GatcomSUMO: primeros pasos para simulaciones VANET

P. Pablo Garrido Abenza, Pablo Piñol Peral y Manuel P. Malumbres[1]

Resumen— **Se presenta una aplicación gráfica llamada GatcomSUMO, la cual permite crear de forma sencilla y segura los diferentes archivos de configuración para realizar simulaciones en entornos VANET, especialmente para el caso de utilizar el simulador de tráfico SUMO y OMNeT++ como simulador de red con el paquete Veins. En concreto, esta herramienta permite la generación de escenarios sintéticos (abstractos) o basados en mapas reales, así como la generación de la movilidad de los vehículos de forma manual o automática, todo ello desde el interfaz gráfico de la propia aplicación. Las rutas de los vehículos cumplirán los requerimientos necesarios para ser utilizadas en ambos simuladores, evitando posibles errores en tiempo de ejecución. La herramienta realiza la mayor parte de su trabajo invocando a las distintas utilidades incluidas con SUMO, ahorrando al usuario el tener que escribir complejas órdenes que se ejecutan desde la consola del sistema. Además, GatcomSUMO visualiza la red y rutas, permitiendo su validación antes de ejecutar la simulación. La aplicación dispone también de ciertas utilidades integradas que permiten la conversión de coordenadas entre los sistemas de coordenadas geodésico, métrico (UTM), de SUMO, y de OMNeT++, lo cual permite, entre otras cosas, la colocación de elementos fijos típicos en una red VANET, como sería el caso de los RSU (Road-Side Unit). GatcomSUMO es una herramienta desarrollada en Java y de código abierto, por lo que puede utilizarse libremente en cualquier plataforma en la que también existan los simuladores SUMO y OMNeT++.**

Palabras clave— **VANET, SUMO, OMNeT++, Veins.**

I. Introducción

Una Mobile Ad hoc Network (MANET) es un tipo de red inalámbrica en la que los nodos se pueden comunicar con otros nodos sin necesidad de utilizar ningún tipo de infraestructura o punto de acceso (AP). Por otro lado, una Vehicular Ad hoc Network (VANET) es un tipo concreto de red MANET en la que la red está compuesta por vehículos móviles e infraestructura fija como Road-Side Units (RSU). De este modo, es posible dos tipos principales de comunicación, Vehicle-to-Vehicle (V2V) y Vehicle-to-Infrastructure (V2I). Mientras que los nodos de una MANET se pueden mover libremente en cualquier dirección, los vehículos móviles de una VANET tienen restringido su movimiento a las calles o carreteras. Este tipo de redes tiene diversas aplicaciones, tales como mejorar la seguridad en carretera, implementar sistemas de información de tráfico, etc. Al trabajar con redes inalámbricas, y sobre todo en el caso de las redes VANET, puede ser extremadamente caro realizar bancos de prueba (*testbed*), al tiempo que muy difícil la obtención de resultados y, sobre todo, su repetibilidad. Por tanto, a la hora de evaluar este tipo de redes se suele hacer uso de la técnica de simulación. En el caso particular de VANET se requiere de dos elementos: uno encargado de controlar la movilidad de los vehículos, y otro para controlar la comunicación entre ellos. Esto es así porque en una VANET, los datos recibidos por un vehículo pueden determinar su movilidad. Por ejemplo, cuando un vehículo recibe un mensaje de aviso indicando de un accidente ocurrido en su ruta, podría buscar una ruta alternativa para evitarlo. Aunque existe algún simulador que integra ambas funciones [1] [2], el modelo del canal inalámbrico es incompleto. Otra alternativa sería utilizar dos simuladores diferentes acoplados entre sí, encargándose cada uno de ellos de una funcionalidad concreta. Algunos simuladores de red bien conocidos son Riverbed Modeler [3], ns-2 [4], ns-3 [5], OMNeT++ [6], y JiST/SWANS [7]. En concreto, OMNeT++ es un simulador de código abierto genérico que puede utilizarse en diversos campos: redes cableadas e inalámbricas, redes de colas, o satélites, entre otros. Para ello se requiere de un paquete o *framework* específico; por ejemplo, para poder simular redes inalámbricas existen los *frameworks* MiXiM [8], INET [9], o Castalia [10], los cuales están basados en el simulador OMNeT++.

Por otro lado, hay una serie de simuladores de tráfico como VanetMobiSim [11], VISSIM [12], y SUMO [13]. SUMO (Simulation of Urban MObility) es un simulador de código abierto microscópico que soporta distintos tipos de vehículos (multi-modo). Ofrece un API (Application Programming Interface) llamado TraCI (Traffic Control Interface) [14] que permite a una aplicación externa controlar la simulación y obtener información sobre los objetos móviles simulados en SUMO. Existen distintas implementaciones de TraCI utilizando diversos lenguajes de programación, incluyendo C++, Java, Python o Matlab. Mediante un software intermedio (*middleware*) que implemente dicha interface es posible conectar a uno de los simuladores de red mencionados anteriormente con SUMO. Ejemplos de ello serían TraNS [15], iCS [16], Veins [17], o VSimRTI [18]. En este trabajo nos centramos en Veins (Vehicles in Network Simulation), el cual es un paquete de OMNeT++ que implementa TraCI con objeto de conectar SUMO con algún *framework* de simulación de redes de OMNeT++ como MiXiM [8] o INET [9].

Por tanto, los elementos utilizados son OMNeT++ como simulador de redes y SUMO como simulador de tráfico, acoplados mediante el paquete Veins, el cual también implementa los protocolos de comunicación de red IEEE 1609 (WAVE). Todos ellos son de código abierto, lo cual es un factor clave que facilita la

[1]Dpto. Física y Arquitectura de Computadores, Universidad Miguel Hernández, Elche, e-mail: pgarrido, pablop, mels@umh.es.

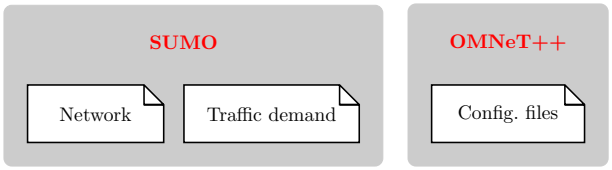

Fig. 1. Grupos de archivos de configuración

reproducibilidad de la investigación [19].

Para poder crear una simulación VANET es necesario escribir ciertos archivos de configuración, tanto para SUMO como para OMNeT++ (Fig. 1): (1) red de carreteras (*Network*), (2) movilidad de los vehículos (*Traffic demand*), y (3) archivos necesarios para la simulación con OMNeT++.

SUMO incluye ciertas herramientas que reciben archivos en formato XML (Extensible Markup Language) como entrada, y generan otros archivos en ese mismo formato. Aunque tanto el uso estas herramientas como el formato de los archivos XML están bien documentados, puede ser complicado hacer manualmente todas las tareas necesarias: preparar la red, rutas de los vehículos, etc. Por otro lado, OMNeT++ también requiere escribir ciertos archivos en diferentes formatos como NED o XML. Todo ello hace que preparar lo necesario para poder lanzar la simulación sea un proceso costoso y propenso a errores, teniendo una curva de aprendizaje alta.

