VDSF-VN: Herramienta para la Simulación de Transmisión de Vídeo en Redes Vehiculares

P. Pablo Garrido Abenza, Manuel P. Malumbres, Pablo Piñol Peral, O. López Granado[1]

Resumen— **Se presenta la plataforma de simulación *Video Delivery Simulation Framework over Vehicular Networks* (VDSF-VN), cuyo objetivo es facilitar y optimizar el desarrollo de experimentos mediante simulación relacionados con la transmisión de vídeo en redes vehiculares. La herramienta automatiza y simplifica la realización de las distintas tareas involucradas en experimentos de este tipo, las cuales requieren el uso de una gran cantidad de elementos *software* independientes, tales como simuladores de tráfico de vehículos, simuladores y extensiones para comunicación en red, codificadores y decodificadores de vídeo, conversores, *software* de generación de gráficos, hojas de cálculo, etc. Sin una herramienta como VDSF-VN, el investigador debe esperar a que finalice una tarea, a veces de larga duración, para lanzar manualmente la siguiente de forma secuencial. Todo ello motivó el desarrollo de VDSF-VN, con la intención de realizar todas las tareas necesarias de una forma integrada y más eficiente. La herramienta hace uso de todas las herramientas mencionadas, pero gestionadas desde dos aplicaciones multi-plataforma que ofrecen un interfaz gráfico sencillo: *GatcomSUMO* y *GatcomVideo*. La primera de ellas se ha desarrollado con el objetivo de generar los escenarios de red y movilidad de vehículos necesarios para el simulador de tráfico SUMO, mientras que la segunda tiene que ver con el proceso completo de transmisión de vídeo, en concreto, la codificación y decodificación de vídeo, la ejecución de las simulaciones y el procesado posterior de los resultados para la generación automática de gráficos personalizados, tablas de datos e informes. La herramienta reduce, por tanto, la curva de aprendizaje necesaria para realizar este tipo de experimentos de principio a fin, al tiempo que realiza las tareas de forma más eficiente, pues muchas de ellas se pueden realizar en paralelo, o lanzarse sin la intervención del usuario.**

Palabras clave— **Simulación, Redes Vehiculares, Vídeo, HEVC.**

I. Introducción

UNA red vehicular o *Vehicular Ad-hoc Network* (VANET) es una red inalámbrica en la que los nodos móviles son vehículos que se mueven a lo largo de la red de carreteras o calles de una ciudad, pudiendo comunicarse entre ellos y con la infraestructura fija, o *Road-Side Units* (RSU). Por tanto, existen varios tipos de comunicación: Vehicle-to-Vehicle (V2V), Vehicle-to-Infrastructure (V2I), o Vehicle-to-Everything (V2X). Este tipo de redes tiene muchas aplicaciones en el campo de los *Intelligent Transportation Systems* (ITS), tales como sistemas de información de tráfico para evitar atascos, aumentar la seguridad en carretera para evitar accidentes, acceso a Internet y otras aplicaciones de entretenimiento (*infotainment*). En concreto, la transmisión de vídeo puede tener muchas aplicaciones, tales como la videovigilancia o la transmisión de vídeo dependiente del contexto.

Sin embargo, las redes inalámbricas experimentan diversos problemas, tales como un medio compartido con un ancho de banda limitado, donde las transmisiones de los diferentes nodos pueden colisionar, así como otros fenómenos como la atenuación de la señal con la distancia (*path loss*), la presencia de obstáculos (*shadowing*), y efectos de refracción y reflexión (*multipath*). Además, en el caso particular de las VANETs, el tiempo de comunicación entre los vehículos es muy reducido debido a su alta movilidad, que provoca cambios continuos en la topología de red. Todo ello hace que se produzca una cantidad elevada de paquetes perdidos, especialmente en la transmisión de vídeo debido al alto ancho de banda necesario y a la limitación del retardo máximo admisible en las aplicaciones en tiempo real.

El uso de técnicas de compresión de vídeo permite reducir la cantidad de datos requeridos, tanto para su almacenamiento como para su transmisión. Se han desarrollado diversos estándares en los últimos años, como el *High Efficiency Video Coding* (HEVC) [1], que duplica la tasa de compresión de su predecesor, el *Advanced Video Coding* (AVC) [2], también conocido como H.264. Aun así, la calidad del vídeo percibida por el usuario puede verse muy afectada por los paquetes perdidos durante su transmisión.

Estudiar una red vehicular en un escenario real, o mediante experimentación con bancos de pruebas (*testbed*), tiene un elevado coste y el análisis de diferentes condiciones o la recogida de estadísticas resultan muy complicados. Por ello, la técnica más utilizada para experimentar con redes vehiculares es mediante simulación, permitiendo, además, que los experimentos sean reproducibles [3]. Sin embargo, para la simulación de una red vehicular en la que se transmiten secuencias de vídeo, se requieren muchos elementos, tales como un simulador de tráfico, un simulador de comunicaciones inalámbricas, escenarios (sintéticos o mapas reales), codificadores y decodificadores de vídeo, *software* para la generación de gráficos, etc. Los elementos utilizados en trabajos previos son los siguientes: SUMO (Simulation of Urban MObility) [4] como simulador de tráfico, OMNeT++ [5] junto con el paquete Veins (VEhicles In Network Simulation) [6] como simulador de red, mapas reales descargados desde OpenStreetMap [7], y el codificador y decodificador de vídeo HM (*HEVC Test Model*) [8], entre otros.

