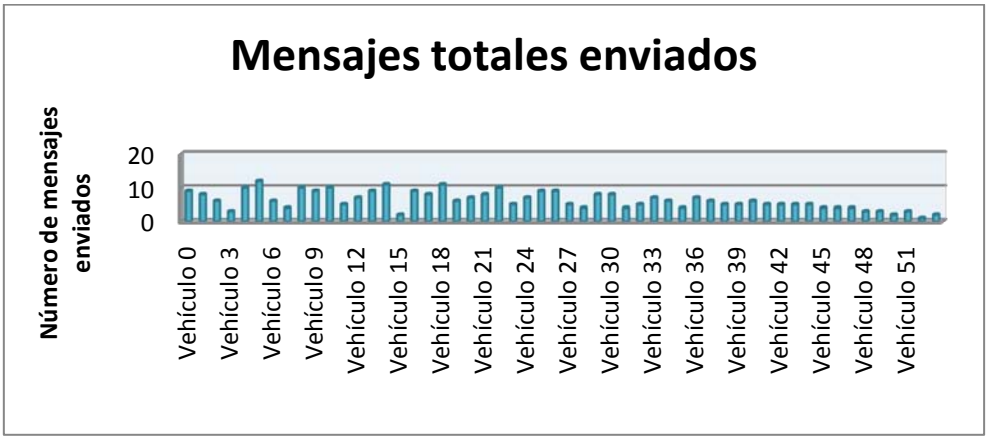


Figura 65 Detalle de mensajes enviados por cada nodo

Concluyendo, en este capítulo V se ha ilustrado el procedimiento para entender, instalar, configurar y tomar medidas en una herramienta dedicada al análisis de las redes vehiculares. Si bien esto es sólo el comienzo de las numerosas posibilidades de cómputo y simulación que nos puede ofrecer una herramienta híbrida como Veins, presentada en este proyecto.

Conclusiones y trabajo futuro

Conclusiones

A través de los cinco capítulos de este proyecto se han abarcado diversas ramas, todas procedentes de un mismo tema troncal: las herramientas de tráfico vehicular. Si bien merece la pena detenerse en cada una de ellas para impregnarnos de las ideas y las conclusiones obtenidas por separado para así al final aunarlas todas en un mismo árbol de conocimiento e investigación común, la simulación de redes VANET.

Comencemos con las raíces del trabajo, la motivación de este proyecto es conocer mejor las redes vehiculares así como por qué la simulación juega un papel tan importante en las mismas. Lo que nos queda tras leer el primer capítulo es una muestra clara de la relevancia que las redes vehiculares tienen y el alcance que pueden llegar a tener en un futuro no muy lejano y como las aplicaciones de las mismas, sobre todo las orientadas a aumentar la seguridad vial y a mejorar la calidad de vida de los usuarios. Como toda tecnología de nueva implantación tiene unas limitaciones tecnológicas y sociales si bien son barreras superables si todas las comunidades interesadas aúnan esfuerzos, de ahí los numerosos proyectos que se están llevando a cabo en Europa, así como en el resto del mundo. La simulación juega un papel más que principal en hacer que estos proyectos VANET vean la luz, ya que la simulación del tráfico vehicular contribuye al estudio del tráfico en su amplia variedad sin tener que realizar experimentos de campo, permitiendo jugar con las variables que pueden ser modificadas para llegar a soluciones que no impliquen inversiones a ciegas y sin necesidad de construir nuevas y costosas infraestructuras. Las herramientas de simulación VANET permiten entender los principales retos y dificultades en comparación con otras redes *wireless* así como llevar a cabo simulaciones que evalúen nuevos protocolos de comunicación. A pesar de que el nivel de detalles en estas redes vehiculares basadas en modelos microscópicos de tráfico conducen a la necesidad de contar con un alto nivel de computación. Si pensamos en simular escenarios muy amplios, como puede llegar a ser una ciudad, con todos los componentes viales que en ella se hallan, así como protocolos de comunicación elaborados con alto nivel de detalle en todas sus capas es cuando tenemos que pensar en otro tipo de máquinas para que resuelvan los cálculos. El capítulo II nos lleva de la mano por el mundo de la supercomputación, los superordenadores, la aceleración de cómputo, así como técnicas de programación paralela. Ilustrando con un ejemplo de módelo matemático que requiere una gran cantidad de recursos para su resolución y cómputo cómo es posible resolver mediante técnicas de paralelización y el uso de máquinas como el *Ben-Arabí* un problema que en un principio era prácticamente inabordable.