El objetivo principal de GatcomSUMO es automatizar las tareas requeridas para la preparación de los archivos de configuración para poder utilizar SUMO y OMNeT++. Además, la aplicación incorpora una serie de conversores de unidades (potencia de señal, velocidad, temperatura) y herramientas de cálculo (rango de comunicación) útiles para el usuario. Con todo ello, GatcomSUMO permite al investigador centrarse en su investigación en vez de solucionar una serie de problemas habituales cuando se prepara una simulación VANET con estos elementos.

La estructura del artículo es la siguiente. Primero, en la sección II se enumeran brevemente algunas herramientas existentes. En la sección III se presentan brevemente las utilidades que incluye SUMO para la generación de escenarios de tráfico, y cómo GatcomSUMO permite hacer esta misma tarea. En la siguiente sección IV se hace algo similar a la sección anterior, pero dirigido a la generación de demandas de tráfico, esto es, la generación de rutas para la movilidad de los vehículos. A continuación, en la sección V se explica cómo utilizar los archivos de red y de demandas de tráfico de SUMO con el simulador OMNeT++, así como los archivos de configuración necesarios que genera GatcomSUMO. La sección VI presenta una alternativa a la hora de establecer la posición de objetos fijos como los RSU. Como complemento, GatcomSUMO incluye una serie de conversores y calculadora para el rango de comunicación, que se describirán en la sección VII. Por último, la sección VIII finaliza el artículo y enumera algunos de los trabajos futuros.

II. Trabajos relacionados

Como se puede comprobar, es necesario utilizar diversas herramientas y escribir varios archivos de configuración para conseguir lanzar una simulación VANET, y cada paso exige cierta experiencia previa. Se han desarrollado algunos trabajos que proporcionan cierta funcionalidad con objeto de mejorar la usabilidad de SUMO.

Por ejemplo, *eNetEditor* [20] ofrece un interface gráfico que permite la creación de red de carreteras para SUMO. Gran parte de su trabajo lo hace invocando a las distintas utilidades de SUMO, como `netconvert`, `jtrrouter`, `dfrouter` y `duarouter` para generar rutas dinámicamente, es decir, teniendo en cuenta el estado del tráfico, no sólo la distancia más corta.

TrafficModeler [21] es otra herramienta desarrollada en Java que permite construir los archivos de configuración de SUMO desde un interface gráfico sencillo. Permite generar demandas de tráfico basadas en la actividad habitual de la población. Además, también permite modificaciones básicas de la red de carreteras.

Por último, *SUMOPy* [22] es una herramienta desarrollada en Python que puede utilizarse tanto de forma gráfica como ejecutando directamente varios scripts desde la consola del sistema. Su principal objetivo es hacer SUMO más accesible a usuarios sin experiencia de programación. Por ejemplo, permite descargar mapas desde OpenStreetMap, convertirlos al formato requerido por SUMO y extraer polígonos que pueden servir como obstáculos. Al igual que *TrafficModeler*, permite generar demandas de tráfico basadas en la actividad de la población, y algunos otros modelos de generación de tráfico. *SUMOPy* realiza su trabajo mediante diversos scripts en Python, e invocando a las utilidades de SUMO como `od2trips`, `duarouter`, y `jtrrouter`.

A pesar de que estas herramientas permiten superar algunos inconvenientes, la aplicación GatcomSUMO presentada en este trabajo se ajusta mejor al campo de las simulaciones VANET, especialmente cuando se utiliza SUMO conjuntamente con el simulador OMNeT++.

III. Escenario (network)

En esta sección se describe brevemente los archivos de configuración que el simulador de tráfico SUMO necesita para su escenario o red de carreteras, así como las órdenes que sería necesario utilizar si el proceso se hiciera de forma manual con las utilidades que SUMO incorpora. También se muestra cómo hacer la mismas tareas mediante GatcomSUMO.

La Fig. 2 muestra los dos tipos principales de datos con los que SUMO trabaja: la red de carreteras (*Network*) y la demanda de tráfico (*Traffic demand*). La red de carreteras se almacena en archivos de texto con el formato XML (.net.xml), y contienen una lista de calles o tramos (*edges*), una lista de carriles en cada calle (*lanes*), y una lista de nodos que representan las intersecciones entre varias calles (*junctions*).