La curva de aprendizaje necesaria para cada uno de estos elementos puede ser elevada, y la preparación de los escenarios de simulación puede requerir experiencia y gran cantidad de tiempo hasta su co-

[1]Dpto. de Ingeniería de Computadores, Universidad Miguel Hernández, Elche (Alicante), e-mail: {pgarrido, mels, pablop, otoniel}@umh.es.

rrecta configuración. Por otro lado, la codificación de las secuencias de vídeo a transmitir, utilizando distintas combinaciones de parámetros de configuración, es un proceso largo. Se requiere la interacción constante del investigador para lanzar las tareas necesarias una tras otra según van finalizando, y el procesamiento de gran cantidad de estadísticas recogidas como resultado de la simulación y evaluación de las secuencias de vídeo recibidas. Aunque pueden escribirse algunos archivos por lotes o *scripts* para automatizar ciertas tareas, se suelen escribir en lenguajes que no son multiplataforma, dificultando la reproducibilidad de los experimentos. Todo lo anterior hace que la realización de un experimento de este tipo sea ardua y muy propensa a errores.

La contribución de este trabajo consiste en una plataforma de simulación denominada *Video Delivery Simulation Framework over Vehicular Networks* (VDSF-VN), que cubre todo el proceso de simulación, desde la creación de los escenarios o la movilidad de los vehículos, pasando por la codificación de las secuencias de vídeo, y terminando con la generación y visualización de forma gráfica de los resultados. Todo el proceso se controla desde un interfaz gráfico que permite lanzar las tareas en paralelo siempre que sea posible. Tanto el interfaz gráfico realizado como los elementos necesarios (simuladores, etc.) son multiplataforma, por lo que se facilita su uso por otros investigadores. La herramienta ya ha sido validada en diversos trabajos y se ha mostrado muy útil y eficiente.

El artículo está organizado de la siguiente manera. En la sección II se hace una revisión de las herramientas o *frameworks* existentes cuyo objetivo es la simulación y evaluación de la transmisión de vídeo en redes vehiculares. La sección III describe la herramienta VDSF-VN en detalle, incluyendo los dos componentes que forman su interfaz gráfico (*GatcomSUMO* y *GatcomVideo*), así como su funcionamiento. Por último, se indican las líneas de trabajo futuro en la sección IV.

II. Trabajos relacionados

Varias herramientas se han desarrollado para la realización de experimentos con VANETs mediante la técnica de simulación, centrándose en alguna tarea concreta. Por ejemplo, para facilitar el uso del simulador SUMO se han desarrollado varias aplicaciones, tales como eNetEditor [9], TrafficModeler [10], [11], SUMOPy [12], CityMob for Roadmaps (C4R) [13], y OSMWebWizard. No obstante, pueden aparecer errores cuando se ejecutan las simulaciones con el simulador OMNeT++ junto con Veins. Otros trabajos se centran en solucionar las dificultades para visualizar y analizar los resultados obtenidos de las simulaciones. Por ejemplo, en [14] se desarrolló un *framework* denominado *Web-based Visualization Tool for VANET Simulations* (WGL-VANET) que proporciona un interfaz *web* interactivo fácil de utilizar que da soporte al simulador ns-3 [15] y SUMO. También en [16] se propone otro *framework* denominado *Efficient Large-scale VANET Simulator* (ELVS), dando soporte a los simuladores JiST/SWANS [17] y SUMO. Fue desarrollado en Java con un interfaz gráfico que permite la visualización de los vehículos en escenarios realistas, el estado interno de sus elementos (simuladores) y también facilita el análisis de los datos obtenidos. En [18] se desarrolló otro *framework* para una aplicación específica en simulaciones VANET, basado también en el simulador JiST/SWANS.

Además de los *frameworks* anteriores, se han desarrollado otros con el objetivo de la evaluación de la transmisión de vídeo sobre redes vehiculares, donde incluso se requieren más elementos. Es el caso de *EvalVid* [19], que permite la evaluación de la calidad perceptual de la transmisión de vídeo codificado con MPEG4 basado en el cálculo de su *Peak Signal-to-Noise Ratio* (PSNR), junto con otras métricas de red como el retardo, el *jitter* y la tasa de pérdidas. Este *framework* se ha extendido para dar soporte a varios simuladores de red, tales como ns-2 [20], ns-3, y OMNeT++ junto con Castalia [21]. Este último fue implementado en otro *framework* basado en *EvalVid* denominado *Mobile Multi-Media Wireless Sensor Networks* (M3WSN) [22]. Se han desarrollado otras herramientas basadas en *EvalVid*, como *QoE Monitor* [23], que es una extensión para dar soporte al simulador ns-3.

A pesar de los *frameworks* existentes relacionados con la evaluación de la transmisión de vídeo, principalmente extensiones de *EvalVid*, ninguna de ellas da soporte al simulador OMNeT++ junto con Veins. Además, para nuestros propósitos se requería el uso del simulador de tráfico SUMO con objeto de utilizar modelos realistas de movilidad, necesarios para modelar las redes vehiculares. Por último, también se requería el uso del estándar de codificación de vídeo HEVC. A la vista de las dificultades existentes para realizar experimentos de esta naturaleza, y ante la falta de una herramienta integrada que cubriese todos nuestros requerimientos, se optó por el desarrollo del entorno VDSF-VN, el cual se describe a continuación.

