Simulación y Evaluación de Transmisión de Vídeo en Redes Vehiculares

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

Resumen— **El envío de contenido multimedia en redes inalámbricas es cada vez más demandado, creciendo proporcionalmente a la expansión de las infraestructuras y la mejora de los dispositivos. Sin embargo, las redes inalámbricas tienen un menor ancho de banda y un mayor número de pérdidas de paquetes durante la transmisión que las redes cableadas. Además, en el caso particular de las Redes Vehiculares, a estos inconvenientes hay que sumar la alta movilidad de los vehículos, con tiempos de comunicación muy cortos y coberturas limitadas. En este trabajo se propone un entorno de simulación basado en los simuladores OMNeT++, Veins y SUMO, con el objetivo de evaluar el rendimiento del envío de vídeo en redes vehiculares. Se trata de un entorno integrado que permite realizar todas las fases involucradas en este tipo de experimentos, desde la codificación inicial de las secuencias de vídeo mediante el estándar HEVC (High Efficiency Video Coding), pasando por la simulación, hasta el procesamiento posterior necesario para decodificar el vídeo recibido. El entorno genera estadísticas y gráficas relativas a la movilidad de los vehículos, de red a nivel físico (ocupación del canal), MAC (*delay*), o de nivel aplicación (porcentaje de paquetes perdidos), así como para evaluar la calidad perceptual del vídeo recibido (*frames* perdidos y PSNR).**

Palabras clave— **Redes vehiculares, Vídeo, HEVC, SUMO, OMNeT++, Veins**

I. Introducción

Las Redes Vehiculares o Vehicular Ad-hoc Networks (VANETS) aportan una serie de servicios que pueden utilizarse para la mejora del tráfico o incrementar la seguridad vial. Adicionalmente, el envío de material multimedia en redes vehiculares permite diversas aplicaciones, como la visualización en tiempo real del estado de las carreteras, difusión de anuncios o información turística según nuestra posición, y otras muchas aplicaciones de entretenimiento (*infotainment*).

Sin embargo, las comunicaciones en redes vehiculares tienen una serie de problemas. Las redes inalámbricas, en general, tienen un ancho de banda más limitado, y una mayor tasa de pérdida de paquetes en comparación con las redes cableadas. A ello hay que sumar que la transmisión de vídeo (*streaming*) requiere un ancho de banda mucho mayor y debe cumplir con unas restricciones temporales (*delay*, *jitter*) para que no se vea afectada la calidad del vídeo percibida por los usuarios. En el caso particular de las redes vehiculares, además, la alta movilidad de los vehículos limita el tiempo de comunicación. Por ello se utiliza el estándar IEEE 1609.4 WAVE (Wireless Access in Vehicular Environments) [1], que es una modificación de la capa MAC del estándar IEEE 802.11 [2] que permite establecer enlaces de forma instantánea.

En este trabajo se presenta un paquete software que permite la simulación y evaluación de la calidad de vídeo transmitido en entornos de redes vehiculares, denominado *Video Delivery Simulation Framework over Vehicular Networks* (VDSF-VN), que permite evaluar el impacto de todos estos factores en distintos escenarios. La simulación es la técnica más utilizada cuando se realiza investigación en este campo debido al alto coste y esfuerzo que sería necesario realizar para utilizar entornos de prueba reales (*testbeds*), incluyendo la captura de resultados y la dificultad para repetir los experimentos. El entorno VDSF-VN hace uso del simulador OMNeT++ [3] y el *framework* Veins (VEhicles In Network Simulation) [4] para las comunicaciones inalámbricas. Además, Veins se comunica con el simulador de tráfico SUMO (Simulation of Urban MObility) [5] mediante TraCI [6] para el control de la movilidad de los vehículos, ya sea en redes de carreteras reales o escenarios sintéticos (p. ej., de tipo rejilla).

El vídeo a transmitir consiste en secuencias de vídeo en formato YUV, codificadas mediante el estándar HEVC (High Efficiency Video Coding) [7], que permite mayores tasas de compresión que su predecesor H.264/AVC (Advanced Video Coding) [8]. Antes de la simulación es necesario codificar (por medio de HEVC) y dividir en paquetes (mediante un paquetizador) la secuencia de vídeo a transmitir, puesto que el simulador utiliza archivos de trazas de vídeo, es decir, archivos de texto con información de los distintos paquetes. Una vez finalizadas las simulaciones hay que reconstruir la secuencia de vídeo a partir de la información de las trazas de vídeo recibidas por los clientes (mediante el depaquetizador) y decodificar el vídeo (por medio de HEVC) para obtener de nuevo un vídeo en el mismo formato que el original (YUV), pero que estará deteriorado debido a los paquetes perdidos durante la transmisión. Por último, hay que analizar los resultados obtenidos, tanto de las estadísticas de red obtenidas durante la simulación, como de la calidad perceptual de los vídeos reconstruidos.

Como se puede apreciar, es necesario hacer uso de una serie de elementos independientes para llevar a cabo un experimento de este tipo. Además de todo lo relativo a la codificación y transmisión de vídeo, y de las herramientas de procesado de datos y generación de gráficos para el análisis de los resultados, hay que sumar las tareas propias de modelado de los escenarios de redes vehiculares tales como la preparación de la red de carreteras, el posicionamiento de los

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

Road-Side Units (RSUs), la creación de los vehículos y las rutas seguidas por éstos, etc. La falta de herramientas que integren todo el proceso de una manera sencilla llevó al desarrollo de VDSF-VN. Con este entorno se pretende que el investigador pueda realizar todas las tareas mencionadas de forma sencilla, gracias a la automatización de todo el proceso, pudiéndose centrar realmente en la tarea de investigación y análisis de los resultados obtenidos. Además, el entorno es multiplataforma, por lo que se puede trabajar del mismo modo en los sistemas operativos más habituales (Windows, Linux y Mac OS X).

La estructura del artículo es la siguiente. Primero, en la sección II se enumeran brevemente algunas herramientas existentes relacionadas. En la sección III se describe el entorno VDSF-VN, y sus distintos componentes. Posteriormente, en la sección IV se muestran los resultados de un primer experimento. Por último, la sección V finaliza el artículo y enumera algunos de los trabajos futuros.

II. Trabajos relacionados

Con el conjunto de simuladores OMNeT++, el *framework* Veins, y el simulador de tráfico SUMO puede realizarse la simulación de redes vehiculares, tanto la comunicación inalámbrica como la movilidad de los vehículos y la red de carreteras de una forma muy realista (carreteras, calles, carriles, semáforos, etc.). Sin embargo, estos simuladores no disponen de utilidades para el envío y recepción de secuencias de vídeo, ni la obtención de estadísticas para medir la calidad del vídeo recibido. En la bibliografía se han encontrado algunos entornos desarrollados con este propósito.