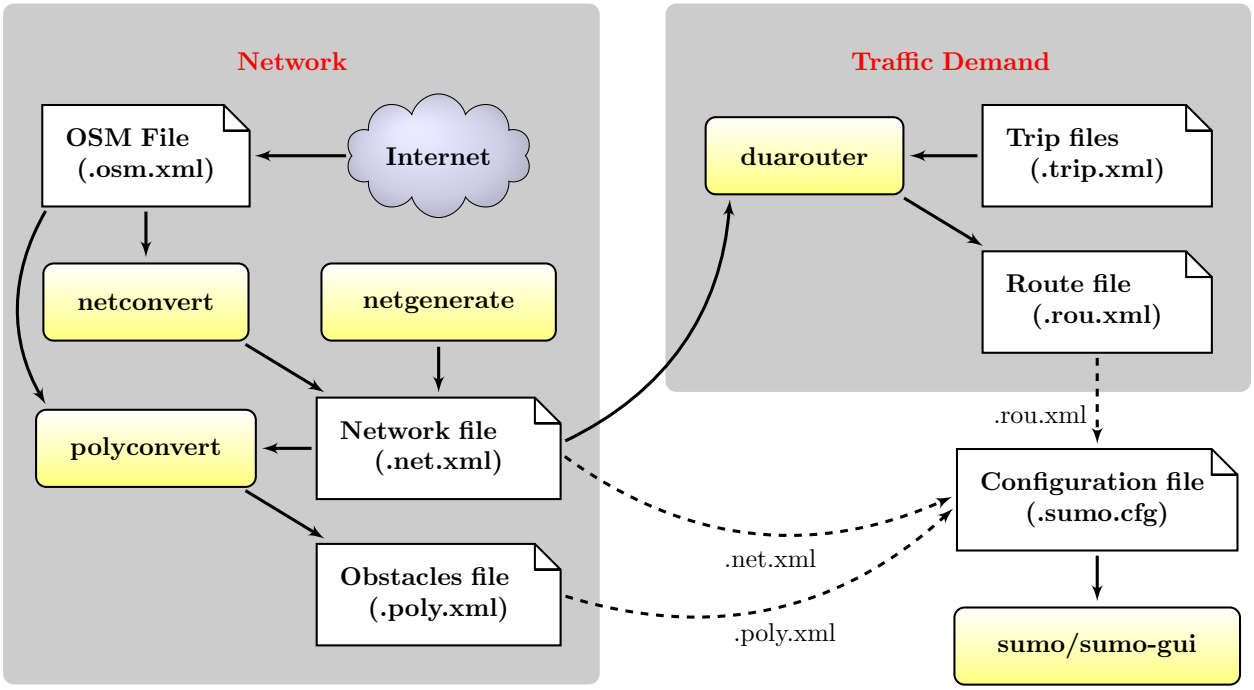

Fig. 2. Flujo de trabajo de SUMO

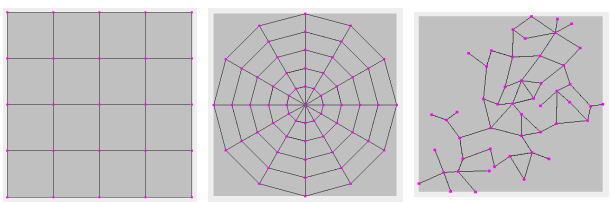

Fig. 3. Redes abstractas: Grid, spider y random

Adicionalmente, pueden incluir otros datos como conexiones y semáforos. Aunque estos archivos pueden modificarse a mano, normalmente son generados por alguna utilidad como `netgenerate` (mapas abstractos) o `netconvert` (mapas reales). También pueden modificarse posteriormente mediante algún editor visual como el propio `netedit` incluido con SUMO.

Mediante la utilidad `netgenerate` el usuario puede generar mapas abstractos o sintéticos con una estructura geométrica en forma de rejilla, tela de araña, o aleatorios (Fig. 3). La siguiente orden genera una red de tipo rejilla con 5x5 calles de longitud 500m:

```
netgenerate --grid true --grid.number 5
            --grid.length 500
            --output-file grid500.net.xml
```

Una ventaja de las redes de tipo rejilla (*grid*) es que el usuario conoce perfectamente la longitud de las calles, lo cual permite validar el rango de comunicación de los vehículos fácilmente. Además, los nombres de los nodos y tramos son fáciles de recordar, facilitando la tarea a la hora de definir rutas de forma manual. Por ejemplo, una calle que vaya desde el nodo `'0/0'` hasta el nodo `'0/1'` se le asigna un identificador de la forma `'0/0to0/1'`, y así sucesivamente.

Por otro lado, cuando se trabaja con escenarios VANET es muy común utilizar mapas reales. Con la utilidad `netconvert` es posible importar diversos formatos de mapas reales, como los de OpenStreetMap (OSM) [23] y PTV VISUM/VISSIM [12]. En este trabajo tan sólo se consideran los de mapas de OSM, puesto que tienen suficientes datos y calidad para nuestros experimentos. Todos los mapas descargados desde OpenStreetMap están basados en el sistema de coordenadas WGS84 (World Geodetic System 1984), el cual es el mismo elipsoide utilizado por el sistema de posicionamiento GPS (Global Positioning System), y muy aproximado al utilizado por el sistema de posicionamiento GLONASS (Global Navigation Satellite System). Actualmente, tanto GPS como GLONASS son sistemas muy utilizados, tanto en vehículos como en smartphones, por lo que es factible utilizar trazas GPS o *tracks* reales sobre mapas de OSM. Los mapas utilizados pueden validarse mediante diversas herramientas como Osmose (OpenStreetMap Oversight Search Engine) [24], y, de forma visual, superponiendo *tracks* GPS reales (archivos .gpx) sobre ellos, mediante el uso de herramientas como JOSM (Java OpenStreetMap) [25] o JGPSTrackEdit [26].

Hay varias formas de descargar mapas desde OSM. La primera es seleccionar un area rectangular directamente en su página web. Otra forma sería hacer uso del API de OSM, que consiste en escribir directamente una URI (Uniform Resource Identifier) en un navegador, indicando las coordenadas del rectángulo a descargar o *bounding box* en forma de grados longitud/latitud de las esquinas suroeste y noreste; existen muchas formas de averiguar dichas coordenadas, desde un mapa en papel hasta diversas utilidades en Internet, como la que se incluye en la propia página de OpenStreetMap. El siguiente ejemplo descarga parte de la ciudad de Elche, generando un fichero denominado 'map.osm.xml':

```
http://api.openstreetmap.org/api/0.6/map?
    bbox=-0.69742,38.26615,-0.67707,38.28310
```

Puede conseguirse lo mismo sin necesidad de utilizar un navegador mediante la utilidad **wget**, la cual recibe la misma URI anterior, y además permite especificar el nombre del archivo de salida:

```
wget "<URI>" -O elche.osm.xml
```

El API OSM limita el número máximo de objetos descargados (p.e., el número máximo de nodos es 50000), y el tamaño máximo del *bounding box* a descargar (0.25 grados en cada dimensión). No obstante, existe otro API que permite descargar áreas mayores denominado OSM Extended API (XAPI).

En cualquier caso, una vez descargado un mapa OSM es necesario procesarlo para que SUMO pueda trabajar con él. Para ello, se necesita la utilidad **netconvert**, que transforma las coordenadas geodésicas WGS84 a coordenadas métricas (UTM), y posteriormente les aplica una traslación (*offset*). Un ejemplo de orden sería la siguiente:

```
netconvert --osm-files    elche.osm.xml
           --output-file  elche.net.xml
```

Cuando se trabaja con escenarios VANET puede ser de interés definir obstáculos como los edificios, puesto que estos afectan al radio de comunicación de los vehículos. En este sentido, los mapas de OSM son también convenientes, ya que incluyen información sobre polígonos, que mediante la utilidad **polyconvert** de SUMO se pueden convertir a un archivo de polígonos (.poly.xml) que Veins identifica como obstáculos. Por ejemplo:

```
polyconvert --osm-files    elche.osm.xml
            --net-file     elche.net.xml
            --output-file  elche.poly.xml
```

Aunque **polyconvert** acepta diversos formatos, no puede generar los polígonos desde redes abstractas, por lo que no podrían utilizarse si se requiere el uso de obstáculos. Como se comentará más adelante, la aplicación GatcomSUMO permite generar directamente, sin utilizar la utilidad **polyconvert**, el archivo de polígonos para redes de tipo rejilla (*grid*), puesto que es muy sencillo generar polígonos en forma de rectángulos.

La aplicación GatcomSUMO permite realizar todas las tareas anteriores de una forma más sencilla desde su interface gráfico, el cual está estructurado en varias solapas principales, coincidiendo con los pasos a seguir. A su vez, cada solapa puede contener otras solapas para proporcionar toda la funcionalidad de forma estructurada (Fig. 4). En primer lugar, es necesario construir o importar la red de carreteras (solapa 'Network'), y después, definir los vehículos y su movilidad (solapa 'Traffic demand'). Tanto la red como las rutas de los vehículos pueden visualizarse con objeto de validarlas (solapa 'Map').