III. Entorno de simulación: VDSF-VN

Como se ha comentado anteriormente, es necesario hacer uso de un gran número de elementos para este tipo de experimentos, tales como simuladores, codificadores de vídeo y otras herramientas *software*. El entorno VDSF-VN permite la gestión de todo el proceso mediante dos aplicaciones gráficas: *GatcomSUMO* y *GatcomVideo*. La primera se encarga de la generación de los archivos de configuración necesarios para las simulaciones mediante SUMO y OMNeT++, incluyendo el escenario vehicular (sintético o real) y la movilidad de los vehículos (demanda de tráfico). Estos archivos serán utilizados por la segunda de las aplicaciones gráficas para la ejecución de las simulaciones, encargándose además de la codificación previa de las secuencias de vídeo (*pre-proceso*), y de la decodificación y evaluación de la calidad de las secuencias de vídeo recibidas (*post-*

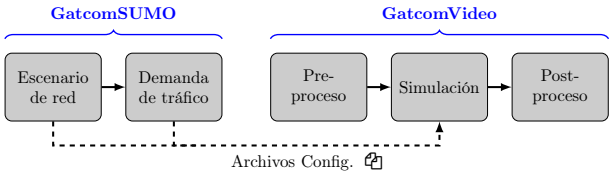

Fig. 1: VDSF-VN: vista general.

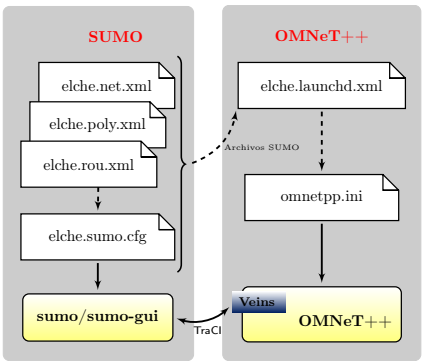

Fig. 2: Archivos de configuración para SUMO y OMNeT++.

proceso). Este esquema general se presenta gráficamente en la Figura 1.

Ambas aplicaciones gráficas han sido desarrolladas en el lenguaje de programación Java, y actúan como *front-end*, invocando a otros programas existentes que se ejecutan desde la línea de órdenes (*back-end*), simuladores (OMNeT++ y SUMO), paquetes gráficos (R, GNUplot), y otros relacionados con el procesamiento de vídeo (codificadores, paquetizadores, etc.). La mayoría de estos elementos están disponibles para múltiples plataformas, y aquellos programas desarrollados específicamente para el sistema operativo Windows pueden ser ejecutados igualmente en cualquier plataforma gracias a WINE (*Wine Is Not an Emulator*) [24].

A. GatcomSUMO

Un escenario vehicular del simulador SUMO consta de varios archivos XML (*Extensible Markup Language*) y archivos de texto plano. Pueden utilizarse escenarios reales, descargados, por ejemplo, desde OpenStreetMap [7], o escenarios sintéticos con diferentes formas geométricas, como, por ejemplo, de tipo rejilla. Dependiendo del tipo de escenario es necesario utilizar varios programas ejecutados desde línea de órdenes para la generación de los archivos correspondientes. En el caso de mapas reales descargados desde OpenStreetMap, deben importarse en el simulador SUMO mediante el programa `netconvert`, el cual genera el archivo de red necesario (.net.xml), y mediante el programa `polyconvert`, generar un archivo de obstáculos (.poly.xml) que afectan a las comunicaciones a partir de los edificios definidos en el escenario. Por otro lado, los escenarios sintéticos se generan mediante el programa `netgenerate`. Además de los archivos para el escenario, también se necesita un archivo que defina la movilidad de los vehículos o demanda de tráfico (.rou.xml), que puede generarse de forma manual, o mediante utilidades como `duarouter`, `randomTrips.py` o `dua-iterate.py`.

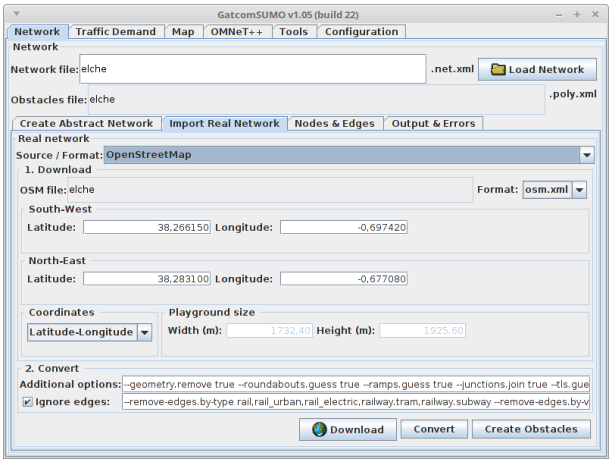

Fig. 3: GatcomSUMO: escenario real de OpenStreetMap.

Todos los archivos comentados se especifican en un archivo de configuración (.sumo.cfg) para que el simulador SUMO pueda utilizarlos (Figura 2).

Los simuladores OMNeT++ y SUMO se comunican entre sí de forma bidireccional mediante el API *Traffic Control Interface* (TraCI) [25] implementado en Veins. Para que esta comunicación sea posible, el servidor de SUMO necesita conocer toda la configuración descrita y estar en ejecución. Para ello se requiere otro archivo en formato XML (.launchd.xml) que se especifica en el archivo de configuración de OMNeT++ (omnetpp.ini). La Figura 2 muestra los diferentes archivos necesarios y su relación.