En este punto del proyecto en el capítulo III, se presenta el resultado de una búsqueda exhaustiva y de un estudio pormenorizado de las herramientas *opensource* y *freeware* para la simulación vehicular que hoy en día podemos encontrar. Tras este análisis a nivel taxonómico y funcional se decide analizar una de las herramientas híbridas para VANET encontradas, ya que aúna mediante un acople bidireccional dos herramientas de gran penetración cada una en su campo, a saber la herramienta Veins. Esta herramienta está formada por un lado por el generador de movilidad SUMO junto con el simulador de redes OMNeT. Llegamos al capítulo IV donde se obtienen resultados concluyentes acerca de cómo hacer uso de la herramienta SUMO para crear escenarios de tráfico vehicular acorde con nuestras necesidades para la simulación.

Finalmente en el capítulo VI hacemos uso de la herramienta Veins, acoplando bidireccionalmente un escenario de un cruce elaborado en SUMO con una red de comunicación implementada en OMNET, haciendo que los vehículos que aparecen en el escenario de simulación se envíen mensajes mediante un algoritmo de *flooding*. También se presentan medidas de movilidad y de la capa de aplicación.

Como conclusiones personales admitir que ha sido un reto omenzar a adentrarme en el mundo VANET desconocido totalmente para mí hasta el momento. Al lidiar con herramientas de simulación de tipo *opensource* me he encontrado con varios obstáculos en el camino, como la dificultad no sólo de instalación y de funcionamiento de éstas sino que en muchos casos no existe un manual de usuario completo o actualizado, así como una mínima documentación en los que poder apoyarse. Gracias al tiempo de estancia en la *Technische Universität* en Munich, pude visitar en Erlangen al creador de la herramienta Veins, Christoph Sommer. Co él puede también comprender que el mundo de la simulación vehicular, es algo que toma su tiempo para poder entender y modelar, si bien merece la pena los resultados que se obtienen, en especial por que también tienen un importante sentido social, lo que no ocurre con otras disciplinas.

Trabajo Futuro

Como trabajos futuros se plantean diversos enfoques en las líneas de investigación:

- ✓ Por un lado continuar a fondo con la herramienta Veins, explorando nuevos protocolos de comunicaciones así como diversos escenarios creados con SUMO.
- ✓ Seguir con la instalación de herramientas VANETS de tipo *OpenSource* y *freeware* , para poder compararlas más exhaustivamente entre ellas y poder elegir la que más se adecue a nuestras necesidades de simulación vehicular.
- ✓ Modelar con la/s herramienta/s decidida/s escenarios complejos con alto nivel de detalle, como puede ser una ciudad como Cartagena, para obtener resultados sobre el valor añadido de las aplicaciones VANET, sobre todo centrándonos en las de seguridad. Conseguir introducir trazas de tráfico vehicular reales en los simuladores para aumentar el realismo de los resultados. Para estos escenarios ejecutar las

herramientas de simulación en el entorno de supercomputación Ben-Arabí para lograr alcanzar las necesidades de computación requeridas.

Conseguir que el tema VANET tenga su impacto en la comunidad universitaria y para esto una vez alcanzado el nivel suficiente en el área lograr impartir seminarios o cursos a los estudiantes, para que se familiaricen con un tema que dentro de unos años será una realidad.

Bibliografía y Figuras

1. Bigrafia de Cugnot. [En línea] http://www.bookrags.com/biography/nicholas-joseph-cugnot-woi/.

2. *"Vehicular Communication Systems: Enabling Technologies, Applications, and Future Outlook on Intelligent Transportation".* **Panos Papadimitratos, Arnaud de La Fortelle, ,Knut Evenssen, Roberto Brignolo and Stefano Cosenza.** s.l. : IEEE Communications Magazine, November 2009.

3. car-2-car project. [En línea] http://www.car-to-car.org/.

4. *"Communication Patterns in VANETs"* . **Elmar Schoch, Frank Kargl, and Michael Weber, Ulm University, Tim Leinmüller.** s.l. : EEE Communications Magazine, November 2008.

5. *BENEFIT ESTIMATION OF ADVANCED DRIVER ASSISTANCE SYSTEMS FOR CARS DERIVED FROM REAL-LIFE ACCIDENTS.* **Matthias Kuehn, Thomas Hummel, Jenoe Bende.** Paper Number 09-0317, Germany : German Insurers Accident Research.