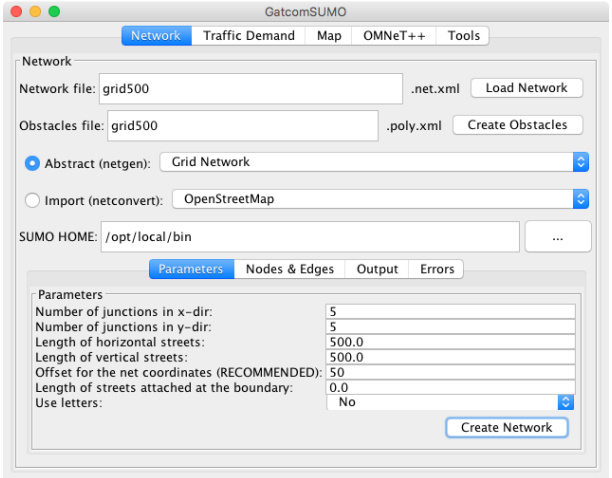

Fig. 4. Solapa 'Network'

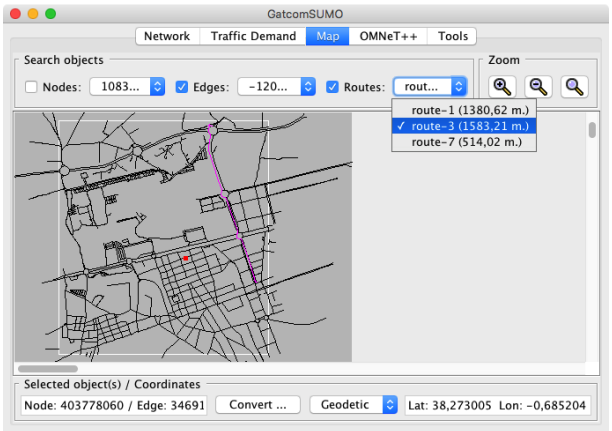

Fig. 5. Mapa de la ciudad de Elche

Desde la solapa 'Network' el usuario puede generar redes abstractas de rejilla, tela de araña o aleatorias (Fig. 3). Al pulsar el botón correspondiente se invoca a la utilidad **netgenerate** con los parámetros adecuados según los valores introducidos (Fig. 4). En este punto se aconseja introducir un desplazamiento (*offset*) con el objeto de que los vehículos no puedan moverse a coordenadas negativas, lo cual generaría un error durante la simulación con OMNeT++.

Desde esta misma solapa, el usuario también puede descargar mapas de OpenStreetMap conociendo las coordenadas (geográficas o UTM) del rectángulo de la zona a descargar o *bounding box*; la aplicación se conectará con el servidor de OSM y descargará dicha zona en formato XML (.osm.xml), o también en algún formato gráfico como PNG, JPEG, SVG, or PDF. Una vez descargado, basta con pulsar el botón correspondiente para que el mapa se convierta al formato adecuado para SUMO, invocando a la utilidad **netconvert** en segundo plano.

Una vez creada la red se puede generar el archivo de obstáculos (.poly.xml) desde la aplicación. Si se trata de una red descargada desde OSM se invocará a **polyconvert** como se explicó anteriormente; en cambio, si se trata de una red abstracta de tipo rejilla, es GatcomSUMO quien genera dicho archivo de forma directa, creando rectángulos un poco más

pequeños que los formados por la intersección de las carreteras, cuyo tamaño es bien conocido (Fig. 12).

Finalmente, la red de carreteras puede cargarse en la aplicación y visualizarse de forma gráfica en la solapa 'Map' (Fig. 5), donde el usuario puede identificar los distintos nodos y tramos pulsando sobre ellos, o localizarlos en el mapa al seleccionarlos de la lista correspondiente.

IV. Demandas de tráfico (Traffic demands)

SUMO es un simulador de tráfico microscópico, lo que quiere decir que cada vehículo individual sigue su propia ruta. Las rutas se expresan como una lista de tramos (*edges*), que junto con la definición de los vehículos se almacenan en un archivo de rutas (.rou.xml). SUMO permite también simulaciones multi-modo, esto es, el uso de distintos tipos de vehículos, con distintas características físicas (longitud, velocidad máxima, aceleración, etc.), definidos también en el archivo de rutas, y a todo esto es a lo que se le denomina demanda de tráfico.

Los archivos de rutas (.rou.xml) se pueden escribir de forma manual, aunque también se puede hacer uso de diversas utilidades como `duarouter` (indicando el tramo de origen y destino), `jtrrouter` (indicando probabilidades de giro en las intersecciones), y `od2trips` (indicando matrices Origen/Destino con el formato PTV VISUM/VISSIM). Las rutas también se pueden generar de forma aleatoria mediante scripts como `randomTrips.py` o `dua-iterate.py`. A modo de ejemplo, la utilidad `duarouter` genera rutas indicando sólo el tramo origen y destino, y de forma opcional, algún tramo intermedio por el que debe pasar la ruta. A esto se le conoce como 'viaje' o *trip*. La utilidad `duarouter` recibe como entrada el archivo de red (.net.xml) y el archivo de definición del *trip* (p.e. mytrip.xml), y busca el camino según un algoritmo concreto (por defecto `'dijkstra'`); como resultado se genera un archivo XML con la ruta encontrada (p.e. myroute.xml):

```
duarouter --net-file    elche.net.xml
          --trip-files  mytrip.xml
          --output-file myroute.xml
```

A pesar de las diversas herramientas disponibles, es necesario tener en consideración lo siguiente. Primero, que las rutas deben estar conectadas, es decir, cada tramo (*edge*) especificado debe ser adyacente al anterior. Segundo, que las rutas deben tener al menos dos tramos, incluso si un vehículo va a estar parado. Tercero, que la ruta asignada a cada vehículo haya sido previamente definida. Por último, que los vehículos deben estar ordenados según su instante de salida, de lo contrario algunos de ellos podrían ignorarse. Aunque existen scripts que comprueban algunas de las situaciones anteriores como `routecheck.py`, hay otras situaciones en las que al ejecutar una simulación con OMNeT++ pueden generarse errores en tiempo de ejecución, como por ejemplo, cuando un vehículo sale de la zona de trabajo (*playground*) o se mueve a una coordenada negativa. Estas condiciones