Como puede comprobarse, es necesario escribir o generar distintos archivos de configuración para poder ejecutar las simulaciones de un escenario de red vehicular, los cuales deben seguir unas reglas estrictas de sintaxis. A pesar de las herramientas proporcionadas por SUMO y algunas otras aplicaciones desarrolladas para facilitar ese fin, la preparación de los archivos de configuración para estos experimentos es una tarea que puede ser ardua y llevar mucho tiempo, debido también a que pueden surgir errores en tiempo de ejecución durante la simulación. La motivación para el desarrollo de *GatcomSUMO* [26] ha sido la de automatizar la creación de los escenarios de red y de movilidad de los vehículos, al tiempo que se realizan ciertas comprobaciones, con objeto de evitar errores en tiempo de ejecución, todo ello de una forma interactiva desde un interfaz gráfico.

GatcomSUMO permite la creación tanto de escenarios reales como sintéticos (abstractos), así como la generación del archivo de obstáculos (Figura 3). En cuanto a la demanda de tráfico, el usuario puede definir manualmente una lista de vehículos y la ruta asociada a cada uno seleccionando cada uno de los tramos que lo componen; tras la selección de cada tramo solo podrá elegirse un tramo adyacente, evitando la posibilidad de construir rutas inconexas. Otra opción disponible es la generación de cualquier número de rutas (*trips*) automáticamente entre dos puntos cualesquiera, fijos o elegidos al azar, pudiendo incluir también puntos intermedios concretos; para esta tarea se invoca a las utilidades que SUMO

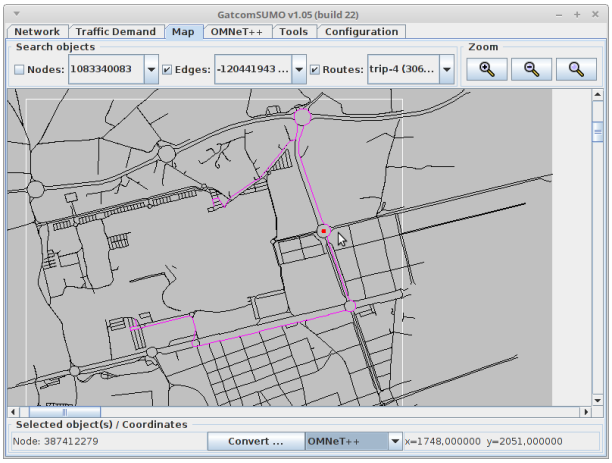

Fig. 4: GatcomSUMO: ciudad de Elche.

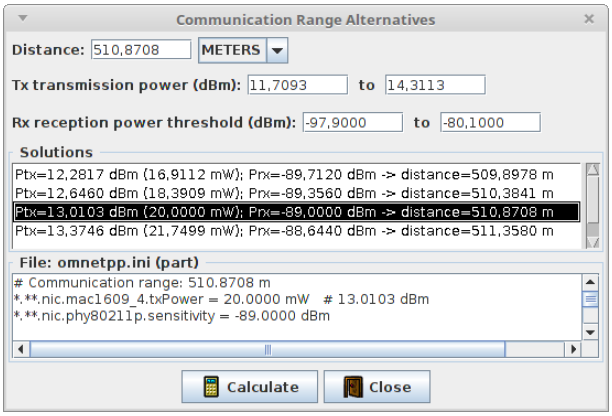

Fig. 5: GatcomSUMO: utilidad de rango de comunicación.

proporciona de forma transparente para el usuario. La aplicación también se asegura de la validez de las rutas generadas para evitar errores en tiempo de ejecución. Por ejemplo, como se ha comentado antes, se evita la creación de rutas inconexas, es decir, cada tramo que forma parte de la ruta va seguido por otro tramo adyacente, y que los tramos elegidos al azar como inicio o final de una ruta no estén aislados. Se comprueba también que la ruta tenga al menos dos tramos, pues con uno solo se obtendría un error (caso de rutas muy cortas o vehículos que no se muevan o lo hagan muy lentamente). Además, la aplicación también se asegura de que los vehículos se escriban en el archivo de rutas ordenados según el instante de salida, pues, de otro modo, algunos vehículos serían ignorados en la simulación. Por otro lado, hay casos en los que una ruta puede ser válida para SUMO pero generar un error al realizar la simulación con OMNeT++; sería el caso de un vehículo que sale de la zona definida para la simulación o *bounding box*. Aunque SUMO incluye también la utilidad `routecheck.py` para realizar ciertas comprobaciones, algunas de las condiciones mencionadas no se comprueban. Además, la aplicación permite definir varios filtros a la hora de generar rutas, descartando aquellas que no cumplan los criterios definidos, como tener una longitud determinada, o que permitan el paso de cierto tipo de vehículos (por ejemplo, para evitar calles peatonales, situación que también generaría un error durante la simulación). Todas las rutas generadas, ya sea de forma manual o automática, se pueden visualizar sobre el mapa (Figura 4).

