Video Transmission Simulations in Vehicular Adhoc Networks

Pablo Piñol[1], Miguel Martínez-Rach[1], Otoniel López[1], José Oliver[2] y Manuel P. Malumbres[1]

Abstract— **Future Vehicular Adhoc Networks (VANETs) will have to face up with different problems like signal attenuation, packet losses, rapidly varying topology, etc. These will be caused by the inherent characteristics of VANETs, mainly, wireless communications and high velocity of the nodes. On the other hand, video transmission is highly resource demanding both in time constraints and bandwidth. So, transmitting video through VANETs is a challenging task. In this work we present a set of software tools that will allow us to do simulations in order to evaluate video transmission over VANETs in future works.**

Keywords— **Simulations, vehicular adhoc networks, video transmission.**

I. Introducción

In the future, vehicular networks will be as common as smartphones are today. Vehicles will be equipped with different kinds of sensors, a certain degree of computing capability and the ability of communicating with other vehicles and with the available infrastructure, if any. They will be able to gather information, elaborate it, and distribute it. This will make Intelligent Transport Systems (ITS) possible. One of the fields of application of VANETs (Vehicular Adhoc NETworks) is video streaming. Video streaming is linked to a deep variety of applications like digital entertainment, video on demand, tourist information, contextual advertising and other regarding security issues like emergency video call, etc.

VANETs are inhospitable networks because of two main characteristics: wireless transmission and a relative high speed between nodes. Problems like attenuation, Doppler Effect, packet losses, rapidly varying network topology and others arise. Furthermore, video streaming is highly resource demanding. With these two premises it is clear that video streaming over VANETs is a hard-to-manage task. Research needs to be done about which are the limits of video streaming over VANETs and which error resilience strategies need to be adopted so as to guarantee a minimum Quality of Experience (QoE).

Nowadays it is not feasible to do real tests with hundreds of vehicles emulating future real urban scenarios. So for this type of research, modeling is mandatory. In this paper, we have used several available modeling tools for VANETs. We have also developed a new module that allows the injection of video streaming traffic in a VANET providing different statistics. This tool will allow us to measure the limits of video streaming over VANETs and the performance of different error resilience strategies for video transmitting and receiving.

For video streaming modeling of VANETs we have used three main blocks (see Figure 1). First of all an open-source vehicular traffic simulator: SUMO (Simulation of Urban MObility) [17]. This simulator models the behavior of vehicles in routes, interacting with other vehicles, junctions, multi-lane roads, traffic lights, etc. Secondly, for the simulation of VANET network protocols, particularly IEEE 802.11p [23] and IEEE 1609 Family of Standards for Wireless Access in Vehicular Environments (WAVE) [7] we have used OMNeT++ [12]. OMNeT++ is a Network Simulation Framework which allows the development of specific network simulators. There are lots of simulators based in OMNeT++. In this particular case we have used the latest Veins (VEhicles In Network Simulation) implementation [19] [16] based in MiXiM (MIXed sIMulator) project [11], which implements wireless and mobile networks. OMNeT++ and its libraries are also open-source. The third block of the construction is video streaming injection. For this purpose we have created a new module in OMNeT++ and used preprocessed video traces like those in the Video Trace Library [20] [14]. This module provides a video sender with video packets so it can send them through the VANET scenario. The video receivers get video packets and try to reconstruct the video stream. In order to evaluate the quality of the received video streams we have generated compressed sequences and their corresponding trace files so as to be able to compare the reconstructed versions with the original ones. This will allow us to measure the received video quality by means of PSNR (Peak Signal-to-Noise Ratio) and other video quality metrics that take into account the Human Visual System, like VIF (Visual Information Fidelity), MSSIM (Mean Structural SIMilarity index) and others.

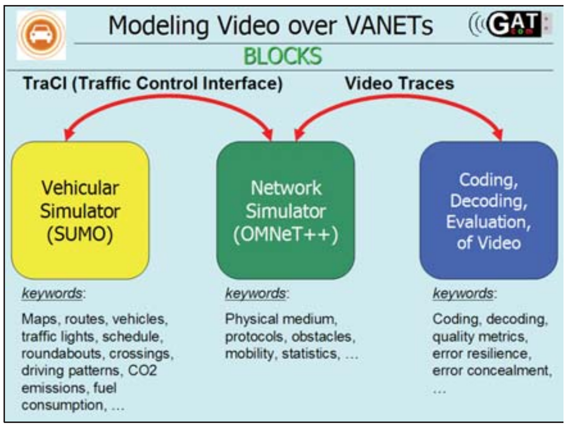

Modeling Video over VANETs
GAT
BLOCKS
TraCI (Traffic Control Interface)
Video Traces
Vehicular Simulator (SUMO)
Network Simulator (OMNeT++)
Coding, Decoding, Evaluation, of Video
keywords:
Maps, routes, vehicles, traffic lights, schedule, roundabouts, crossings, driving patterns, CO2 emissions, fuel consumption, ...
keywords:
Physical medium, protocols, obstacles, mobility, statistics, ...
keywords:
Coding, decoding, quality metrics, error resilience, error concealment, ...