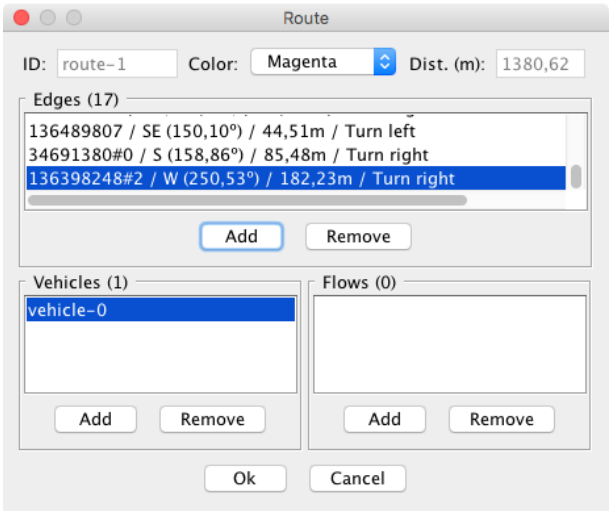

Fig. 6. Diálogo de creación de ruta

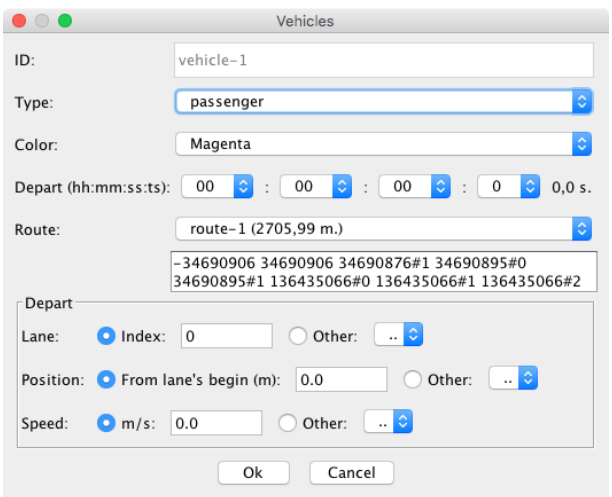

Fig. 7. Diálogo de creación de vehiculos

son difíciles de comprobar sin la existencia de una utilidad como GatcomSUMO.

Como puede apreciarse, la generación de rutas requiere el conocimiento de diversas herramientas, a las cuales habría que invocar también de forma reiterada, y con la posibilidad de obtener errores posteriormente durante la simulación mediante OMNeT++. GatcomSUMO puede generar fácilmente archivos de rutas desde la solapa 'Traffic demand', incluyendo la definición de tipos de vehículos, las rutas propiamente dichas cumpliendo diversas condiciones, y los vehículos o flujos de vehículos a las que se les asigna cada ruta, todo ello desde el interface gráfico y sin conocimiento de las herramientas anteriormente mencionadas. La aplicación invoca a las utilidades anteriores en algunos casos, aunque también realiza otros procesados internamente.

Para empezar, la aplicación incorpora una serie de tipos de vehículos predefinidos, como `passenger`, `emergency`, `taxi`, o `bus`, entre otros. No obstante, el usuario puede modificar sus parámetros o añadir otros tipos nuevos. A la hora de escribir las distintas rutas de los vehículos se especifica la lista de tramos (*edges*) que las componen. Cuando se utilizan

mapas reales de OpenStreetMap es muy frecuente que existan tramos muy cortos, siendo muy difícil identificarlos si sólo se utiliza la ventana gráfica de SUMO (sumo-gui), haciendo que se escriban rutas inconexas. La aplicación permite crear rutas de forma manual seleccionando los tramos desde una lista, con la particularidad de que sólo aparecerán en ella los tramos adyacentes al último tramo añadido (normalmente sólo 2 o 3), e informando de su dirección (ángulo respecto el norte), haciendo el proceso más rápido y, sobre todo, evitando la posibilidad de definir rutas inconexas (Fig. 6). Una vez creadas las rutas, es posible asignarlas a uno o más vehículos, tanto desde la ventana de definición de la ruta, como desde la ventana de definición de cada uno de dichos vehículos (Fig. 7). El usuario puede en todo momento visualizar las rutas creadas en la solapa 'Map' con el color asociado, para su validación antes de ejecutar la simulación con OMNeT++.

Aunque lo anterior facilita mucho la definición de rutas, ese procedimiento sólo es factible cuando hay pocos vehículos y las rutas que seguirán son conocidas de antemano. Como se ha comentado antes, otra posibilidad es el uso de 'viajes' o *trips*, desde la solapa 'Trips' dentro de 'Traffic demand'. Los tramos origen y destino pueden seleccionarse de una lista, como en el caso anterior, pero también pueden ser elegidos aleatoriamente. De forma opcional, el usuario puede especificar también uno o más tramos intermedios (Fig. 8). A partir de los datos anteriores, GatcomSUMO genera el archivo de *trip* necesario (mytrip.xml) e invoca a la utilidad `duarouter` con los parámetros necesarios. Como resultado se genera un archivo con la ruta encontrada (myroute.xml), que el programa puede incorporar si se desea a la lista de rutas creadas.

En el caso de necesitar un mayor número de rutas, la aplicación permite especificar un número de iteraciones, y el proceso anterior se repetirá tantas veces como se pida. Esto trabaja de forma parecida a como lo hace el script `randomTrips.py`, pero con las siguientes mejoras. Primero, el programa detecta en cada iteración si se puede encontrar una ruta válida entre origen y destino, ya que puede darse el caso de que se haya elegido aleatoriamente un tramo aislado como origen o destino; en estos casos, la aplicación lo reintentará de nuevo. Segundo, el programa permite definir una serie de filtros o condiciones que deben cumplir las rutas generadas: (1) con una longitud menor o mayor que un valor especificado, (2) evitando que la ruta salga de cierta zona (rectángulo), (3) que la ruta sea válida para un cierto tipo de vehículo, ya que, por ejemplo, podría seleccionarse un tramo peatonal por el que no podría circular un vehículo. Al final del proceso, el usuario podrá visualizar los datos de las rutas generadas de forma temporal, y añadirlas definitivamente a la tabla de rutas generadas (Fig. 9), todas o sólo las seleccionadas, pudiendo aplicar otros filtros adicionales para reducir su número.

La aplicación también permite crear vehículos o

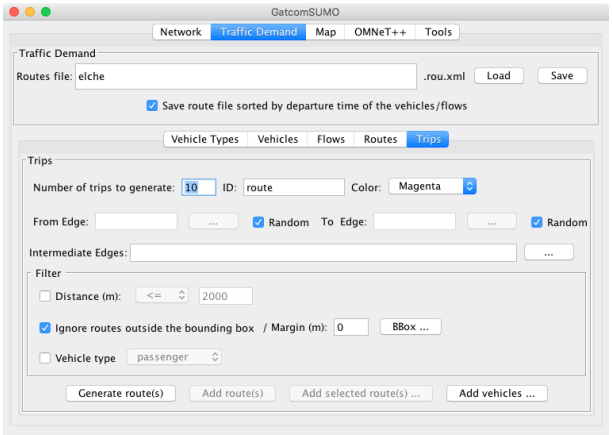

Fig. 8. Diálogo de definición de 'Trips'

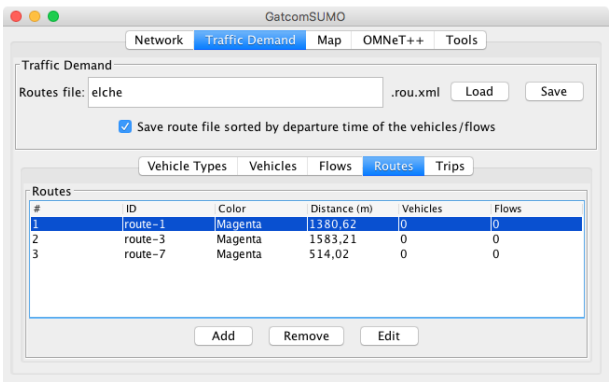

Fig. 9. Diálogo de rutas creadas

flujos de vehículos de forma automática para cada una de las rutas generadas, variando su identificador y su instante de salida mediante un incremento concreto respecto al vehículo anterior.