A la hora de trabajar con infraestructura fija o *Road-Side Unit* (RSU) en una red vehicular con OMNeT++ y Veins surge un problema a la hora de especificar su posición. Esto puede hacerse de dos formas: (1) definiendo en SUMO un vehículo sin movilidad (detenido), y (2) definiendo un objeto en el archivo de configuración omnetpp.ini de OMNeT++ con sus coordenadas fijas. La primera opción tiene el inconveniente de que no es posible colocar un RSU en una posición exacta, solo puede colocarse al comienzo de uno de los tramos de la red de carreteras, pudiendo afectar también a la movilidad del resto de vehículos y provocar un atasco. Esto último podría solventarse definiendo dicho vehículo como aparcado, colocándolo fuera de la calzada, pero esto no es soportado por Veins, y se genera un error en tiempo de ejecución. Además, en versiones actuales de SUMO (por ejemplo, en v1.18.0) definir un vehículo detenido (`maxSpeed=0`) genera también un error, por lo que no sería posible utilizar esta alternativa. Con la segunda opción, un RSU se define como un objeto en OMNeT++ con unas coordenadas cualesquiera, fuera de la red de carreteras, por lo que no afectaría a la movilidad de los vehículos definidos en SUMO. Sin embargo, los simuladores OMNeT++ y SUMO utilizan sistemas de coordenadas diferentes, que además no son coordenadas reales geodésicas (latitud y longitud) o UTM. La aplicación *GatcomSUMO* permite realizar conversiones entre todos estos sistemas de coordenadas mencionados realizando la proyección y traslación necesarias, simplemente pulsando en cualquier posición en el mapa o escribiendo valores exactos, y los valores obtenidos en el sistema de coordenadas de OMNeT++ son los que pueden utilizarse en el archivo de configuración (omnetpp.ini) para especificar la posición del RSU. En la Figura 4 se muestra un RSU colocado en el centro de una rotonda (punto rojo), y sus coordenadas en la barra de estado de la parte inferior. Como pequeño inconveniente de esta segunda opción, el RSU definido como objeto en OMNeT++ no se mostrará en el interfaz gráfico de SUMO (`sumo-gui`), lo cual podría ser útil para validar su posición o realizar alguna captura de pantalla del escenario. Sin embargo, este inconveniente se ha solucionado creando automáticamente un punto de interés, o *Point of Interest* (POI), para cada RSU, apareciendo en la ventana gráfica de SUMO como un círculo, como cualquier otro POI definido.

Además de los archivos para el escenario de red y la demanda de tráfico, mediante la aplicación *GatcomSUMO* se pueden generar todos los demás archivos de configuración necesarios, tanto para SUMO como para OMNeT++ (Figura 2). Por último, *GatcomSUMO* también incluye un conjunto de herramientas que pueden ser de utilidad para escribir los valores de ciertos parámetros del archivo de configuración (omnetpp.ini), como, por ejemplo, conversores de unidades (dBm-W, velocidad, temperatura) y una

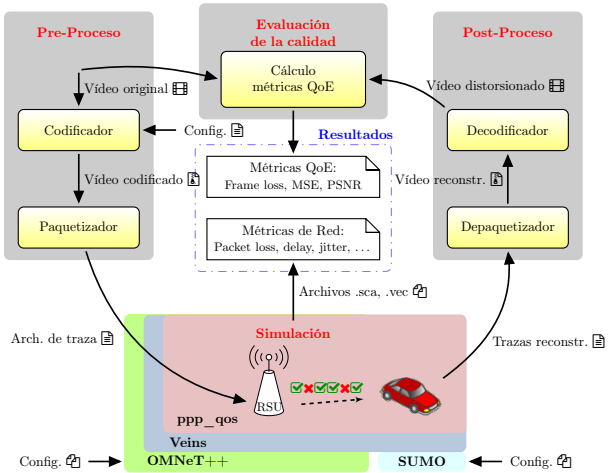

Fig. 6: GatcomVideo: flujo de trabajo.

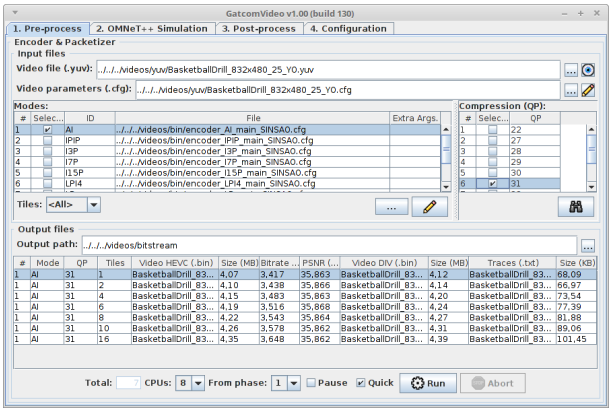

Fig. 7: GatcomVideo: pre-proceso.

calculadora para el rango de comunicación (Figura 5) que da soporte a dos modelos físicos: *Free-Space Path Loss* (FSPL) y *Two-Ray*.

B. GatcomVideo

Para evaluar la transmisión de vídeo en una red vehicular, además de la creación de los archivos necesarios para los simuladores enumerados en la sección anterior, se requieren otros elementos, como las secuencias de vídeo a transmitir. Para reducir la cantidad de información a transmitir, las secuencias de vídeo se deben comprimir, así como generar los correspondientes archivos de trazas para simular su transmisión. Por tanto, se requiere utilizar un codificador de vídeo y un paquetizador en una fase previa (*pre-proceso*). Tras la ejecución de las simulaciones, la secuencia de vídeo debe ser reconstruida y descomprimida (*post-proceso*). Estas tres fases, las cuales se describen a continuación, pueden realizarse completamente desde la aplicación *GatcomVideo*, tal como se muestra en la Figura 6. Su interfaz gráfico incluye una solapa diferente para cada una de estas fases, junto con otra más con parámetros de configuración.