6. *A Tutorial Survey on Vehicular Ad Hoc Networks.* **Hannes Hartenstein, Kenneth P. Laberteaux.** June, 2008, s.l. : IEEE Communications Magazine.

7. CVIS project. [En línea] http://www.cvisproject.org/.

8. **Hoffman, W.G.** he m-distribution, a General Formula of Intensity Distribution of the Rapid Fading. *Statistical Methods in Radio Wave Propagation.* Oxford, England : s.n., 1960.

9. *Oper. Res. 7, 79.* **Greenberg, H.,.** 1959.

10. *High-Fidelity and Time-Driven Simulation of Large Wireless Networks with Parallel.* **Hyunok Lee, Vahideh Manshadi, and Donald C. Cox,.** s.l. : IEEE Communications Magazine, March 2009.

11. Centro de Supercomputación de Murcia. [En línea] http://www.cesmu.es/.

12. Hewlett Packard. [En línea] http://welcome.hp.com/country/us/en/welcome.html#Product.

13. Proyecto para MPI. [En línea] http://www.mcs.anl.gov/research/projects/mpi/.

14. **Rohit Chandra, Dagum, Leonardo/ Kohr, Dave/ Maydan, Dror/ Mcdonald, Jeff/ Menom, Ramesh.** *Parallel Programming in OpenMP.* s.l. : Elsevier Science Ltd . 1558606718.

15. **Cuenca, Javier.** Curso de Computación Científica en Clusters I. Universidad de Murcia : s.n., 2010.

16. Tutorial para OpenMP. [En línea] https://computing.llnl.gov/tutorials/openMP/.

17. *Performance evaluation of a CCA application for VANETs using IEEE 802.11p.* **Juan-Bautista Tomas-Gabarron, Esteban Egea-Lopez, Joan Garcia-Haro, Rocio Murcia-Hernandez.** Proceedings del ICC, Ciudad del Cabo : s.n., 2010.

18. *Mobility Models for Vehicular Ad Hoc Networks: A Survey and Taxonomy.* **Jerome Harri, Fethi Filali, and Christian Bonnet.** 4, s.l. : IEEE COMMUNICATIONS SURVEYS & TUTORIALS, FOURTH QUARTER 2009, Vol. 11 .

19. *Progressing Towards Realistic Mobility Models in VANETS Simulations.* **Dressler, Christoph Sommer and Falko.** s.l. : IEEE Communications Magazine, 2008, Vol. 2008.

20. **Thomas W. Rioux and Clyde E. Lee, Center for Highway Research, University of Texas at Austin.** *Microscopic Traffic Simulation Package for Isolated Intersections.* 1977.

21. **Cases, Pablo.** Mejora De Una Herramienta De Simulación De Tráfico Rodado Mediante La Integración De Mecanismos De Redes Vehiculares. *Proyecto fin de carrera.UPCT.ETSIT.* 2010.

22. Proyecto SUMO. [En línea] http://sourceforge.net/apps/mediawiki/sumo/index.php?title=Main_Page.

23. Proyecto SmartPath. [En línea] http://www.path.berkeley.edu/PATH/Research/Smartpath/SmartPath/sm.html.

24. FreeSim Simulator. [En línea] http://www.freewaysimulator.com/.

25. *"FreeSim - A V2V and V2R Freeway Traffic Simulator."* . **Millller, Jeffrey, Ellis Horowitz.** s.l. : IEEE 3rd International Workshop on Vehicle-to-Vehicle Communications in conjunction with IEEE 3rd International Intelligent Vehicles Symposium, Istanbul, Turkey, June 2007.

26. STRAW Aqualb. [En línea] http://www.aqualab.cs.northwestern.edu/projects/STRAW/index.php.

27. VanetMobiSim website. [En línea] http://vanet.eurecom.fr/.

28. *VanetMobiSim: generating realistic mobility patterns for VANETs.* **J. Härri, M. Fiore, F. Fethi, and C. Bonnet.** Los Angeles : s.n., 2006.

29. TIGER databes. [En línea] http://www.census.gov/geo/www/tiger/.

30. *A Voronoi-based Mobility Model for Urban Environments.* **H. M. Zimmermann and I. Gruber, "A Voronoi-based Mobility Model.** s.l. : Proc. European Wireless 2005 (EW'05).

31. ns2 website. [En línea] http://www.isi.edu/nsnam/ns/.