Por ejemplo, EvalVid [9] es un entorno de simulación para la transmisión de secuencias de vídeo sobre una red de comunicación real o simulada, así como la evaluación de la calidad del vídeo recibido. Por un lado, permite medir una serie de parámetros de calidad de servicio o Quality-of-Service (QoS) sobre la transmisión a través de la red, como el porcentaje de pérdida de paquetes, retardo o *delay*, y variación del retardo o *jitter*, entre otros. Por otro lado, permite evaluar la calidad del vídeo recibido y reconstruido mediante su comparación con el vídeo original transmitido, obteniendo el valor de la métrica Peak Signal-to-Noise Ratio (PSNR). El codificador de vídeo incluido en EvalVid es MPEG4, aunque otros trabajos han incluido otros estándares de codificación de vídeo más actuales.

Este *framework* se ha integrado con varios simuladores (ns-2, ns-3, OMNeT++), aunque, según sus autores, EvalVid podría utilizarse teóricamente con cualquier simulador. Por ejemplo, en [10] desarrollaron un *framework* denominado Mobile Multi-Media Wireless Sensor Networks (M3WSN) basado en EvalVid, el cual utilizaba el simulador OMNeT++ y Castalia [11]. Este *framework* permite la simulación de transmisión de vídeo de forma inalámbrica en una red de sensores, aunque no ofrece los modelos de movilidad necesarios para una red inalámbrica de tipo vehicular (VANET). Además, utiliza exclusivamente el codificador de vídeo MPEG4 que integra EvalVid.

En [12] se presenta una herramienta de código abierto denominada QoE Monitor basada en EvalVid. Se trata de una modificación de la arquitectura de EvalVid para integrarlo con el simulador ns-3, con objeto de permitir la medición de parámetros Quality-of-Experience (QoE) en dicho simulador. En [13] también se integra EvalVid con el simulador ns-3, utilizando el codificador de vídeo H.264/AVC.

Ninguno de los *frameworks* analizados por los autores permite la simulación de la transmisión de secuencias de vídeo codificadas con HEVC en Veins. Los modelos de movilidad para los vehículos son demasiado básicos y no resultan válidos para las características de las redes vehiculares. Además, la modificación de los *frameworks* existentes no es trivial. Por ejemplo, para utilizar un módulo codificador de vídeo más actual con EvalVid, tal como H.264/AVC [8] o HEVC [7], es necesario adaptar el paquetizador a las características propias del formato del vídeo codificado.

III. Entorno de Simulación

En esta sección se describe el entorno de simulación desarrollado denominado *Video Delivery Simulation Framework over Vehicular Networks* (VDSF-VN). El entorno está compuesto por un proyecto de OMNeT++ denominado 'ppp_qos' que referencia a Veins, una serie de ejecutables necesarios para el tratamiento del vídeo (codificador, paquetizador, etc.), y un entorno gráfico denominado GatcomVideo que hace de *front-end* de los ejecutables anteriores y otras tareas. Como complemento de este entorno cabe mencionar también la aplicación gráfica Gatcom-SUMO [14], que permite la generación de los escenarios y los archivos de configuración para los simuladores OMNeT++ y SUMO.

La Fig. 1 muestra el flujo de trabajo con VDSF-VN, el cual se compone de tres partes principales o fases: (1) pre-proceso, (2) simulación, y (3) post-proceso. Estas tres fases se pueden lanzar desde la aplicación gráfica GatcomVideo, la cual invoca a los archivos ejecutables para realizar su trabajo, desde la codificación del vídeo y su decodificación posterior, pasando por la ejecución de las simulaciones y generación de una serie de gráficas a partir de todas las estadísticas recogidas.

A. Pre-Proceso

En la vida real, los servidores de vídeo son los encargados de capturar las imágenes a transmitir desde una fuente externa (p. ej., una cámara de vídeo), la codificación, y su transmisión en tiempo real o bajo demanda. En nuestros experimentos, los servidores de vídeo envían trazas generadas a partir de secuencias de vídeo en formato YUV que se codifican en la fase de pre-proceso. Esta fase toma una secuencia de vídeo en formato YUV generada por el propio usuario o descargada desde diversos repositorios en Internet [15] [16]. También pueden convertirse