Como se ha comentado anteriormente, las rutas generadas se pueden visualizar en la solapa 'Map' para comprobar su coherencia (Fig. 5). Por ejemplo, si se había seleccionado el filtro de evitar que las rutas salgan de una cierta zona, se podrá comprobar que ninguna de las rutas de la lista sale fuera del rectángulo de color blanco, el cual muestra el último rectángulo utilizado como límite. Por defecto, el rectángulo es el *bounding box* especificado al descargar un mapa de OSM, lo cual evita que los vehículos salgan del *playground* utilizado por OMNeT++, ya que eso generaría un error durante la simulación.

Sobre el mismo mapa es posible también calcular la distancia entre dos puntos, lo cual puede ser útil para medir la longitud de alguna calle o validar el rango de comunicación en una red inalámbrica.

Por último, todos los datos anteriores se almacenarán en un único archivo de rutas (.rou.xml), que se podrá pre-visualizar en la ventana gráfica de SUMO, o utilizarse para ejecutar una simulación con OMNeT++. Otro aspecto destacable es que los vehículos y flujos de vehículos se escriben en el archivo ordenados según su instante de salida, evitando que se ignoren algunos vehículos. Como se puede comprobar, el proceso de escribir un archivo de rutas se ha simplificado enormemente.

V. Archivos de OMNeT++

Como se explicó anteriormente, además del simulador de tráfico se requiere de un simulador de red para poder realizar una simulación VANET. Esta sección explica brevemente cómo se utilizan los archivos de SUMO generados anteriormente desde el simulador OMNeT++. Como se verá a continuación, GatcomSUMO también permite generar los archivos que son necesarios.

El flujo de datos (⟶) y referencias entre archivos (- - ➝) se muestra en la Fig. 10, utilizando la ciudad de Elche como ejemplo. Los tres archivos generados con SUMO (.net.xml, .rou.xml, y .poly.xml) necesitan referenciarse en un archivo de configuración llamado 'elche.sumo.cfg', que podría utilizarse desde la ventana gráfica de SUMO. Los 4 archivos anteriores necesitan ser referenciados desde otro archivo llamado 'elche.launchd.xml' utilizado por el simulador OMNeT++. Asimismo, se requiere que este archivo sea referenciado desde el archivo de configuración omnetpp.ini para poder ejecutar simulaciones. En caso de utilizar obstáculos en el escenario es necesario también un archivo (p.e. config.xml) que defina la atenuación de la señal para cada tipo de obstáculo definido en el archivo de obstáculos (.poly.xml), tal como `building`, `parking`, `industrial`, etc., ya que de lo contrario los polígonos no se verán en la ventana de OMNeT++ durante la simulación. Mediante la solapa 'OMNeT++' de GatcomSUMO, el usuario puede generar fácilmente los ficheros necesarios según las opciones seleccionadas: elche.sumo.cfg, elche.launchd.xml, config.xml, y parte del archivo omnetpp.ini. Respecto este último, se incluyen los parámetros relacionados con el objeto `TraCIScenarioManager` y el tamaño recomendado para la zona de trabajo (*playground*). Fragmentos de los mismos pueden verse en la Fig. 11. Además de los archivos mostrados, se genera algún otro, como un script shell `run.sh`, que puede ser útil para ejecutar simulaciones tanto de forma secuencial como en paralelo si se dispone de un sistema multi-core o multi-procesador.

VI. Colocando Road-Side Units (RSU)

A la hora de colocar objetos fijos en el escenario como antenas o Road-Side Units (RSU) hay dos opciones: (1) definir vehículos sin movilidad en SUMO, o (2) especificar su posición desde el archivo de configuración de OMNeT++ (omnetpp.ini). En el primer caso, se trata de definir un tipo de vehículo cuya velocidad máxima sea 0, y un vehículo que sigue una ruta de dos tramos (el mínimo posible). La posición del RSU será el inicio del primer tramo especificado, por lo que es complicado ajustar su posición exacta. Además, otro inconveniente es que los RSU así colocados se considerarían como cualquier otro vehículo, por lo que su presencia podría afectar al resto de vehículos y provocar un atasco (Fig. 12).

La segunda alternativa evita estos problemas, pues se trata de especificar la posición de forma exacta de un RSU en las propiedades `x` e `y` del objeto

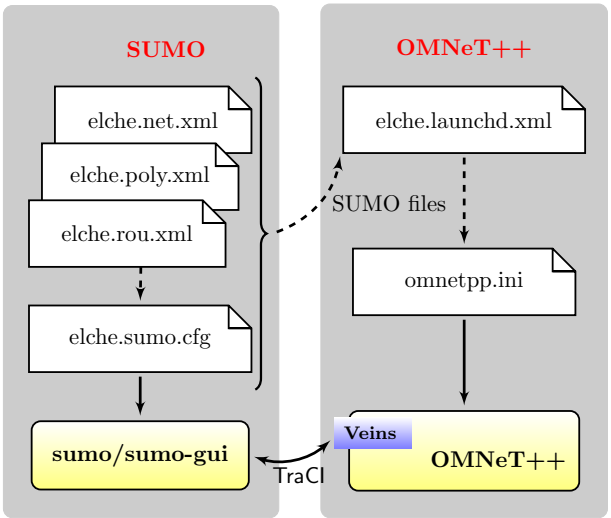

Fig. 10. Flujo de trabajo de OMNeT++

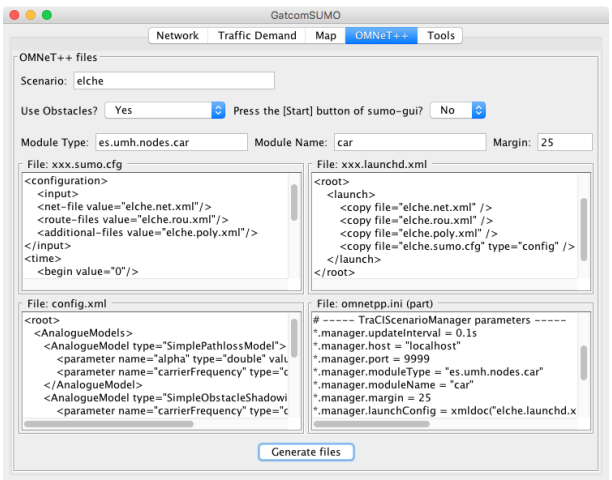

Fig. 11. Archivos de simulacion de OMNeT++

`BaseMobility` de Veins. Puesto que SUMO desconoce dicho objeto no afectará a la movilidad del resto de vehículos. Por ejemplo, el siguiente fragmento establece la posición `(1500,1500)` para el objeto `rsu[1]` de OMNeT++ en el archivo omnetpp.ini:

```
*.rsu[1].mobility.x = 1500
*.rsu[1].mobility.y = 1500
```

Sin embargo, el problema es que se requiere conocer las coordenadas en el sistema de coordenadas de OMNeT++, el cual es diferente del de SUMO. Por tanto, no pueden utilizarse las coordenadas mostradas en la ventana gráfica de SUMO (sumo-gui), ni coordenadas reales (UTM o geodésicas). GatcomSUMO permite hacer las conversiones necesarias a partir de los datos incluidos en el archivo de red (.net.xml), simplemente pulsando en cualquier punto del mapa o tecleando en un conversor. Como pequeño inconveniente, los objetos colocados de esta manera no aparecen en la ventana gráfica de SUMO, pero esto se puede solventar fácilmente escribiendo algo de código en el modelo de OMNeT++ para que se cree un punto de interés (POI) en SUMO cuando el objeto RSU reciba su primer auto-mensaje (Fig. 12).