Figure 1. Three main blocks for building integrated simulations.

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

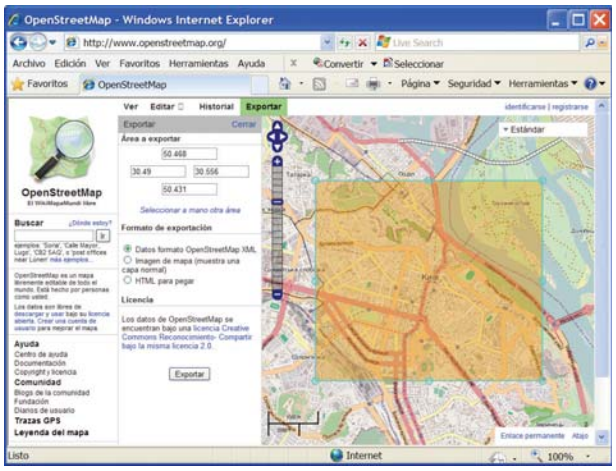
[2] Dpto. de Informática de Sistemas y Computadores, Universidad Politécnica de Valencia, e-mail: `joliver@disca.upv.es`

The vehicular simulator and the network simulator blocks will be joined together by TraCI (TRAffic Control Interface) [22]. TraCI creates a TCP connection to allow the communication between the two simulators. SUMO acts as a server (*TraCI-Server*) and OMNeT++ as a client (*TraCI-Client*). The client can send commands to the server to change the behavior of the vehicles. It also periodically requests data from the vehicular simulator to know the status of the simulation and of every single vehicle. This interaction, in both directions, can be useful to model the vehicles behavior when receiving certain type of messages. For instance, vehicles receiving warnings about a congested road could change their route. In our scenarios we could, for example, model accidents by giving a vehicle the order to stop at a certain moment. This could trigger the recording and sending of video images so that emergency services can have a preview of the accident environment.

In the rest of the paper we will show a complete example drawn from scratch to illustrate the use of all this software tools together. In section 2, we explain the creation of the main scenario by importing a real map into the traffic simulator and defining the vehicles and the routes that they will follow. Then, in section 3, we present Veins software tool and how it models the wireless vehicular network, taking into account the physical obstacles which introduce signal fading effects. In section 4, we explain how we have defined video trace files and the statistics that our module provides. At last we will present our conclusions and will outline the next steps in our research.

II. CREATING THE VEHICULAR ENVIRONMENT

As we have mentioned before, the vehicular traffic simulator used herein is SUMO. In SUMO, road networks are XML files. These files can be generated by hand (by defining nodes and edges that connect them) or by importing data with different formats. We have chosen OpenStreetMap (OSM) [13] data because you can get free maps of all around the world. OpenStreetMap is a collaborative project whose aim is to provide a free map of the whole world. As the OpenStreetMap project grows, more and more data are available. As it is maintained by users (in such a way as Wikipedia is) not all regions are described in a similar level of detail. But the main cities of the world are usually well documented. OpenStreetMap does not only gather road information but it also collects data of other kinds like buildings, rivers, parks, bus stops, schools, etc. Some pieces of this information will help us to define obstacles for wireless signals. From OpenStreetMap webpage you can download XML data in OSM format and also bitmap images of the desired zone. For the example in this paper we have selected a part of the city of Kiev (Ukraine). In Figure 2 you can see the OpenStreetMap webpage, and specially the interface for exporting map data and map images. To select a certain region and export it to an OSM file you can manually draw a rectangle over the displayed map or else you can introduce the coordinates of the region.