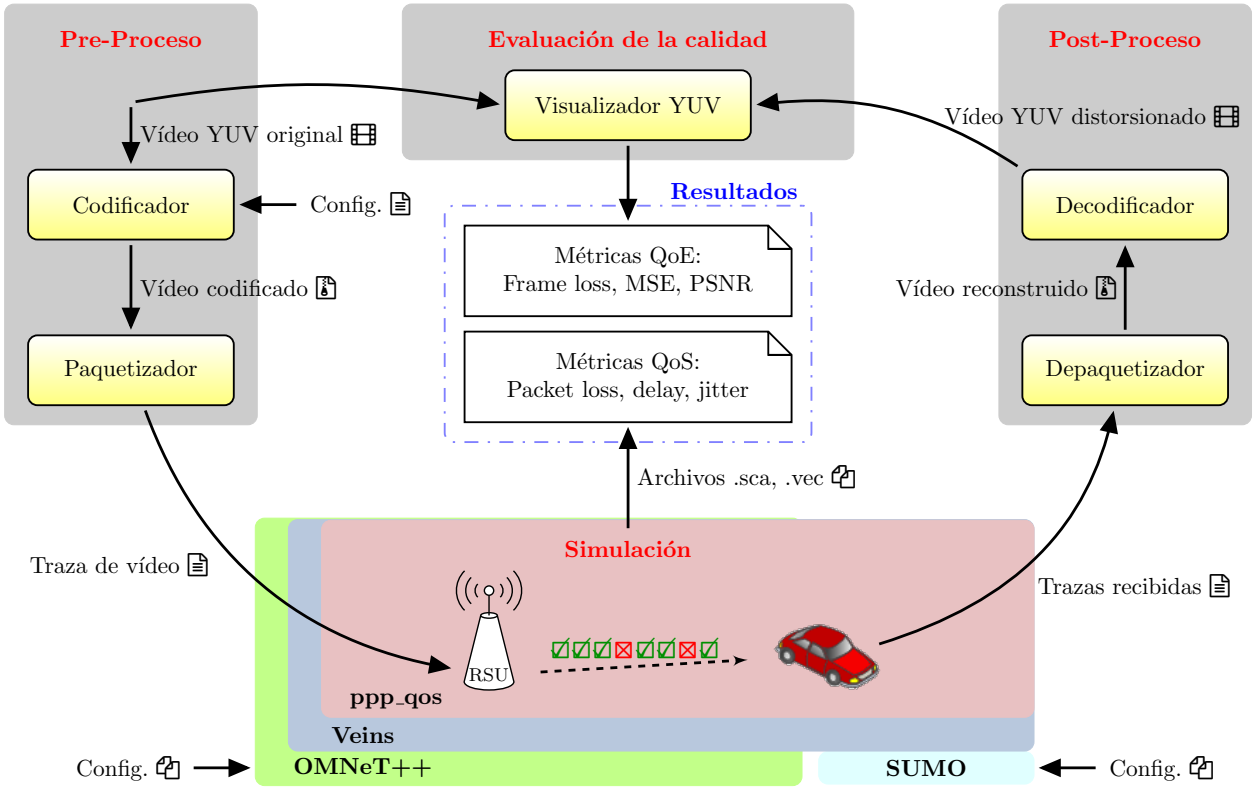

Fig. 1. Flujo de trabajo de VDSF-VN

vídeos existentes a formato YUV desde otros formatos con utilidades como *ffmpeg* o *mencoder* (incluida en MPlayer).

Para visualizar estas trazas de vídeo se requiere un visualizador YUV. Aunque existen muchos visualizadores de este formato, con objeto de reducir en lo posible el uso de herramientas externas, se ha integrado un visualizador propio en la aplicación. Este visualizador permite ver una secuencia de vídeo paso a paso o de forma continua, o a distinta velocidad (fps). También permite calcular la diferencia de calidad entre dos secuencias de vídeo, como sería el caso de la secuencia de vídeo original, y la que recibe un destinatario tras la simulación.

El codificador de vídeo (*codec*) utilizado es el software de referencia para HEVC, abreviado HM (HEVC Test Model) [17]. Este codificador recibe como entrada un archivo de vídeo en formato YUV, así como un archivo de configuración que incluye una serie de parámetros necesarios tales como el modo de codificación y el factor de cuantificación o Quantization Parameter (QP).

Los vídeos codificados (comprimidos) con HEVC se componen de una secuencia de imágenes o *frames* que pueden ser de 3 tipos: I, P, B. La pérdida de un *frame* de un tipo u otro afecta de forma distinta [18]. Por ejemplo, la pérdida de un *frame* de tipo I (I-frame) tendrá un impacto negativo alto, ya que afectará al resto de *frames* que lo referencian, normalmente un grupo de *frames* denominado Group of Pictures (GoP). El decodificador requiere un I-frame para ser capaz de decodificar el resto de *frames*. La pérdida de un *frame* de tipo P (P-frame) afectará al resto de *frames* dentro del GoP. Por último, la pérdida de un *frame* de tipo B (B-frame), aunque depende mucho del contenido de la secuencia de vídeo, tiene un menor impacto que la pérdida de *frames* de tipo I o P.

Existen diversos modos de codificación (AI, IPIP, I3P, I7P, I15P, LPi4, LP, etc.), dependiendo de la frecuencia en la que aparecen *frames* de tipo I. El valor de QP permite controlar el nivel de calidad o el ancho de banda necesario (*bitrate*) para la transmisión del vídeo codificado. Cuanto mayor valor de QP mayor compresión se conseguirá, requiriendo un menor espacio de almacenamiento y menor *bitrate* para su transmisión, a costa de una calidad menor. La aplicación ya integra los modos de codificación y valores de QP más habituales, pero permite añadir los que se necesiten, así como editar los archivos de configuración del codificador HEVC.

Como resultado de la codificación se obtiene un archivo de vídeo binario (*bitstream*) compuesto por los distintos cuadros de vídeo o *frames* codificados. Como el tamaño de los *frames* suele ser mayor que el tamaño máximo de paquete permitido por el medio inalámbrico, ~~antes de que se puedan transmitir en la red~~ es necesario ~~que cada *frame* se encapsule~~ en uno o varios paquetes, es decir, realizar una tarea de paquetización [19]. Como resultado ~~se~~ obtiene un archivo de traza de vídeo ~~que consiste en un archivo de texto (.txt)~~ con información sobre los paquetes a transmitir, incluyendo los siguientes datos: un número de paquete correlativo, el tipo de *frame* (I, P, ó B), el instante de reproducción (ms), el tamaño del paquete (bytes), el número de fragmento dentro del *frame*, y el número total de fragmentos dentro del *frame* al que pertenece. Este archivo de traza lo

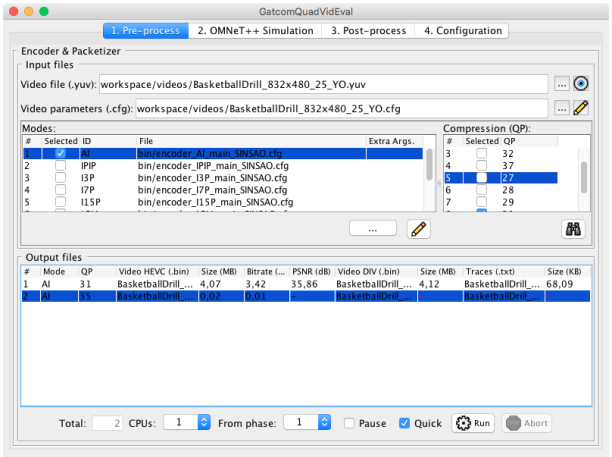

Fig. 2. Solapa: '1. Pre-process'

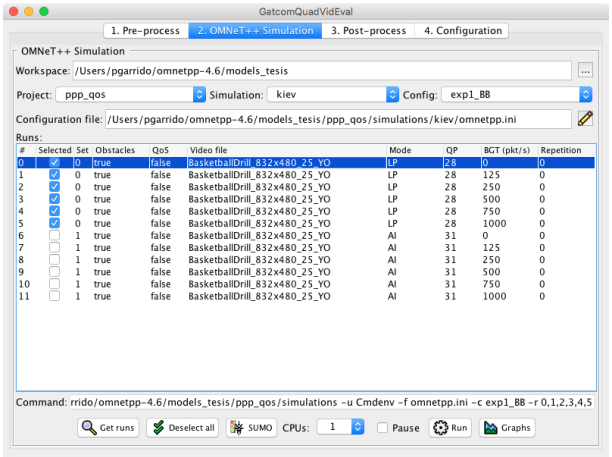

Fig. 3. Solapa: '2. OMNeT++ Simulations'

utilizan los servidores de vídeo (RSUs) durante la simulación del envío del vídeo comprimido.