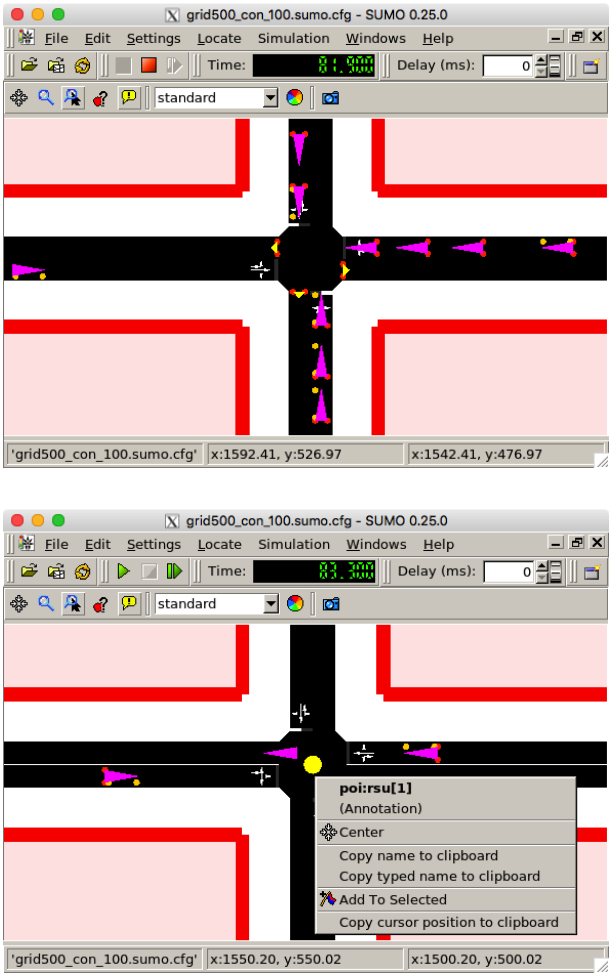

Fig. 12. RSUs desde SUMO (arriba) vs. OMNeT++ (abajo)

VII. Utilidades

Por último, la aplicación incluye una serie de utilidades en la solapa 'Tools', incluyendo conversores entre unidades de distintas magnitudes: potencia de la señal, velocidad, o temperatura (Fig. 13). Incorpora una calculadora de la atenuación de la señal según los modelos Free-Space Path Loss (FSPL) y Two-Ray, utilizada para calcular la potencia de recepción y el rango de comunicación en las comunicaciones inalámbricas. Esta herramienta permite también obtener los valores de la potencia de transmisión P_{tx} y recepción P_{rx} para un rango de comunicación deseado, pudiendo el usuario elegir cualquiera de las diferentes soluciones encontradas. El resultado se muestra de forma textual, de tal manera que se puede simplemente copiar y pegar en el archivo omnetpp.ini (Fig. 14).

VIII. Conclusión y Trabajos futuros

Se ha presentado una aplicación llamada GatcomSUMO que permite generar de forma sencilla los archivos de configuración necesarios para lanzar simulaciones VANET utilizando los simuladores SUMO, OMNeT++, y el framework Veins. La aplicación actúa como un *front-end*, e invoca a las utilidades de SUMO para hacer parte de su trabajo. Las rutas generadas son comprobadas para su correcto funcionamiento durante la simulación con OMNeT++.

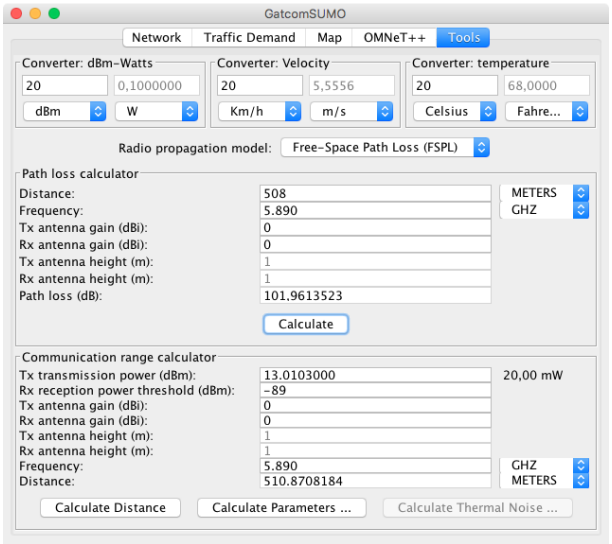

Fig. 13. Solapa 'Tools'

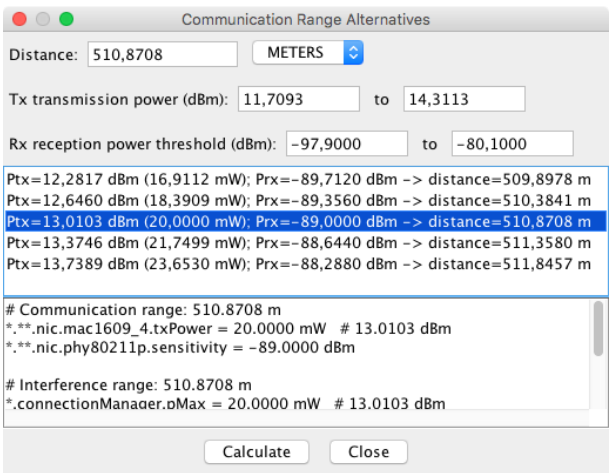

Fig. 14. Diálogo de rango de comunicación

Como trabajos futuros se pretende mejorar la visualización del mapa, integrar modelos adicionales para generación de rutas, incluir otros modelos de propagación de la señal, así como ampliar soporte para otros *frameworks* de OMNeT++ para redes inalámbricas como INET a la hora de calcular los valores para el rango de comunicación.

GatcomSUMO se puede descargar desde la página web del grupo Gatcom (`http://atc.umh.es/gatcom/soft/gatcomsumo`).