OpenStreetMap - Windows Internet Explorer
http://www.openstreetmap.org/
Archivo Edición Ver Favoritos Herramientas Ayuda
OpenStreetMap
Buscar
Ver Editar Historial Exportar
Exportar
Cerrar
Exportar

Área a exportar
50.468
30.49 30.556
50.431
Seleccionar a mano otra área
Formato de exportación
[✓ Datos formato OpenStreetMap XML]
Imagen de mapa (muestra una capa normal)
HTML para pegar
Licencia
Los datos de OpenStreetMap se encuentran bajo una licencia Creative Commons Reconocimiento- Compartir bajo la misma licencia 2.0

Figure 2. OpenStreetMap webpage. Selecting a region and exporting XML data and bitmap images.

In Figure 3 you can see the picked area (as a bitmap image). At the bottom of the picture you can clearly see the Olympic National Sports Complex (Національний спортивний комплекс "Олімпійський").

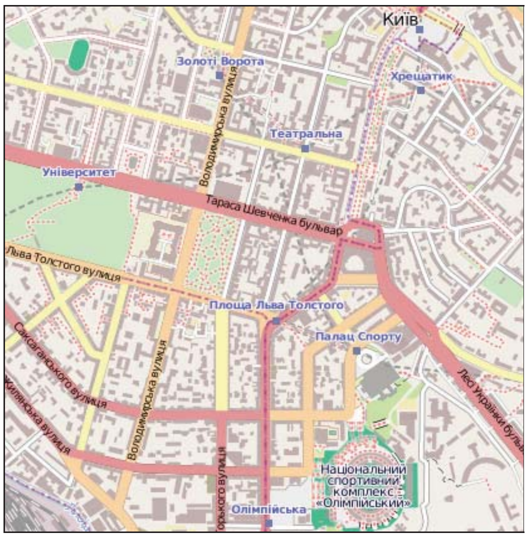
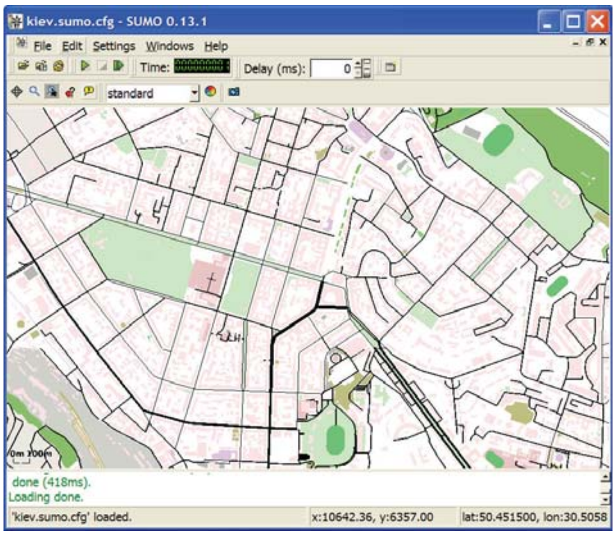


Figure 3. City of Kiev (PNG image) exported from OpenStreetMap.

SUMO includes several tools which help us to convert map data. For converting OSM data (`kiev.osm`) into SUMO road networks (`kiev.net.xml`) we will use **netconvert** tool. This tool has different options like importing data from many formats (openDrive, VISUM, VISSIM, RoboCup, MATsim) and several choices for building traffic lights. For extracting buildings (`kiev.poly.xml`) from OSM data we will use **polyconvert** tool. Once OSM data is converted into SUMO format, it can be opened with SUMO-GUI (SUMO-Graphical User Interface), as shown in Figure 4.

Now we have to define vehicles and the routes they will trace. For this purpose we will use **DUAROUTER** tool, which is also a component part of SUMO simulator. First we define trips or flows. A trip consists of an origin, a destination and the moment in which one vehicle will part from the initial edge. We can create a file including several trips and DUAROUTER tool will calculate the route for each trip and will generate a routes file (`kiev.rou.xml`). Also we can create a file with several flows. A flow is similar to a trip but the difference is that it will be used by several vehicles departing periodically. From flows file, DUAROUTER will generate the different routes for the vehicles. Also, SUMO includes another tool called **randomTrips** which generates a file with multiple random trips. Alternatively, third party tools like **C4R** (Citymob for Roadmaps) [5] [2] may be used to properly define vehicles and routes. C4R implements a wizard that imports data directly from OSM, generates vehicles and routes for SUMO and also provides with trace files for ns-2 simulator with the position of each vehicle throughout the simulation. With these trace files, it is possible to launch tests offline without the needing of connecting SUMO with ns-2.