Como el objetivo de esta herramienta es la evaluación de la calidad de vídeo en el receptor bajo diversas condiciones, se requiere realizar diferentes codificaciones utilizando modos y valores de QP distintos. La aplicación permite seleccionar múltiples combinaciones de estos parámetros para indicar todas las codificaciones que se quieren realizar.

En la Figura 2 se muestra la solapa del pre-proceso, en donde se especifica el vídeo a comprimir, la configuración del compresor HEVC deseada y los parámetros específicos de codificación (modo de codificación y valor de QP). Se pueden definir varias tareas de codificación modificando los parámetros de esta solapa. Pulsando el botón [Run] se inicia la ejecución de las tareas (codificación y paquetización) y la tabla se va actualizando según finalizan las tareas definidas, mostrando el tamaño de archivos obtenidos, valor PSNR del vídeo codificado, etc.

La aplicación invoca a varias herramientas externas, encargándose de controlar la ejecución y extraer los valores de interés para actualizar la tabla. Como la fase de codificación es bastante costosa temporalmente, la aplicación puede ejecutar las tareas programadas en paralelo, sacando partido del multiproceso según el número de CPUs indicados (como máximo el número de CPUs de la máquina). Además, la aplicación permite saltarse aquellas tareas que ya se han ejecutado previamente.

Cuando se codifica un vídeo con HEVC con unos valores concretos para un modo de codificación y un valor de QP, no se conoce a priori la calidad del vídeo resultante (PSNR) ni el *bitrate* necesario para su transmisión. En ocasiones puede interesar obtener un vídeo codificado con una determinada calidad o que requiera un determinado ancho de banda. Para ello, se podría ir aumentando o disminuyendo el valor de QP hasta conseguir el valor deseado, pero sería una tarea pesada, sobre todo si no se tiene experiencia. La aplicación dispone de una utilidad que permite realizar estos cálculos de forma automática para todos los modos seleccionados mediante una búsqueda binaria del valor de QP entre dos valores extremos. Como resultado, se obtiene una tabla con los valores de QP encontrados para cada uno de los modos seleccionados, incluyendo el valor de PSNR y *bitrate*. Estos valores se pueden exportar a LaTeX pulsando el botón [LaTeX], el cual genera un archivo (.tex) en el subdirectorio 'reports'.

B. Simulaciones con OMNeT++

La aplicación permite la ejecución de las simulaciones de OMNeT++ sin necesidad de utilizar el entorno gráfico del simulador ni la consola del sistema (Fig. 3). Para ello, se deben haber instalado todos los componentes necesarios: los simuladores SUMO y OMNeT++, el *framework* Veins, y el proyecto 'ppp_qos'. Dentro de este proyecto ya deben haberse creado los archivos de configuración de OMNeT++ y los de SUMO (red de carreteras y demandas de tráfico), lo cual puede hacerse de forma manual o con la ayuda de GatcomSUMO [14].

El proyecto 'ppp_qos' referencia al proyecto original de Veins, y contiene el módulo 'TraCIDemo11p' de Veins modificado. También incluye algún otro archivo que ha sido modificado para incluir nuevas estadísticas, como los archivos de la capa MAC, a los que se les ha añadido estadísticas desglosadas por categoría de servicio o Access Category (AC). El módulo 'TraCIDemo11p' lo utilizan los servidores, los clientes, y los vehículos que emiten tráfico de fondo (*background*). Por ejemplo, los servidores pueden leer archivos de trazas y simular su transmisión a velocidad constante o Constant Bit Rate (CBR). Por otro lado, los clientes escriben en un archivo los datos relativos a los paquetes recibidos, a partir del cual, es posible reconstruir en el post-proceso las secuencias de vídeo recibidas. Por último, los coches que emiten tráfico de fondo envían paquetes de un tamaño fijo a una velocidad constante.

Las secuencias de valores definidas en el archivo de configuración de OMNeT++ (omnetpp.ini) determinan el número de simulaciones a ejecutar (*runs*), en todas sus combinaciones, así como el número de repeticiones (parámetro `repeat`) de cada una variando la semilla (parámetro `seed-set`) del generador de números aleatorios. La aplicación permite obte-

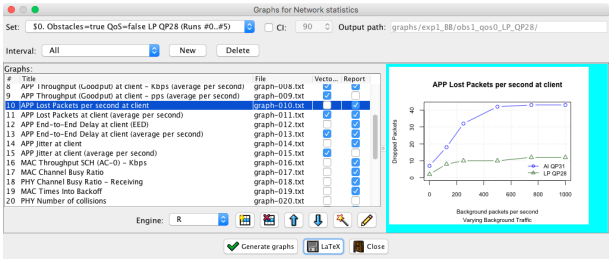

Fig. 4. Diálogo de gráficos de estadísticas de red

ner la lista de simulaciones de la configuración seleccionada al pulsar el botón [Get runs], rellenando la tabla con los parámetros de cada una de las ejecuciones. El archivo omnetpp.ini debe tener definidos estos parámetros, indicando si se va a utilizar calidad de servicio o no (parámetro `qos`), el nombre del archivo de vídeo a transmitir (`videoEntrada`), el modo de codificación (`encoderMode` y `nEncoderQP`), y el número de paquetes por segundo a transmitir como tráfico de fondo (`nBGTraffPPS`). Los parámetros `encoderMode` y `nEncoderQP` son dos secuencias de valores en paralelo, es decir, no se consideran todas las combinaciones, sino cada par de valores de cada una (p. ej. LP_QP28, AI_QP31). Lo siguiente es un fragmento del archivo omnetpp.ini con los valores de los parámetros que definirían las simulaciones a realizar:

```
*.withObstacles = ${withObstacles=true}
*.**.appl.dataPriorityFrameType = ${qos=
    false}
*.**.appl.sVideoEntrada = ${videoEntrada=
    "BasketballDrill_832x480_25_YO"}
*.**.appl.sEncoderMode  = ${encoderMode=
    "LP", "AI"}
*.**.appl.nEncoderQP    = ${encoderQP=
     28 ,  31 ! encoderMode}
*.node[1..].appl.nBGTraffPPS = ${bgt=
     0,125,250,500,750,1000}
```

El usuario puede seleccionar las simulaciones que realmente quiere lanzar, mostrando en la parte inferior la orden que ejecutará la aplicación. Antes de lanzar las simulaciones, el servidor SUMO [5] tiene que estar en marcha, lo cual puede hacerse ejecutando un script desde la consola (sumo-launchd.py), o simplemente pulsando el botón [SUMO]. Una vez iniciado el servidor ya se pueden lanzar las simulaciones seleccionadas pulsando el botón [Run], ejecutándose en paralelo tantas simulaciones como número de CPUs seleccionados.