La fase de *pre-proceso* se encarga de codificar las secuencias de vídeo mediante el estándar HEVC, utilizando el software de referencia HM (*HEVC Test Model*) [8]. Como resultado se obtiene una lista de *frames* que pueden ser de tipo I, P ó B. Puesto que es posible que el tamaño de estos *frames* sea mayor que el *Maximum Transmission Unit* (MTU) de la red, es necesario encapsular cada *frame* en uno o más paquetes. El software HM se ha modificado para incluir un módulo paquetizador que realiza la paquetización del flujo de vídeo mediante el protocolo *Real-time Transmission Protocol* (RTP) [27]. Como resultado se obtiene un archivo de trazas [28] que incluye los datos relativos a cada paquete a transmitir: número correlativo de paquete, tipo de *frame* de vídeo (I, P, B), instante de reproducción, tamaño, número total de paquetes del *frame* actual y su posición dentro de él. La codificación se puede realizar utilizando distintos parámetros, tales como el modo de codificación (All-Intra, Low-delay P, etc.), *Quantization Parameter* (QP) y número de *tiles* por *frame*. La codificación de vídeo es una tarea que requiere mucho tiempo de procesamiento, sobre todo si se realiza con diferentes combinaciones de estos parámetros de entrada. Todo ello se puede realizar desde la solapa *Pre-process* (Figura 7), pudiendo especificar el número de procesadores o tareas que se realizan en paralelo para reducir el tiempo necesario.

Para la fase de *simulación*, puesto que Veins no implementa la generación de tráfico de red a partir de los archivos de trazas de vídeo, fue necesaria su ampliación en un nuevo proyecto de OMNeT++ denominado 'ppp_qos' que referencia al proyecto original de Veins (Figura 6). Este nuevo proyecto contiene básicamente modificaciones a la aplicación 'TraCIDemo11p' y a la capa *Medium Access Control* (MAC) de Veins. En concreto, la funcionalidad añadida es la siguiente: (1) lectura de los archivos de trazas para la simulación de la transmisión de vídeo desde un servidor, como, por ejemplo, un RSU, (2) generación de los archivos de trazas de vídeo con los paquetes recibidos en los clientes, (3) generación de tráfico de fondo con diferente carga y número de vehículos, (4) recogida de estadísticas sobre la movilidad de los vehículos (distancia entre cada par de nodos, número de vecinos, etc.), estadísticas a diferentes niveles (aplicación, MAC y PHY) en cada nodo y globales a la red (Load, Goodput, Packet Delivery Ratio (PDR), End-to-End Delay (EED), jitter, o número medio de colisiones), y resumidas todas las anteriores por cada segundo de simulación. Las estadísticas de la capa MAC de IEEE 802.11 también se resumen por categoría de acceso, o *Access Category* (AC), para realizar estudios de rendimiento con calidad de servicio, o *Quality of Service* (QoS).

La aplicación *GatcomVideo* muestra una lista de las simulaciones definidas en el archivo de configuración de OMNeT++ y permite seleccionar las que se van a lanzar, junto con el número de procesadores a utilizar para ejecutarlas en paralelo (Figura 8). Antes de lanzar las simulaciones, el servidor de SUMO debe estar en marcha, lo cual también puede hacerse desde la propia aplicación, sin necesidad de tener que ejecutarlo desde la consola del sistema. Una vez finalizadas las simulaciones, pueden generarse una serie gráficos para el análisis de los resultados en varios formatos gráficos (.png, .eps) y resoluciones (Figu-

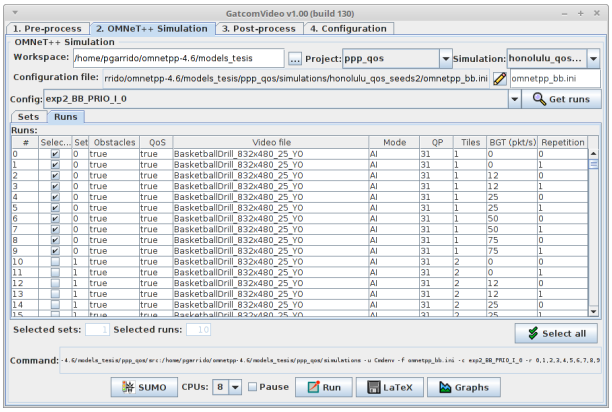

Fig. 8: GatcomVideo: conjunto de simulaciones.

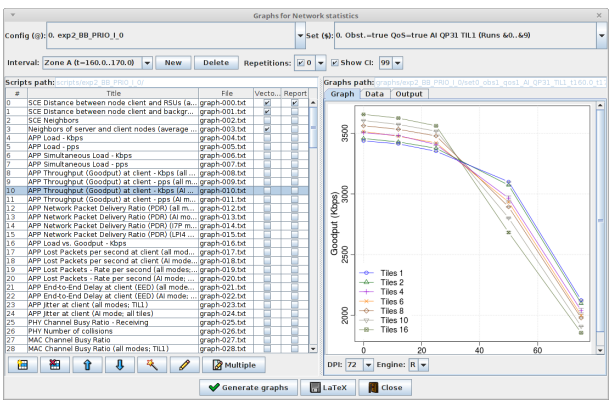

Fig. 9: GatcomVideo: gráficos de resultados.

ra 9). Los gráficos se definen en unos archivos de texto plano que pueden escribirse de forma manual o generarse desde un asistente gráfico. En un gráfico pueden incluirse resultados de una única simulación o superponer valores calculados de varias simulaciones. Gracias a las estadísticas resumidas en cada segundo de simulación, también es posible generar los gráficos definidos sólo para un intervalo de tiempo concreto; de este modo puede estudiarse el rendimiento de la red en una zona concreta, en lugar de considerar el escenario completo. Por último, los gráficos pueden incluir también intervalos de confianza en el caso de definir varias semillas de valores aleatorios. Para generar los gráficos se utiliza el paquete estadístico R [29] y Gnuplot [30], pero ambos se ejecutan de forma transparente para el usuario, es decir, el usuario no necesita escribir ningún *script* ni conocer estos programas.