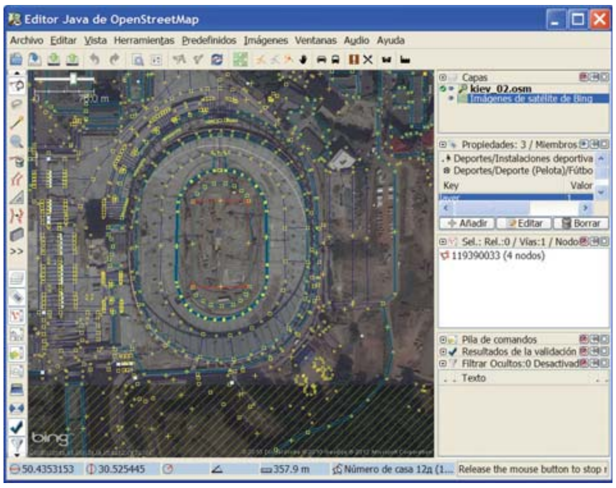


done (418ms).
Loading done.
'kiev.sumo.cfg' loaded.
x:10642.36, y:6357.00
lat:50.451500, lon:30.5058

Figure 4. City of Kiev. Data converted from OpenStreetMap format to SUMO format and opened with SUMO-GUI.

By opening all these generated files (network, buildings, vehicles/routes) with SUMO-GUI and running the simulation, we are able to see the behavior of the vehicles through this scenario: braking at junctions, stopping when traffic lights are red, accelerating when traffic lights change to green, etc.

If the OSM data of a city does not have information about buildings (because users that collaborate in OpenStreetMap project have not introduced it yet) or some of the city buildings have been drawn and some other have not, obstacles will not be modeled in the network simulator and attenuation caused by non-represented buildings will not be taken into account. To avoid this we can use JOSM (Java OpenStreetMap editor) [9] and create the missing buildings. JOSM is the main tool used to edit OpenStreetMap maps by contributors. There are two plugins that will help us with the addition of buildings: **BuildingsTools** [1] allows us to draw complete buildings with only two or three clicks of the mouse; **Terracer** [18] helps drawing terraced houses. Both plugins can be used in conjunction with orthophotos. JOSM can import orthophotos directly from different servers like Bing, Landsat or Open Aerial.


Editor Java de OpenStreetMap
Archivo
Editar
Vista
Herramientas
Predefinidos
Imágenes
Ventanas
Audio
Ayuda
Capas
kiev_02.osm
Imágenes de satélite de Bing
Propiedades: 3 / Miembros
Key
Valor
Añadir
Editar
Borrar
Pila de comandos
Resultados de la validación
Filtrar Ocultos:0 Desactivad
Texto
bing

Deportes/Instalaciones deportiva
Deportes/Deporte (Pelota)/Fútbo
Sel.: Rel.:0 / Vías:1 / Nodo
119390033 (4 nodos)
50.4353153
30.525445
357.9 m
Número de casa 12д (1...
Release the mouse button to stop r

Figure 5. Java OpenStreetMap editor (JOSM). Kiev city loaded. Detail of Olympic Stadium mixed with an orthophoto.

In Figure 5 it can be seen the JOSM editor with Kiev city OSM data loaded and mixed with a Bing no-so-recent orthophoto (where the Olympic Stadium is still under construction).

III. ADDING NETWORK SIMULATION

Network behavior of wireless communication between vehicles has been implemented by using OMNeT++ and two of the projects developed for it. Let's describe briefly these three components.

A. *OMNeT++*

As the definition extracted from its own webpage explains: "OMNeT++ is an extensible, modular, component-based C++ simulation library and framework, primarily for building network simulators". It provides the structure and libraries for developing your own network simulator and, afterwards, creating network topologies for performing tests. It has a lot of contributors who have developed many projects (INET, Castalia, ReaSE, OverSim, INETMANET, MiXiM, Veins, etc.). Two of them will help us in creating VANET simulations: MiXiM and Veins. OMNeT++ has some characteristics that make it interesting for our work: it is an open source tool so it is free and customizable; it has an easy to use GUI which is based in Eclipse (a well-known programming environment); it works under Linux/Unix, Windows and Mac OS X so you can use it in almost every computer; the two projects that offer VANET support (MiXiM, Veins) are open and in continuous development, so it is probable that they will keep on adopting VANET standards progressively [3] [4].

On the other hand, the main drawback is that the learning curve is not negligible. But this seems to be shared by most of the network simulators (OPNET, ns-2, ns-3, etc.). Performance has not been measured against other simulators by us. It is not the aim of this paper to make a comparison between network simulators or vehicular simulators. The reader can further learn on simulators and VANETs in [10].