32. GloMoSim website. [En línea] http://pcl.cs.ucla.edu/projects/glomosim/.

33. Scalable Network Technologies. Qualnet. . [En línea] http://www.scalablenetworks.com.

34. CityMob website. [En línea] http://www.grc.upv.es/Software/citymob.html.

35. *CityMob: a mobility model pattern generator for VANETS.* **Francisco J. Martinez, Juan-Carlos Cano, Carlos T. Calafate, Pietro Manzoni.** s.l. : IEEE Communications Society , 2008.

36. Proyecto Monarch. [En línea] http://www.monarch.cs.rice.edu/.

37. MITSIMLAB website. [En línea] http://mit.edu/its/mitsimlab.html#mitsimpapers.

38. VISSIM Website. [En línea] http://www.tomfotherby.com/Websites/VISSIM/.

39. MobiTools website. [En línea] http://www.vehicularlab.org/home.do.

40. **Gustavo Marfia, Paolo Lutterottiy, Giovanni Pau.** *MobiTools: An Integrated Toolchain for Mobile Ad Hoc Networks.*

41. Google Maps. [En línea] http://maps.google.com/.

42. Vergilius website. [En línea] http://nrl.cs.ucla.edu/~egiordano/vergiliusJoomla/.

43. CORSIM. [En línea] http://mctrans.ce.ufl.edu/featured/tsis/Version5/corsim.htm.

44. OpenStreetMap website. [En línea] http://www.openstreetmap.org/.

45. Google Earth. [En línea] http://earth.google.es/.

46. TraNS Website. [En línea] http://www.traffic-simulation.de/ger.

47. [En línea] http://www.traffic-simulation.de/ger.

48. Jist-Swans website. [En línea] http://jist.ece.cornell.edu/sw.html.

49. The Georgia Tech Network Simulator (GTNetS). [En línea] http://www.ece.gatech.edu/research/labs/MANIACS/.

50. OMNeT++ website. [En línea] http://www.omnetpp.org/.

51. SNS website. [En línea] http://www.cs.cornell.edu/people/egs/sns/.

52. GrooveSim website. [En línea] http://www.seas.upenn.edu/~rahulm/Research/GrooveNet/.

53. [En línea] http://www.cs.odu.edu/~vanet/Main/Home.

54. *HIGHWAY MOBILITY AND VEHICULAR AD-HOC NETWORKS IN NS-3.* **Hadi Arbabi, Michele C. Weigle.** Vol. Proceedings of the 2010 Winter Simulation Conference.

55. NCTUns website. [En línea] http://nsl10.csie.nctu.edu.tw./.

56. OPENT WEBSITE. [En línea] http://www.opnet.com/.

57. MobiReal website. [En línea] http://www.cs.odu.edu/~vanet/Main/Home.

58. Veins website. [En línea] www7.informatik.uni-erlangen.de/veins/ .

59. *The Open Source Traffic Simulation Package SUMO.* **Daniel Krajzewicz, Michael Bonert,Peter Wagner.** German Aerospace Centre, Institute of Transportation Research : s.n., http://sumo.sourceforge.net/docs.

60. [En línea] http://www.dlr.de/ts.

61. *Mobility Models for Vehicular Ad Hoc Networks:A Survey and Taxonomy.* **Jerôme Härri, Fethi Filali, and Christian Bonnet.** 4, FOURTH QUARTER 2009, IEEE COMMUNICATIONS SURVEYS & TUTORIALS, Vol. 11, págs. 19-41.

62. [En línea] http://sourceforge.net/apps/mediawiki/sumo/index.php?title=SUMO_User_Documentation#Network_Building.

63. *Microscopic Modelling of Traffic Flow: Investigation of Collision Free Vehicle.* **Krauß, S.** s.l. : DLR (Hauptabteilung Mobilität und Systemtechnik), 1998. 1434-8454.

64. [En línea] http://www.isi.edu/nsnam/ns/.

65. QualNet Developer. [En línea] http://www.scalable-networks.com/.

66. Community Site. [En línea] http://www.omnetpp.org/.

67. [En línea] http://prdownloads.sourceforge.net/sumo/sumo-winbin-0.12.0.zip?download.

68. JOSM . [En línea] http://josm.openstreetmap.de/.

69. IDM por Martin Treiber. [En línea] http://vwitme011.vkw.tu-dresden.de/~treiber/MicroApplet/IDM.html.

70. [En línea] http://www7.informatik.uni-erlangen.de/~sommer/omnet/traci/.