El principal objetivo de la fase de *post-proceso* es realizar la evaluación de la calidad perceptual de las secuencias de vídeo recibidas. Como en nuestros experimentos los servidores transmiten de forma cíclica una misma secuencia de vídeo, lo primero que se debe hacer es dividir las distintas secuencias. La solapa *Post-process* (Figura 10) permite hacer esta división y visualizar en una tabla las distintas secuencias individuales recibidas por un cliente concreto. De este modo, el usuario puede seleccionar aquellas secuencias de interés para realizar su análisis de calidad, que consiste en reconstruir (*depaquetizar*) previamente la secuencia a partir del archivo de trazas de vídeo re-

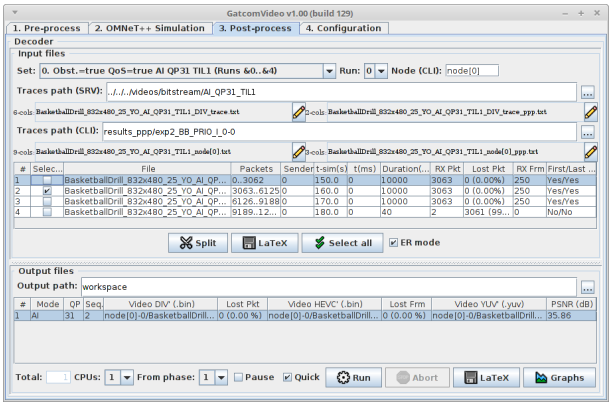

Fig. 10: GatcomVideo: post-proceso.

cibidas, y decodificarla para obtener la secuencia en su formato original, la cual estará distorsionada debido a las pérdidas de paquetes ocurridas durante su transmisión. Para esto se utiliza una versión modificada del decodificador HM que incluye un depaquetizador y la funcionalidad de ocultación de errores, o *Error Concealment* (EC), para minimizar el impacto la los paquetes perdidos en la calidad percibida. La técnica utilizada es la de *Frame Copy Concealment* [31] basada en la predicción temporal, que sustituye las zonas perdidas de un *frame* (o el *frame* completo) con las correspondientes zonas del último *frame* recibido.

Una vez decodificadas las secuencias de vídeo seleccionadas se realiza la evaluación de la calidad percibida mediante el cálculo de algunas estadísticas *Quality-of-Experience* (QoE), tales como *Frame Loss Ratio* (FLR), *Tile Loss Ratio* (TLR), *Mean Squared Error* (MSE), y el *Peak Signal-to-Noise Ratio* (PSNR). El MSE se calcula para cada *frame* promediando las diferencias cuadradas entre la intensidad de cada píxel (i, j) del *frame* original (Y_S) y del distorsionado (Y_D), tal como muestra la Ecuación 1, donde $i \in 1..N_{\text{col}}$ y $j \in 1..N_{\text{row}}$, siendo N_{col} and N_{row} el ancho y alto de los *frames* de vídeo en píxeles. El MSE también se utiliza para el cálculo del PSNR, que se calcula *frame*-a-*frame* para el componente de la luminancia (Y) promediando todos los *frames*, tal como se muestra en la Ecuación 2, donde $V_{peak} = 2^k - 1$ es el valor máximo para un número de *bits* por píxel k.

$$MSE(n) = \frac{\sum_{i=1}^{N_{\text{col}}} \sum_{j=1}^{N_{\text{row}}} \left[Y_S(n,i,j) - Y_D(n,i,j)\right]^2}{N_{\text{col}} \cdot N_{\text{row}}} \tag{1}$$

$$PSNR(n)_{dB} = 20 \cdot \log_{10}\left(\frac{V_{\text{peak}}}{\sqrt{MSE(n)}}\right) \tag{2}$$

Como este *post-proceso* requiere mucho tiempo, también se ha implementado utilizando procesamiento multihilo para ejecutar las distintas tareas de forma paralela según el número de procesadores disponibles o seleccionados, reduciendo drásticamente el tiempo requerido. Por último, se pueden generar

gráficos con las estadísticas mencionadas de forma similar a los gráficos de las estadísticas de red recogidas durante las simulaciones. Además de gráficos, la aplicación también genera archivos de texto en formato LaTeX para que puedan ser fácilmente incorporados a informes o artículos, incluyendo tablas con los datos de las secuencias de vídeo codificadas o tablas con los parámetros de las distintas simulaciones ejecutadas.

IV. Conclusiones y Trabajo futuro

Se ha presentado la herramienta *Video Delivery Simulation Framework over Vehicular Networks* (VDSF-VN), desarrollada con el objetivo de dar soporte a la investigación en la transmisión de vídeo sobre redes vehiculares mediante simulación. La herramienta se compone de varios programas que se ejecutan desde línea de órdenes, y también hace uso de otras utilidades incluidas con los simuladores SUMO y OMNeT++, y paquetes estadísticos para la generación de gráficos como R o Gnuplot. Incluye dos aplicaciones con interfaz gráfico que son las encargadas de invocar a los programas y utilidades mencionados de forma transparente: *GatcomSUMO* y *GatcomVideo*. De esta forma se evita tener que conocer los distintos parámetros o archivos de configuración que son necesarios para su invocación, evitando errores y optimizando su ejecución. Por último, los resultados pueden ser visualizados de forma gráfica desde la propia herramienta para facilitar su análisis.