B. MiXiM

MiXiM is a modeling framework for OMNeT++ created to simulate mobile and fixed wireless networks. It offers detailed models of radio propagation, interference estimation and wireless MAC protocols. This project allows us to model at a certain degree of detail the characteristics of wireless devices. It defines several analogue models which add the effects of the channel to the transmitted signal, by adding an attenuation mapping (which defines the attenuation factors) to the signal. It also implements different physical and MAC layers models (CSMA, 802.11, etc.) and a special module whose aim is to decide if the incoming signal is received correctly or is perceived as noise. Diverse mobility patterns and a module to simulate battery consumption are also parts of MiXiM project. MiXiM is the right framework over which Veins project can be built.

C. Veins

Veins is a tool developed by Christoph Sommer that has evolved through time. At the beginning it was developed under INET Framework package for OMNeT++ [8]. Recent versions are based on MiXiM, where wireless communications are more detailed. Veins incorporates IEEE protocols approved for its use in VANETs (namely IEEE 802.11p and IEEE 1609 WAVE) and an obstacle model [15] that adds the attenuation caused to the wireless signal by buildings. For the communication between nodes you have two channels available, the Control Channel (CCH) and a Service Channel (SCH). The standard IEEE 1609.4 (WAVE – Multi-Channel Operation) [7] defines more service channels but this is a first approximation to the implementation of the standard and it is more realistic than using one only channel for all transmissions.

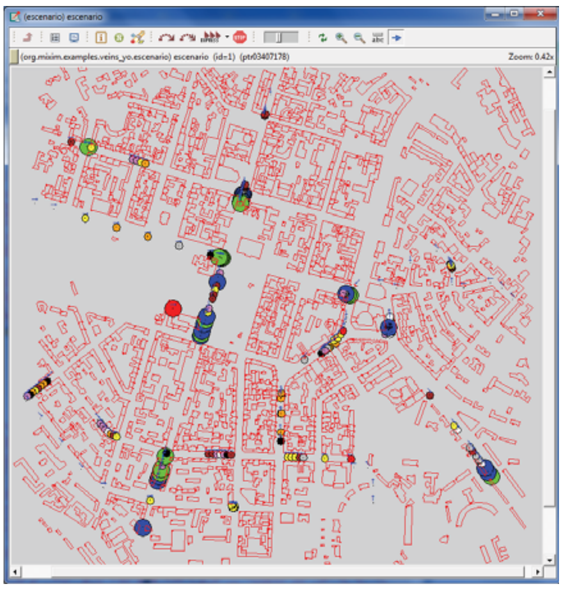

(escenario) escenario
(org.mixim.examples.veins_yo.escenario) escenario (id=1) (ptr03407178)
Zoom: 0.42x

Figure 6. City of Kiev in OMNeT++/Veins. Red polygons represent obstacles (buildings). Blue arrows denote vehicles. Big circles surround vehicles transmitting or receiving video. Small circles surround vehicles receiving other kind of data.

D. Visual representation and enhancement

In Figure 6 it can be seen the map zone that we have selected for our example (Kiev city), loaded in OMNeT++/Veins. Red drawings represent the buildings that we have imported from OpenStreetMap in the first stages of the example. Veins uses the file with polygons (kiev.poly.xml) to sketch the obstacles and to calculate the visibility between two wireless network interface cards.

Veins represents vehicles with a small rectangle and a blue arrow (which is the approximate motion direction of the vehicle). In order to make data transmission in the simulation “visible” to our eyes we draw circles of two different sizes surrounding the vehicles that transmit or receive data. A small circle around a car indicates that this car has received some data (but not video data). These small circles will cycle through 7 different colors for making reception of data clearly noticeable. Big circles will surround vehicles transmitting or receiving video data. The circle around a video transmitter will alternate between 2 colors (red/pink) each time it sends a video packet. The circle around a video receiver will alternate between 2 colors (blue/green) each time it receives a video packet. This graphical representation allows us to check visually what is really happening in the simulation and to detect possible errors in the design of the experiments. Also we can see how obstacles hinder the transmissions and contribute to packet losses. In the example that we have drawn, all the cars in the simulation keep sending a beacon every 1 second through the control channel. These beacons are not graphically represented for clarity reasons.