Para cada una de las simulaciones ejecutadas se genera un archivo con estadísticas escalares (.sca) y otro con estadísticas vectoriales (.vec). En este punto se pueden generar gráficos para visualizar los datos obtenidos pulsando el botón [Graphs], que muestra el diálogo de gráficos de estadísticas de red (Fig. 4). Los gráficos se definen en unos archivos de texto con un formato propio (graph-000.txt, graph-001.txt, etc.) que pueden editarse manualmente o con un asistente que facilita la selección de las estadísticas y evita tener que conocer los nombres de los parámetros y sus valores válidos. Los gráficos se generan utilizando dos motores de generación de gráficos: el paquete estadístico R [20] y GNUPlot [21], y pueden visualizarse desde esta misma ventana. También se pueden incluir intervalos de confianza a los gráficos, lo cual puede ser útil en el caso de haber definido varias repeticiones en las simulaciones variando la semilla.

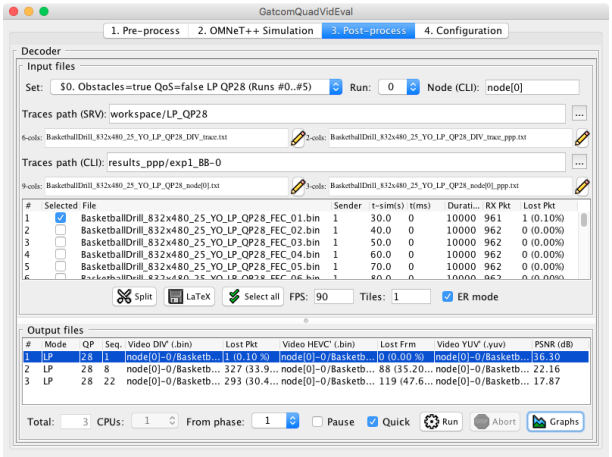

Fig. 5. Solapa: '3. Post-process'

Por último, la aplicación permite seleccionar una serie de gráficos e incluirlos en un archivo de LaTeX (.tex) junto con sus datos. Este archivo puede editarse y compilarse externamente para generar un informe (.pdf) para su análisis posterior.

C. Post-Proceso

Como resultado de la simulación se habrán recogido una serie de estadísticas que nos proporciona OMNeT++ (*delay*, *jitter*, paquetes perdidos, etc.) y otras que se han definido a nivel de aplicación (calidad de vídeo recibido, tasa de *frames* no reconstruidos, etc.). La fase de post-proceso se encarga de reconstruir las secuencias de vídeo recibidas por cada cliente a partir de las trazas de vídeo y la secuencia de vídeo original, con objeto de poder evaluar su calidad desde el punto de vista del usuario. Para evaluar la calidad perceptual podrían realizarse encuestas subjetivas con personas reales que valorasen la calidad del vídeo recibido según las recomendaciones ITU-T [22] [23] con una escala de 1 a 5, obteniéndose un valor Mean Opinion Score (MOS). Sin embargo, la realización de este tipo de tests tiene diversos inconvenientes, como por ejemplo, su coste. Por ello, se suelen utilizar otras métricas de calidad de vídeo como el Mean Squared Error (MSE) (Eq. 1) y Peak Signal-to-Noise Ratio (PSNR) (Eq. 2), ya que son sencillas y eficientes de calcular.

$$MSE = \sqrt{\frac{\sum_{i=0}^{N_{col}} \sum_{j=0}^{N_{row}} [Y_S(n,i,j) - Y_D(n,i,j)]^2}{N_{col} \cdot N_{row}}} \quad (1)$$

donde:

N_{col} = ancho de las imágenes a comparar
N_{row} = alto de las imágenes a comparar
$Y_{S|D}(n,i,j)$ = luminancia pixel (i,j) del *frame* n

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

donde:

$V_{peak} = 2^k - 1$
k = número de bits por pixel

Las ecuaciones anteriores permiten medir la similitud entre dos imágenes, por lo que en el caso de secuencias de vídeo es necesario comparar uno a uno cada par de *frames* del vídeo original de referencia (Y_S) y el vídeo reconstruido (Y_D), y obtener el valor medio del componente Y (luminancia). Si no hay pérdida de paquetes durante la transmisión, ambas secuencias de vídeo deberán tener el mismo valor en estas métricas.

En el experimento realizado (detallado en la siguiente sección), los servidores (RSUs) retransmiten cíclicamente una misma secuencia a lo largo de la simulación. La solapa 'Post-proceso' de la aplicación (Fig. 5) permite analizar las secuencias recibidas, dando como resultado una tabla con información sobre cada una de ellas: archivo, remitente, instante de simulación en el que se comienza la recepción, instante de la secuencia de vídeo (0 si es desde el principio de la secuencia), duración, número de paquetes recibidos y número de paquetes perdidos. A partir de toda esta información, el usuario puede elegir aquellas secuencias que se quieren analizar (zonas de buena recepción, zonas conflictivas, etc.). Para cada una de las secuencias que se hayan elegido, la aplicación permite reconstruir el vídeo a partir de las trazas recibidas, y analizar su calidad perceptual calculando los *frames* perdidos y su valor PSNR.

En la tabla inferior de la Figura 5 se muestran los valores de estas métricas para cada una de las secuencias seleccionadas (o un signo de guión '-' en el caso de que aun no se hayan obtenido). Además, también pueden visualizarse de forma gráfica con un diálogo similar al de los gráficos de estadísticas de red. En este caso, los archivos de definición de gráficos se denominan graph-qoe-000.txt, graph-qoe-001.txt, etc. En concreto, existen gráficos con los paquetes perdidos (valor o ratio), *frames* perdidos (valor o ratio), PSNR del vídeo reconstruido, PSNR *frame*-a-*frame* y media cada segundo.

IV. Experimento

Como ejemplo se uso del *framework* propuesto se muestran los resultados de un primer experimento. Primero se describe el escenario, y la secuencia de vídeo transmitida, y posteriormente se comentan los resultados obtenidos.

A. Escenario

El escenario (Fig. 6) consiste es una zona de 2000x2000m de la ciudad de Kiev (Ucrania), descargada desde OpenStreetMap [24]. Los parámetros físicos utilizados para las comunicaciones (PHY/MAC) se muestran en la Tabla I. Se han colocado 3 RSUs (Road-Side Units) en una posición fija a lo largo de una avenida, estando prácticamente cubierta por su rango de comunicación (visualizado con un círculo de color azul en el escenario). El rango de comunicación es el valor por defecto en Veins, y se puede comprobar en el gráfico de la Fig. 7 (arriba), que muestra la distancia del cliente a los 3 servidores durante toda la simulación, así como la distancia calculada al remitente de cada paquete recibido, quedando ligeramente por encima de los 500m antes de cambiar al siguiente RSU.