Como futuros desarrollos se tiene previsto dar soporte a nuevas técnicas de protección de vídeo como *Forward Error-Correction* (FEC) y optimizar algunas de las tareas para alguna plataforma concreta.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Ciencia, Innovación y Universidades de España (Ref. RTI2018-098156-B-C54), cofinanciado con fondos FEDER (MINECO/FEDER/UE).

Referencias

[1] "High Efficiency Video Coding (HEVC)," ITU-T Recommendation H.265, 2013.

[2] "Advanced Video Coding (AVC) for generic audiovisual services," ITU-T Recommendation H.264, 2003.

[3] 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.

[4] 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, 2012.

[5] 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*, 2008, Simutools '08, pp. 60:1–60:10.

[6] 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, 2011.

[7] Mordechai (Muki) Haklay and Patrick Weber, "OpenStreetMap: User-Generated Street Maps," *IEEE Pervasive Computing*, vol. 7, no. 4, pp. 12–18, 2008.

[8] Joint Collaborative Team on Video Coding (JCT-VC), "HEVC reference software HM (HEVC Test Model) and Common Test Conditions," Retrieved August 20, 2018 from `https://hevc.hhi.fraunhofer.de`.

[9] 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.

[10] 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.

[11] W. Arellano and I. Mahgoub, "TrafficModeler extensions: A case for rapid VANET simulation using, OMNET++, SUMO, and VEINS," in *2013 High Capacity Optical Networks and Emerging/Enabling Technologies*, Dec 2013, pp. 109–115.

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

[13] M. Fogue, P. Garrido, F. J. Martinez, J. Cano, C. T. Calafate, and P. Manzoni, "Using roadmap profiling to enhance the warning message dissemination in vehicular environments," in *2011 IEEE 36th Conference on Local Computer Networks*, Oct 2011, pp. 18–20.

[14] C. Gocmenoglu, T. Acarman, and B. Levrat, "WGL-VANET: A web-based visualization tool for VANET simulations," in *2015 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, 2015, pp. 62–63.

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

[16] C. Barberis, E. Gueli, M. T. Le, G. Malnati, and A. Nassisi, "A customizable visualization framework for vanet application design and development," in *2011 IEEE International Conference on Consumer Electronics (ICCE)*, Jan 2011, pp. 569–570.

[17] R. Barr, Z. J. Hass, and R. van Renesse, "JiST/SWANS: Java in Simulation Time / Scalable Wireless Ad hoc Network Simulator," Retrieved August 20, 2018 from `http://jist.ece.cornell.edu`.

[18] John Finnson, Jie Zhang, Thomas Tran, Umar Farooq Minhas, and Robin Cohen, "A framework for modeling trustworthiness of users in mobile vehicular ad-hoc networks and its validation through simulated traffic flow," in *Proceedings of the 20th International Conference on User Modeling, Adaptation, and Personalization*, Berlin, Heidelberg, 2012, UMAP'12, p. 76–87, Springer-Verlag.

[19] Jirka Klaue, Berthold Rathke, and Adam Wolisz, *EvalVid – A Framework for Video Transmission and Quality Evaluation*, pp. 255–272, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

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

[21] 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*, 2010, p. 53.

[22] Denis Rosário, Zhongliang Zhao, Claudio Silva, Eduardo Cerqueira, and Torsten Braun, "An OMNeT++ Framework to Evaluate Video Transmission in Mobile Wireless Multimedia Sensor Networks," in *Proceed. of the 6th International ICST Conference on Simulation Tools and Techniques*, 2013, pp. 277–284.

[23] D. Saladino, A. Paganelli, and M. Casoni, "A tool for multimedia quality assessment in NS3: QoE Monitor," *Simulation Modelling Practice and Theory*, vol. 32, pp. 30 – 41, 2013.

[24] A. Julliard et al., "Wine," April, Retrieved May 18, 2020 from `https://www.winehq.org`.

[25] 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.

[26] P. Pablo Garrido Abenza, Manuel P. Malumbres, and

Pablo Piñol Peral, "GatcomSUMO: A Graphical Tool for VANET Simulations Using SUMO and OMNeT++," in *Proceedings of the SUMO User Conference 2017 (SUMO2017)*, Berlin-Adlershof, 2017, vol. 31, pp. 113–133.

[27] Ye Wang, Yago Sanchez, Thomas Schierl, Stephan Wenger, and Miska Hannuksela, "RTP Payload Format for High Efficiency Video Coding," RFC 7798, 2016.

[28] P. Seeling and M. Reisslein, "Video transport evaluation with h.264 video traces," *IEEE Communications Surveys Tutorials*, vol. 14, no. 4, pp. 1142–1165, Fourth 2012.

[29] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2016, Retrieved May 18, 2020 from `https://www.r-project.org/`.

[30] Thomas Williams, Colin Kelley, and many others, "Gnuplot 4.6: an interactive plotting program," April 2013, Retrieved May 18, 2020 from `http://gnuplot.sourceforge.net/`.

[31] S. K. Bandyopadhyay, Z. Wu, P. Pandit, and J. M. Boyce, "An error concealment scheme for entire frame losses for h.264/avc," in *2006 IEEE Sarnoff Symposium*, March 2006, pp. 1–4.