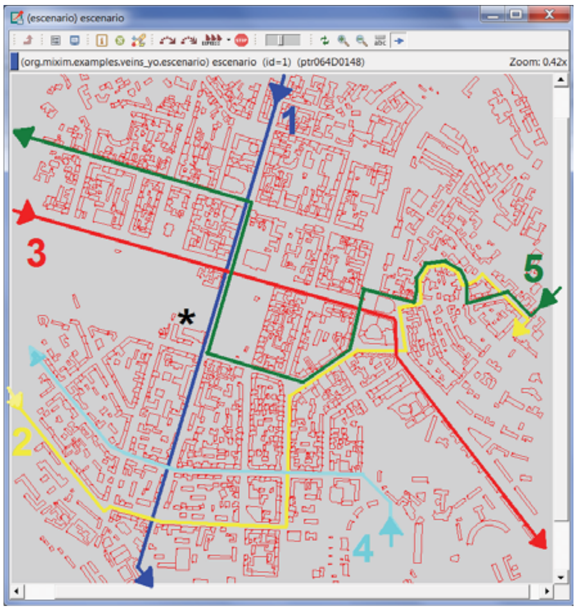
In this simple example we have modelled a static wireless device (which is implemented by a car that remains stopped for the whole simulation). This static point transmits continuously and cyclically a short sequence of video. In real life this could be a contextual advertising application, where a certain establishment sends a video sequence to the vehicles driving near it, showing the content of its bargains, the inside of the premises, and the way to get to their car park. It could also be a crashed car transmitting a few seconds of video with the images around the vehicle previous to and later than the accident. This could help identify victims outside the car and possible dangers like combustible spills. It could also be an emergency video-call that would allow the emergency services to see the exact condition of the people inside the car in order to plan the rescue and medical treatments. In all these cases the video stream should be sent through the service channel, leaving the control channel (a critical resource) available for normal network operation and also for broadcasting beacons with critical information about the accident (short bandwidth required).

IV. INJECTING VIDEO STREAMS

Up till now we have defined a generic simulation environment which could be useful for any researcher in VANETs at any field, from hardware prototyping to protocol developing and network performance evaluation. Now we will explain the operation of the module that we have developed to do our research in video transmission over VANETs. Also we will complete our illustrating example.

A. *The developed module*

The information that OMNeT++/MiXiM needs for simulating network transmissions is the packet size: header plus payload data sizes It does not need to know the exact contents of the packet to drive the simulations. So we have prepared pre-compressed sequences of video and produced trace files with the information needed for the simulation, that is, the size of every video packet. We have also included the packet sequence number in order to be able to compare the received and decompressed videos with the original sequences. Our module lets the video transmitter read a trace file and send video packets. On the other hand, video receivers will write a similar trace file with the packets they receive correctly. Another file is written by every video receiver where the number of the first video packet received, the number of the last video packet received, the total number of received video packets and the formula to calculate the percentage of video packets lost (in the time window where the receiver can "see" the transmitter) are stored. The received and decompressed video can be compared (by computing PSNR or other video quality metrics) with the use of some tools that at the moment we have not integrated with OMNeT++. We use them offline. To be able to compute the PSNR of a sequence, frame by frame we need the same number of frames in the original sequence and the received one (which may have missing frames). So here what we do is to reproduce the behavior of a vehicle when it misses some frames: it freezes the last received frame on the screen. So we have to replicate the last received frame until a new frame is received. This will allow us to compute correctly quality metrics. (This is not strictly necessary with no-reference video quality metrics but further details on this are out of the scope of this paper).


(escenario) escenario

(org.mixim.examples.veins_yo.escenario) escenario (id=1) (ptr064D0148)

Zoom: 0.42x

Figure 7. Static vehicle (asterisk) and flows of moving vehicles (colored lines).

B. *Parameters of our example*

As we have said before, our example will have one static video transmitter and several moving video receivers. In Figure 7 you can see the static vehicle marked with a *black asterisk*. We have designed 5 vehicle flows. Three of them (1, 3, 5) pass near the black asterisk and the other two do not. Flows 1 to 4 are composed of 50 vehicles each. Flow 5 introduces 250 vehicles into the scenario. 49 out of these 450 moving cars "want" to receive the video stream. Each car will send a beacon through the control channel every 1 second (as stated before). Video will be sent through the service channel.

Our example uses the following video parameters. The video injected into the network is the well-known test video sequence called Foreman. It has a frame size of 176x144 pixels (QCIF format) and a length of 300 frames with a frame rate of 30 frames/second. In order to send sequences longer than 300 frames we concatenate the original video with a reversed version (frame by frame), and then with the original one and so on. This is a normal procedure stated in common conditions [21] for testing video compressors so that the sequence keeps smooth at the edges of concatenation and does not have abrupt discontinuities. We have compressed the sequence with H.264/AVC JM 18.2 reference software [6]. We have chosen to code every frame as an IDR frame (which means they are individually decodable and do not need other frames to be decoded). We have selected a QP value of 35 for compression.