71. [En línea] http://sourceforge.net/apps/mediawiki/sumo/index.php?title=Main_Page.

72. [En línea] http://www.omnetpp.org/.

73. *"On the Need for Bidirectional Coupling of Road Traffic Microsimulation and Network Simulation".* **Christoph Sommer, Zheng Yao, Reinhard German and Falko Dressler.** Hong Kong, China : s.n., May de 2008, Proceedings of 9th ACM International Symposium on Mobile Ad Hoc Networking and Computing (Mobihoc 2008): 1st ACM International Workshop on Mobility Models for Networking Research (MobilityModels 2008), págs. 41-48.

74. INET Framework. [En línea] http://inet.omnetpp.org/.

75. *Performance evaluation of a CCA application for VANETs using IEEE 802.11p.* **uan-Bautista Tomas-Gabarron, Esteban Egea-Lopez, Joan Garcia-Haro, Rocio Murcia-Hernandez.** Ciudad del Cabo : ICC Proceedings. , 2010.

76. TRACI INTERFACE WEBSITE. [En línea] http://sourceforge.net/apps/mediawiki/sumo/?title=TraCI#Introduction_to_TraCI.

77. [En línea] http://www.python.org/.

78. [En línea] http://www.activestate.com/activepython/downloads.

79. MPI Tutorial, University of Notre Dame, Laboratory for Scientific Computing. [En línea] http://www.lam-mpi.org/tutorials/nd/.

80. **Clyde E. Lee, Thoma W. Rioux, Charlie R. Copeland.** *THE TEXAS MODEL FOR INTERSECTION TRAFFIC - DEVELOPMENT.* 1977.

81. Thomas Rioux Home Page. [Online] http://www.caee.utexas.edu/prof/rioux/.

82. **Francisco J. Martinez, Chai Keong Toh, Juan-Carlos Cano, Carlos T. Calafate and Pietro Manzoni.** *A survey and comparative study of simulators for vehicular "ad hoc" networks (VANETs).* 2009.

Relación de Figuras

FIGURA 1 ESCENARIO INTERURBANO VANET 9
FIGURA 2 EJEMPLO DE APLICACIÓN DE SEGURIDAD ACTIVA: FRENADA EN CURVA PELIGROSA 13
FIGURA 3 APLICACIÓN VANET PARA MOTOCICLETAS 14
FIGURA 4 APLICACIÓN VANET PARA ACCIDENTES 15
FIGURA 5 APLICACIÓN VANET PARA VEHÍCULOS DE EMERGENCIA 15
FIGURA 6 APLICACIÓN VANET PARA OBRAS 16
FIGURA 7 DATOS DE ACCIDENTES VIALES EN ALEMANIA PARA 2009 17
FIGURA 8 ESQUEMA DE ENTORNO 21
FIGURA 9 ESCENARIO VANET Y SUS COMPONENTES 22
FIGURA 10 ESQUEMA DEL SUPERDOME HP INTEGRITY SX2000 28
FIGURA 11 ESQUEMA PARA EL CLUSTER DE 102 NODOS CON 8 CORES POR CADA NODO DE CÁLCULO 29
FIGURA 12 PROCEDIMIENTO GENÉRICO PARA CONSEGUIR PARALELISMO 31
FIGURA 13 SISTEMA DE MEMORIA COMPARTIDA FRENTE A PASO DE MENSAJES EN MEMORIA DISTRIBUIDA MPI (8) 32
FIGURA 14 RESULTADOS PARA CONFIGURACIONES DE CCA CON DESPLIEGUE DEL 50% PARA 20 VEHÍCULOS 38
FIGURA 15 EVOLUCIÓN HISTÓRICA DE LAS ESTRATEGIAS PARA LOS MODELOS DE MOVILIDAD EN LA INVESTIGACIÓN VANET (19) 41
FIGURA 16 TÉCNICAS PARA LA SIMULACIÓN DE REDES VEHICULARES (19) 42
FIGURA 17 ESQUEMA PARA SIMULADOR AISLADO 43
FIGURA 18 ESQUEMA PARA SIMULADOR INTEGRADO 43
FIGURA 19 ESQUEMA PARA SIMULADOR HÍBRIDO 44
FIGURA 20 COMPONENTES BÁSICOS DE UN SIMULADOR DE TRÁFICO 56
FIGURA 21 DIAGRAMA DE BLOQUES PARA *CAR-FOLLOWING* 58
FIGURA 22 ESPACIO CONTINUO VS DISCRETO 58
FIGURA 23 LÍNEA DE COMANDOS PARA SUMO 60
FIGURA 24 APLICACIONES INCLUIDAS EN SUMO 61
FIGURA 25 ESQUEMA DE LAS APLICACIONES EN SUMO 61
FIGURA 26 PROCEDIMIENTO PARA GENERAR SIMULACIONES 62
FIGURA 27 CAPTURA DE LA GUI PARA SUMO 64
FIGURA 28 RED *GRID* GENERADA 66
FIGURA 29 DETALLE DE INTERSECCIÓN DE RED *GRID* 66
FIGURA 30 DETALLE DE INTERSECCIÓN CONTROLADA POR SEMÁFOROS 67
FIGURA 31 RED *GRID* CON AMPLIACIÓN EN INTERSECCIONES 67
FIGURA 32 RED *SPIDER* DE GENERACIÓN AUTOMÁTICA 68
FIGURA 33 RED *SPIDER* SIN CENTRO 69
FIGURA 34 RED ALEATORIA 70
FIGURA 35 ESQUEMA PARA LA GENERACIÓN DE REDES MANUALES 70
FIGURA 36 SISTEMAS DE COORDENADAS PARA SUMO 71
FIGURA 37 ESQUEMA DE RED MANUAL 72
FIGURA 38 VISUALIZACIÓN DE LA RED EN GUI 79
FIGURA 39 DETALLE DE INTERSECCIÓN CON SEMÁFOROS PARA REDMANUAL.NET 79
FIGURA 40 DETALLE DE ESTRECHAMIENTO DE CARRILES 80
FIGURA 41 ZONA DE ANTIGONES CON *OPENSTREETMAP* 81