En este primer experimento sólo hay dos vehículos

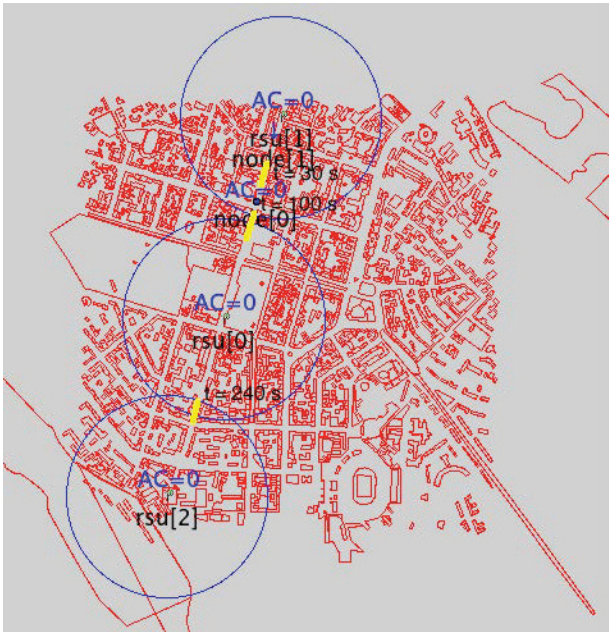

Fig. 6. Zona del escenario: ciudad de Kiev

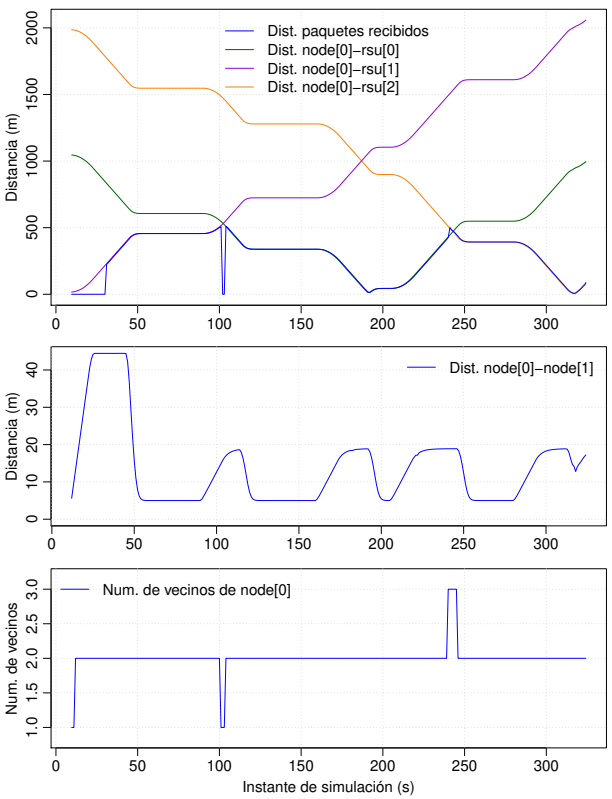

Fig. 7. Dist. cliente-RSUs, cliente-background, y vecinos

Frecuencia	5.890 GHz
Modelo de propagación	
Sin obstáculos	SimplePathlossModel
Con obstáculos	SimpleObstacleShadowing
Bitrate	18 Mbps
Potencia transmisión	20 mW
Sensibilidad recepción	-89 dBm
Rango de comunicación	510.87 m
Tamaño colas MAC	0 (infinito)

TABLA I
PARÁMETROS PHY/MAC

que se desplazan juntos a lo largo de toda la avenida, con una velocidad máxima establecida a 14 m/s (50 km/h). El tiempo de simulación es de 340 segundos, tiempo suficiente para que los dos vehículos finalicen su ruta tras haber pasado cerca de los 3 RSUs. Uno de los vehículos actúa de cliente (node[0]), recibiendo las secuencias de vídeo que los RSUs transmiten; el otro vehículo (node[1]) transmite tráfico de fondo (*background*) continuamente, emitiendo paquetes de un tamaño fijo de contenido arbitrario. El experimento se repite variando la tasa a la que el vehículo de tráfico de fondo transmite los paquetes, es decir, variando el número de paquetes por segundo (pps), con objeto de ver cómo afecta a la recepción de vídeo, definiéndose la siguiente secuencia de valores: 0,125,250,500,750,1000 pps. Como puede apreciarse en la Fig. 7 (centro), el cliente está dentro del rango de cobertura del ~~otro vehículo~~ a lo largo de toda la simulación. También puede comprobarse ~~su número de vecinos~~ dentro del rango de comunicación (~~abajo~~), coincidiendo con su paso por los distintos RSUs, zona de sombra (t=100s) y zona de solape (t=240s).

La secuencia de vídeo que transmiten los RSUs es una modificación de la denominada 'BasketballDrill', que pertenece a las Common Test Conditions del software de referencia HM [17]. La secuencia original tiene 832x480 pixels, una longitud de 500 *frames* transmitidos a una tasa de 50 fps, con una duración de 10 segundos. Para este experimento, esta secuencia se ha submuestreado ligeramente para reducirla a 250 *frames* transmitidos a 25 fps. Estos RSUs actúan como servidores de vídeo, transmitiendo todos la misma secuencia de vídeo de forma cíclica y sincronizada.

Se han realizado dos conjuntos de simulaciones utilizando el vídeo codificado con dos modos de codificación distintos: All Intra (AI) y Low-delay P (LP). En el modo All Intra (AI), todos los *frames* de la secuencia de vídeo son codificados como *frames* de tipo I, es decir, que no referencian a ningún otro *frame*. Por otro lado, en el modo Low-delay P (LP), sólo el primer *frame* es codificado como tipo I, y el resto son codificados como tipo P, los cuales usan un *frame* codificado previamente como referencia. El modo LP es muy eficiente en cuanto a compresión porque utiliza estimación y compensación de movimiento, pero es muy sensible a pérdidas de paquetes debido a las

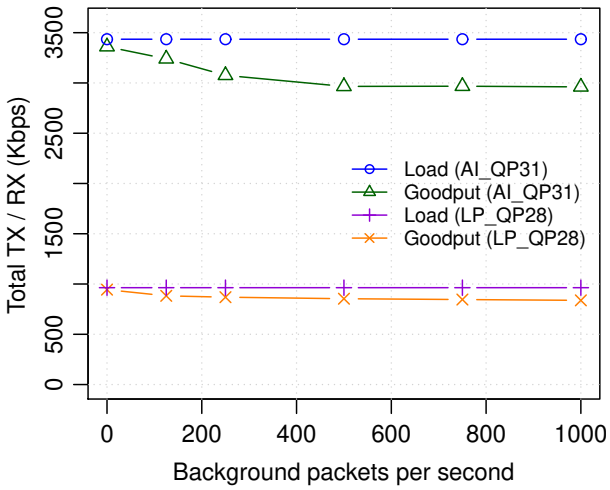

Fig. 8. Load y Goodput para LP_QP28 y AI_QP31

dependencias entre *frames*. Por el contrario, el modo AI no comprime tanto, pero es mucho más robusto, ya que el error en la recepción de un *frame* no se propaga a ningún otro *frame*.