C. *Observations and measurements*

Now we will outline the measurements and behaviors observed in our test.

For two of the three flows that pass nearby the video transmitter (flows 1 and 5) there is no interfering obstacle during the time window when they can receive video and they do not lose any packet. From the statistics files compiled we can infere that, for these two flows, all the cars belonging to the same flow have a similar time window which is around 21 seconds for flow 1 and around 16 seconds for flow 5. Vehicles in flow 3 approximate to the point of transmission through an open space so they receive a big amount of packets correctly. Then these vehicles arrive at a corner where the buildings shadow the wireless signal and they begin losing packets. In that corner there is a traffic light and its cycle affects directly to the amount of packets lost by each vehicle. Vehicles that arrive at the corner when the traffic light has just gone red, will lose lots of packets. Vehicles that arrive on a green light or that have to wait little time, will lose less packets. Once cars have surpassed those buildings they continue to receive the video sequence because there are no obstacles in between. For this flow, time windows from the first received packet to the last received packet, vary from 45 seconds to 97 seconds.

In Figure 8 we have represented the percentage of packet losses for each of the vehicles in flow 3 receiving video. By looking at it you can clearly see which vehicles have suffered a long red light. These cars have loss rates around and over 60%.

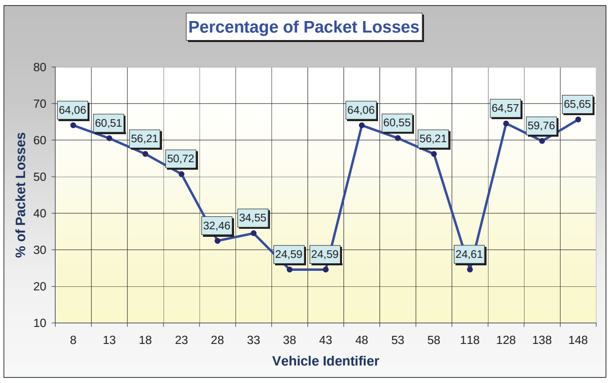


Figure 8. Percentage of packet losses for some vehicles following the route of flow 3.

In Figure 9 we show an analysis of the video quality (PSNR) of the reconstructed video sequences received by three cars belonging to flow 3. They are vehicles numbered 23, 33 and 38 which have corresponding loss rates of 50.72%, 34.55% and 24.59%, respectively. Line tagged as "Original" acts as a reference value and represents the quality of compressed video without packet losses. You can see that quality is inside a range of 30-35 dB. When vehicles are near the corner, quality drops to values near 10 dB. This is caused by keeping on the screen the last received frame which produces the "freezing" effect. This happens because burst losses appear (instead of isolated losses). You could guess packet loss rates for each vehicle by looking at Figure 9. This simple example allows us to observe the behavior and the conditions that affect video streaming in vehicular environments.

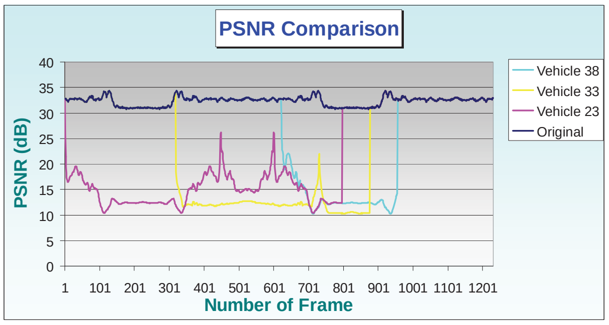


Figure 9. Comparative of PSNR values for three vehicles video reception.

V. CONCLUSIONS AND FUTURE WORK

In this work we have presented current (open source) available software tools that allow the simulation of communications in VANETs. We have introduced the development of a new module that allows the simulation and evaluation of video streaming over these networks by using trace files characterizing compressed video sequences. An example has been drawn from scratch (importing real maps) that illustrates the use of all these software tools. These simulation tools will help us in defining the limits of video streaming over VANETs, and the most relevant factors that affect the feasibility of these transmissions. We will evaluate different error resilience strategies in order to guarantee a certain quality of experience in this inhospitable environment. These items form our present and future work.

VI. ACKNOWLEDGEMENTS

This work was supported by Spanish *Ministerio de Ciencia e Innovación* under projects TIN2011-27543-C03-03 and TEC2010-11776-E.