Agradecimientos

Este trabajo ha sido financiado parcialmente mediante el proyecto TIN2015-66972-C5-4-R y cofinanciado por fondos FEDER (MINECO/FEDER/UE). Los autores también les gustaría agradecer a los desarrolladores del software de código abierto utilizado en nuestra investigación, como SUMO (`dlr.de/ts/sumo`), OMNeT++ (`https://omnetpp.org`), y Veins (`http://veins.car2x.org`). También nos gustaría expresar nuestra gratitud a los desarrolladores del lenguaje Java y el compilador Excelsior JET (`https://www.excelsiorjet.com`).

Referencias

[1] Rahul Mangharam, Daniel S. Weller, Daniel D. Stancil, Ragunathan Rajkumar, and Jayendra S. Parikh, "Groovesim: A topography-accurate simulator for geographic routing in vehicular networks," in *Proceedings of the 2Nd ACM International Workshop on Vehicular Ad Hoc Networks*, New York, NY, USA, 2005, VANET '05, pp. 59–68, ACM.

[2] Xiaolan Tang, Juhua Pu, Ke Cao, Yi Zhang, and Zhang Xiong, "Integrated extensible simulation platform for vehicular sensor networks in smart cities," *International Journal of Distributed Sensor Networks*, vol. 8, no. 12, pp. 860415, 2012.

[3] Riverbed Technology, "Riverbed Modeler," http://www.riverbed.com/products/performance-management-control/network-performance-management/network-simulation.html, [Accessed April 20, 2017].

[4] ns-2, "The Network Simulator," http://www.isi.edu/nsnam/ns/, [Accessed April 20, 2017].

[5] George F. Riley and Thomas R. Henderson, *The ns-3 Network Simulator*, pp. 15–34, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[6] András Varga and Rudolf Hornig, "An overview of the OMNeT++ simulation environment," in *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, ICST, Brussels, Belgium, Belgium, 2008, Simutools '08, pp. 60:1–60:10, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[7] R. Barr, Z. J. Hass, and R. van Renesse, "JiST/SWANS: Java in Simulation Time / Scalable Wireless Ad hoc Network Simulator," http://jist.ece.cornell.edu, [Accessed April 20, 2017].

[8] Karl Wessel, Michael Swigulski, Andreas Köpke, and Daniel Willkomm, "MiXiM: the physical layer an architecture overview," in *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, ICST, Brussels, Belgium, Belgium, 2009, Simutools '09, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[9] András Varga et al., "INET Framework," https://inet.omnetpp.org, [Accessed April 20, 2017].

[10] Dimosthenis Pediaditakis, Yuri Tselishchev, and Athanassios Boulis, "Performance and scalability evaluation of the castalia wireless sensor network simulator," in *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2010, p. 53, [Accessed April 20, 2017].

[11] Jérôme Härri, Marco Fiore, Fethi Filali, and Christian Bonnet, "Vehicular mobility simulation with vanetmobisim," *Simulation*, vol. 87, no. 4, pp. 275–300, Apr. 2011.

[12] Nicholas E. Lownes and Randy B. Machemehl, "VISSIM: A Multi-parameter Sensitivity Analysis," in *Proceedings of the 38th Conference on Winter Simulation*. 2006, WSC '06, pp. 1406–1413, Winter Simulation Conference.

[13] Daniel Krajzewicz, Jakob Erdmann, Michael Behrisch, and Laura Bieker, "Recent development and applications of SUMO - Simulation of Urban MObility," *International Journal On Advances in Systems and Measurements*, vol. 5, no. 3&4, pp. 128–138, December 2012.

[14] Axel Wegener, Michał Piórkowski, Maxim Raya, Horst Hellbrück, Stefan Fischer, and Jean-Pierre Hubaux, "Traci: An interface for coupling road traffic and network simulators," in *Proceedings of the 11th Communications and Networking Simulation Symposium*, New York, NY, USA, 2008, CNS '08, pp. 155–163, ACM.

[15] M. Piórkowski, M. Raya, A. Lezama Lugo, P. Papadimitratos, M. Grossglauser, and J.-P. Hubaux, "TraNS: Realistic Joint Traffic and Network Simulator for VANETs," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 12, no. 1, pp. 31–33, Jan. 2008.

[16] Daniel Krajzewicz, Laura Bieker, Jérôme Härri, and Robbin Blokpoel, "Simulation of V2X applications with the iTETRIS system," in *TRA 2012, Transport Research Arena, April 23-26, 2012, Athens, Greece / Also published in Procedia - Social and Behavorial Sciences, Volume 48, 2012*, Athens, GRÈCE, 04 2012.

[17] Christoph Sommer, Reinhard German, and Falko Dressler, "Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis," *IEEE Transactions on Mobile Computing*, vol. 10, no. 1, pp. 3–15, January 2011.

[18] Björn Schünemann, "V2X simulation runtime infrastructure VSimRTI: An assessment tool to design smart traffic management systems," *Computer Networks*, vol. 55, no. 14, pp. 3189 – 3198, 2011, Deploying vehicle-2-x communication.

[19] Adelinde M. Uhrmacher, Sally Brailsford, Jason Liu, Markus Rabe, and Andreas Tolk, "Reproducible Research in Discrete Event Simulation: A Must or Rather a Maybe?," in *Proceedings of the 2016 Winter Simulation Conference*, Piscataway, NJ, USA, 2016, WSC '16, pp. 1301–1315, IEEE Press.

[20] Tamás Kurczveil and Pablo Alvarez López, "eNetEditor: Rapid prototyping urban traffic scenarios for SUMO and evaluating their energy consumption," in *SUMO 2015 - Intermodal Simulation for Intermodal Transport*. May 2015, pp. 137–160, Deutsches Zentrum für Luft und Raumfahrt e.V.

[21] L. G. Papaleondiou and M. D. Dikaiakos, "Trafficmodeler: A graphical tool for programming microscopic traffic simulators through high-level abstractions," in *VTC Spring 2009 - IEEE 69th Vehicular Technology Conference*, April 2009, pp. 1–5.

[22] Joerg Schweizer, *SUMOPy: An Advanced Simulation Suite for SUMO*, Lecture Notes in Computer Science (LNCS). Springer-Verlag Berlin Heidelberg, November 2014, [Accessed April 20, 2017].

[23] Mordechai (Muki) Haklay and Patrick Weber, "Openstreetmap: User-generated street maps," *IEEE Pervasive Computing*, vol. 7, no. 4, pp. 12–18, Oct. 2008.

[24] Etienne Chové, Jocelyn Jaubert, Frédéric Rodrigo, et al., "Osmose," http://osmose.openstreetmap.fr/es/map/, [Accessed April 20, 2017].

[25] Immanuel Scholz, Dirk Stöcker, et al., "JOSM," https://josm.openstreetmap.de/, [Accessed April 20, 2017].

[26] Hubert Lutnik, "JGPSTrackEdit," https://sourceforge.net/projects/jgpstrackedit/, [Accessed April 20, 2017].