Este experimento se ha ajustado para que el vídeo codificado tenga un valor de calidad aproximado de PSNR≈36 dB. Mediante la utilidad de búsqueda binaria incluida en la solapa 'Pre-proceso' de la aplicación GatcomVideo, se ha obtenido un valor de QP=31 para el modo AI (PSNR=35.86 dB), y de QP=28 para el modo LP (PSNR=36.16 dB). Para esos casos, la tasa de bits (*bitrate*) correspondiente es de 3.42 Mbps y 0.96 Mbps, respectivamente (Fig. 8). Alternativamente, podría haberse optado por buscar un valor de QP para conseguir un mismo *bitrate*, obteniéndose un valor de calidad (PSNR) distinta en cada caso.

Por último, los paquetes de vídeo se envían en forma de mensajes WSM (WAVE Short Messages) definidos en el estándar IEEE 1609.4/WAVE, a través de uno de los cuatro canales de servicio disponibles. A cada mensaje se le puede asignar una prioridad a nivel aplicación, o User Priority (UP), ya que el estándar WAVE utiliza el mecanismo EDCA (Enhanced Distributed Channel Access) del estándar 802.11, tanto para el canal de control (CCH) como para el canal de servicio (SCH). Según esta prioridad, al paquete se le asignará una categoría de acceso o Access Category (AC), utilizando una de las 4 colas de nivel MAC con mayor o menor preferencia. Todo ello permite el uso de calidad de servicio (QoS) mediante la asignación de diferentes prioridades a cada mensaje individual. En este primer experimento no se ha hecho uso de QoS, es decir, todos los paquetes se han transmitido con la misma prioridad (0), tanto los paquetes de vídeo como los de tráfico de fondo. Sin embargo, el *framework* está preparado para la asignación de diferentes prioridades a cada caso, o incluso, a cada paquete de vídeo según el tipo de *frame* al que corresponda (I, P, o B).

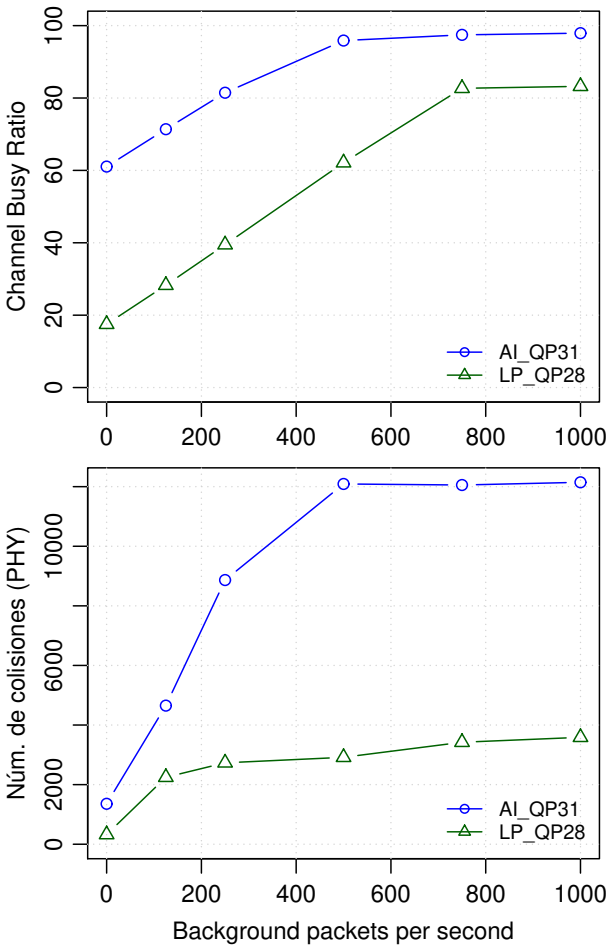

Fig. 9. Channel Busy Ratio (MAC) y Colisiones (PHY)

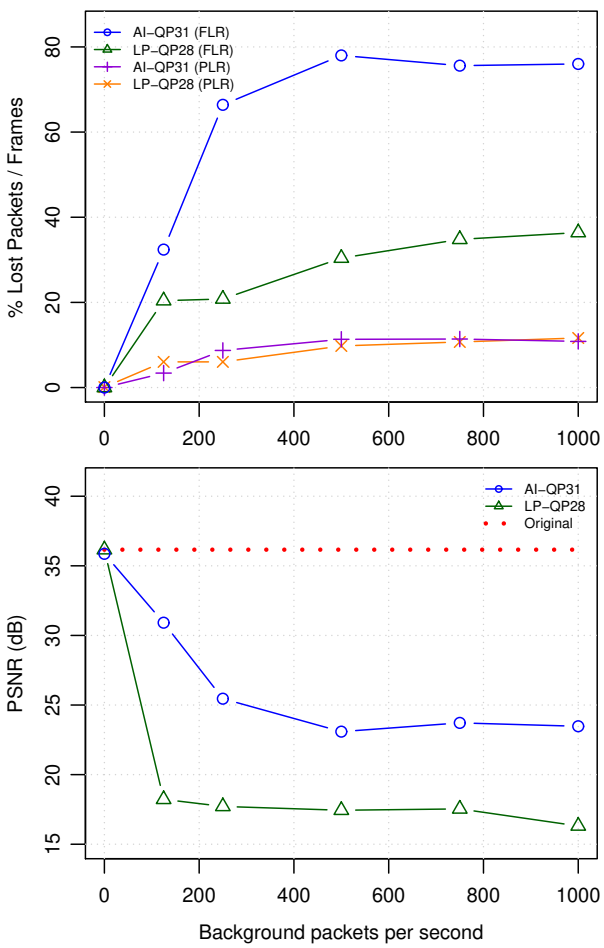

Fig. 10. Métricas Zona A: PLR vs. FLR y PSNR

B. *Resultados*

Tras la ejecución de las secuencias de simulaciones correspondientes a los dos modos de codificación de vídeo elegidos (LP_QP28 y AI_QP31) variando tráfico de fondo generado, se han obtenido los resultados que se presentan en esta sección. El estudio se centra en 3 zonas del escenario, correspondientes a la recepción de una secuencia de vídeo completa de 10s de duración: (1) Zona A, en la que el cliente está claramente dentro del rango de cobertura del primer RSU (t=30s), (2) Zona B, en la que el cliente atraviesa un espacio de 'sombra' fuera del rango de cobertura de cualquier RSU (t=100s), y (3) Zona C, en la que se produce un solape de los rangos de comunicación entre el segundo y tercer RSU (t=240s).