VII. REFERENCES

[1] *BuildingsTools* plugin for JOSM, http://wiki.openstreetmap.org/wiki/JOSM/Plugins/BuildingsTools
[2] *C4R: Citymob for Roadmaps*, http://www.grc.upv.es/Software/c4r.html
[3] Eckhoff, D., and Sommer, C., 2012. *A Multi-Channel IEEE 1609.4 and 802.11p EDCA Model for the Veins Framework*. In Proceedings of 5th ACM/ICST International Conference on Simulation Tools and Techniques for Communications, Networks and Systems: 5th ACM/ICST International Workshop on OMNeT++. (Desenzano, Italy, 19-23 March, 2012). OMNeT++ 2012.
[4] Eckhoff, D., Sommer, C., and Dressler, F., 2012. *On the Necessity of Accurate IEEE 802.11p Models for IVC Protocol Simulation*. In Proceedings of 75th IEEE Vehicular Technology Conference (Yokohama, Japan, May 2012). VTC2012-Spring.
[5] Fogue, M., Garrido. P., Martínez, F. J., Cano, J. C., Calafate, C. T., and Manzoni P., 2012. *A Realistic Simulation Framework for Vehicular Networks*. In Proceedings of 5th International ICST Conference on Simulation Tools and Techniques (Desenzano, Italy, 19-23 March, 2012). SIMUTools'12.
[6] *H.264/AVC reference software*, version JM 18.2, http://iphome.hhi.de/suehring/tml
[7] *IEEE Standard for Wireless Access in Vehicular Environments (WAVE) -- Multi-channel Operation*, 2011. IEEE Std 1609.4-2010, 1-89. (Feb. 2011). DOI= 10.1109/IEEESTD.2011.5712769
[8] *INET Framework package*, http://inet.omnetpp.org
[9] *JOSM – Java OpenStreetMap editor*, http://josm.openstreetmap.de
[10][10] Martínez, F. J., Toh, C. K., Cano, J. C., Calafate, C. T., and Manzoni, P., 2011. *A Survey & Comparative Study of Simulators for Vehicular Ad Hoc Networks (VANETs)*. Wireless Communications and Mobile Computing Journal. 11, 7 (Jul, 2011), 813-828.
[11] *MiXiM project – Mixed Simulator*, http://mixim.sourceforge.net
[12] *OMNeT++ Network Simulation Framework*, http://www.omnetpp.org
[13] *OpenStreetMap*, http://www.openstreetmap.org
[14] Seeling, P., Reisslein, M., 2011. *Video Transport Evaluation With H.264 Video Traces*. IEEE Communications Surveys & Tutorials, 99 (Sept. 2011), 1-24. DOI= 10.1109/SURV.2011.082911.00067
[15] Sommer, C., Eckhoff, D., German, R., Dressler, F., 2011. *A Computationally Inexpensive Empirical Model of IEEE 802.11p Radio Shadowing in Urban Environments*. In Proceedings of 8th International Conference on Wireless On-Demand Network Systems and Services (Bardonecchia, Italy, 26-28 January, 2011). WONS 2011.
[16] Sommer, C., German, R., and Dressler, F., 2011. *Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis*. IEEE Transactions on Mobile Computing. 10, 1 (Jan. 2011), 3-15. DOI= 10.1109/TMC.2010.133
[17] *SUMO – Simulation of Urban MObility*, http://sumo.sourceforge.net
[18] *Terracer plugin for JOSM*, http://wiki.openstreetmap.org/wiki/JOSM/Plugins/Terracer
[19] *Veins – Vehicles in Network Simulation*, http://veins.car2x.org
[20] *Video Trace Library*, http://trace.eas.asu.edu
[21] Wang, Y. K., Wenger, S., and Hannuksela, M. M., 2005. *Common Conditions for SVC Error Resilience Testing*. JVT-P206d0, In 16th Meeting of Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG. (Poznan, Poland, 24-29 July, 2005).
[22] Wegener, A., Piórkowski, M., Raya, M., Hellbrück, H., Fischer, S., and Hubaux, J. P., 2008. *TraCI: An Interface for Coupling Road Traffic and Network Simulators*. In Proceedings of 11th Communications and Networking Simulation Symposium (Ottawa, Canada, April 14-17, 2008). CNS'08.
[23] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 6: Wireless Access in Vehicular Environments*, 2010. IEEE Std 802.11p-2010, 1-51. (Jul. 2012). DOI= 10.1109/IEEESTD.2010.5514475.