FIGURA 42 MENÚ PARA EXPORTACIÓN DE MAPAS 81
FIGURA 43 MAPA ANTIGONES EN SUMO-GUI 82
FIGURA 44 VISTA DE MAPA ANTIGONES EN JOSM 82
FIGURA 45 ESCENARIO PARA GENERAR *TRIPS* VS. *FLOWS* 87
FIGURA 46 MOMENTO DE LA SIMULACIÓN CON LOS 10 VEHÍCULOS EN LA RED 88
FIGURA 47 ESQUEMA DEL *FRAMEWORK* DE SIMULACIÓN. MÁQUINAS DE ESTADO PARA LOS MÓDULOS DE COMUNICACIÓN DE LOS SIMULADORES DE TRÁFICO RODADO Y DE REDES (73) 104
FIGURA 48 INTERCAMBIO DE MENSAJES ENTRE LOS SIMULADORES (73) 104
FIGURA 49 ESTRUCTURA DE MENSAJES TCP 105
FIGURA 50 ARQUITECTURA VEINS CON SUMO-LAUNCHD.PY 108
FIGURA 51 CAR.NED 111
FIGURA 52 ESQUEMA DE *NETWORKLAYER* 116
FIGURA 53 ÁRBOL DE FICHEROS PARA TRACI (OMNET) 125
FIGURA 54 MENSAJES TRACI EN OMNET 128
FIGURA 55 VEINS EN FUNCIONAMIENTO: OMNET Y SUMO ACOPLADAS 129
FIGURA 56 AMPLIACIÓN DE LA INTERSECCIÓN EN SUMO 129
FIGURA 57 AMPLIACIÓN DONDE SE DETECTAN VEHÍCULOS EN SUMO 130
FIGURA 58 NODOS ENVIANDO MENSAJES Y DETALLE DE EVENTOS EN OMNET 130
FIGURA 59 MEDIDA ESCALAR PARA LAS DISTANCIAS RECORRIDAS POR LOS NODOS 131
FIGURA 60 MEDIDA ESCALAR PARA LAS VELOCIDADES MÁXIMAS DE LOS NODOS 131
FIGURA 61 MEDIDA ESCALAR PARA LOS TIEMPOS EN LOS QUE LOS NODOS TERMINAN LA SIMULACIÓN 131
FIGURA 62 MEDIDAS DE VELOCIDAD JUNTO CON VELOCIDAD MEDIA PARA NODO 0 132
FIGURA 63 MEDIDAS DE ACELERACIÓN JUNTO CON ACELERACIÓN MEDIA PARA NODO 0 132
FIGURA 64 MENSAJES CON ID ENVIADOS POR LOS 54 NODOS 133
FIGURA 65 DETALLE DE MENSAJES ENVIADOS POR CADA NODO 133