En el caso de no existir tráfico de fondo, es decir, en condiciones ideales, en la Zona A no se produce ninguna pérdida de paquetes, recibiéndose una secuencia de vídeo con la misma calidad que la original codificada, tal y como se esperaba. En la zona B se produce un porcentaje alto de pérdida de paquetes debido simplemente a su no recepción, indicando que se debería modificar la ubicación de alguno de los RSU para evitar zonas de sombra. Por último, en la zona C también se pierden muchos de los paquetes debido a la colisión en el medio durante su transmisión por parte de dos RSUs. En este caso ~~podría~~ utilizarse ~~algún mecanismo~~ de '*handover*'.

En la Figura 9 se puede observar la mayor ocupación del canal (Channel Busy Ratio) del modo AI_QP31 debido al mayor *bitrate* ~~necesario~~ en comparación con el modo LP_QP28, provocando también un mayor número de colisiones en el acceso al medio ~~conforme se incrementa~~ el tráfico de fondo.

En cuanto a la evaluación de la calidad de vídeo ~~percibida por el receptor,~~ en la Zona A (la de mejor recepción) vemos en la Figura 10 que se produce un porcentaje de pérdida de paquetes (PLR) similar para ambos modos, aunque una mayor pérdida de *frames* (FLR) para el caso AI_QP31. Esto es debido a que los *frames* en este modo requieren un mayor número de paquetes, y la pérdida de ~~unos pocos paquetes pueden~~ provocar que el *frame* completo sea descartado. Sin embargo, la calidad de vídeo percibida por el receptor (valor de PSNR) es claramente mayor en todos los casos ~~con~~ el modo AI_QP31 debido a que la pérdida de un *frame* no afecta al resto. Por el contrario, en el modo LP_QP28 ~~sí que~~ existen interdependencias entre *frames*, y la pérdida de un *frame* ~~puede hacer que otros *frames* también sean descartados~~. A la vista de ~~los~~ resultados, la ~~calidad~~ en el modo LP_QP28 ~~es~~ insuficiente para poder ser utilizado en este tipo de escenarios.

V. Conclusión y Trabajos futuros

Se ha presentado un entorno de trabajo integrado para la simulación de envío de flujos de vídeo en redes vehiculares. El entorno VDSF-VN se basa en los simuladores OMNeT++, Veins y SUMO, y permite realizar todas las fases necesarias desde el propio entorno, como la codificación de secuencias de vídeo con diferentes modos y nivel de compresión, la generación de trazas de vídeo, la ejecución de las simulaciones, el estudio de la calidad de los vídeos recibidos, y la obtención de gráficas para el análisis de los resultados.

Se han mostrado también los resultados de un primer experimento consistente en un escenario urbano en el que hay varios RSUs que actúan como servidores de vídeo y dos vehículos, uno actuando como receptor del vídeo y el otro como generador de tráfico de fondo. Como conclusión se ha comprobado la importancia de una buena planificación de la posición de los RSUs según su rango de comunicación para evitar solapes y zonas de sombra, así como técnicas de *handover* en el caso de zonas de solape. Aun así, la calidad de vídeo se ve muy afectada por la pérdida de paquetes, en particular en los modos de codificación de *frames* con dependencias entre sí (LP), haciendo inviable su utilización en estas circunstancias.

Esto hace que sea necesario incluir otras técnicas que permitan mejorar sensiblemente el envío de vídeo en redes VANET. Actualmente se están integrando diversas técnicas en el *framework* con las que experimentar a la hora de intentar conseguir una mejor calidad del vídeo recibido, tales como: (1) QoS a nivel MAC; (2) Forward Error-Correction (FEC); (3) protección de flujos de vídeo en la fase de codificación con el uso de *slices* o *tiles*; y (4) Error Concealment (EC).

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`).

Referencias

[1] "IEEE Standard for Wireless Access in Vehicular Environments (WAVE) – Multi-Channel Operation," *IEEE Std 1609.4-2016 (Revision of IEEE Std 1609.4-2010)*, pp. 1–94, March 2016.

[2] "IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *IEEE Std 802.11-2016*, pp. 1–3534, Dec 2016.

[3] 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).

[4] 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.

[5] 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.

[6] 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.

[7] Joint Collaborative Team on Video Coding (JCT-VC), "High Efficiency Video Coding (HEVC)," ITU-T Recommendation H.265, 2013.

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

[9] 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.

[10] 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.

[11] 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].

[12] 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.

[13] E. B. Smida, S. G. Fantar, and H. Youssef, "Video streaming challenges over vehicular ad-hoc networks in smart cities," in *2017 International Conference on Smart, Monitored and Controlled Cities (SM2C)*, Feb. 2017, pp. 12–16.

[14] P. Pablo Garrido, 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.

[15] Xiph.org, "Video Test Media," `https://media.xiph.org/video/derf/`, [Accessed April 26, 2018].

[16] Arizona State University (ASU), "Video Trace Library: YUV 4:2:0 video sequences," `http://trace.eas.asu.edu/yuv/`, [Accessed April 26, 2018].

[17] Joint Collaborative Team on Video Coding (JCT-VC), "HEVC reference software HM (HEVC Test Model) and Common Test Conditions," `https://hevc.hhi.fraunhofer.de`, [Accessed May 11, 2018].

[18] Péter Orosz, Tamás Skopkó, and Pál Varga, "Towards estimating video QoE based on frame loss statistics of the video streams," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2015, pp. 1282–1285.

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

[20] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2016.

[21] Thomas Williams, Colin Kelley, and many others, "Gnuplot 4.6: an interactive plotting program," `http://gnuplot.sourceforge.net/`, April 2013, [Accessed April 26, 2018].

[22] International Telecommunication Union, "ITU-T Recommendation P.800.1: Mean Opinion Score (MOS) terminology," Tech. Rep., July 2006.

[23] International Telecommunication Union, "ITU-T Recom-

mendation P.910: Subjective video quality assessment methods for multimedia applications," Apr. 2008.
[24] Mordechai (Muki) Haklay and Patrick Weber, "Openstreetmap: User-generated street maps," *IEEE Pervasive Computing*, vol. 7, no. 4, pp. 12–18, Oct. 2008.