Autobot: An Emulation Environment For Cellular Vehicular Communications

Quentin Ricard, Philippe Owezarski

▶ To cite this version:

Quentin Ricard, Philippe Owezarski. Autobot: An Emulation Environment For Cellular Vehicular Communications. 2019 IEEE/ACM 23rd International Symposium on Distributed Simulation and Real Time Applications, Oct 2019, Cosenza, Italy. hal-02347361

HAL Id: hal-02347361

https://laas.hal.science/hal-02347361

Submitted on 5 Nov 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Autobot: An Emulation Environment For Cellular Vehicular Communications

Quentin Ricard[1,2], Philippe Owezarski[1]
[1]CNRS-LAAS, Université de Toulouse, Toulouse, France
[2]Continental Digital Services France, Toulouse, France
firstname.lastname@laas.fr

***Abstract*—The rising interest in Intelligent Transportation Systems shines light on a complex task: The evaluation of new applications and services that could prevent accident, regulate traffic, and help the automotive industry in designing energy efficient vehicles. These applications will rely on a new communication channel between vehicles, infrastructure and cloud services, and will have to operate under various network performances. However, testing applications and services in real-life networks is costly and reproducing network behaviour in a controlled environment is challenging. Furthermore, simulation tools lack real-time evaluation capabilities. Therefore, we present in this paper an environment for the real-time emulation of cellular vehicular communications. It allows the user to rapidly and cost efficiently implement and test applications under realistic mobile network performances.**

***Index Terms*—network emulation, docker, netem**

I. Introduction

In recent years, the automotive industry and the research community showed a lot of interest in connecting vehicles to the digital world facilities. In fact, today's vehicles embed complex networks. They consist of numerous computer components called Electronic and Telematic Control Unit (ECU, TCU). Each of these components are responsible of specific features in a vehicle (steering wheel, cruise control, assistance braking system (ABS)) and operate by communicating with others over a CAN bus (Control Area Network). It allows vehicles to take complicated decisions based on multiple sources of information. The introduction of a new communication channel between vehicles and the rest of the internet allows the extraction of interesting pieces of information from a vehicle enabling new ways to improve traffic efficiency, safety, and energy consumption. This new type of cyber-physical system takes a major part in future Intelligent Transportation Systems (ITS).

In this paper, we focus on Cellular vehicular networks (CVN) as a way for vehicle to communicate with the rest of the world. Unlike vehicular ad-hoc networks, CVNs rely on established mobile devices communications infrastructures such as Long-Term-Evolution (4G) and soon 5G.

While VANETs were designed to provide safety applications that require low latency between vehicles such as collision avoidance, CVNs, introduce three types of services that improve safety, fleet management and driver experience. The first type is driver assistance services (DAS), which aim in providing relevant traffic information and road condition knowledge to the driver, e.g. congestions, accidents, weather conditions. Fleet management services allow entities to monitor a fleet of vehicles in order to gather useful knowledge on each vehicle such as fuel consumption, durability, to name a few. The last type of service is related to Infotainment applications and they aim at improving the driver and passenger experience on the road by providing services such as video streaming, mail services, or other third parties applications.

Because of the rise of connected vehicles nowadays and in a very short future, evaluating ITS applications on cellular network becomes paramount. Over years, the research community studied vehicular networks in mostly two ways, simulation and real-life experiments.

Network simulation is an efficient and cost effective way to validate new designs and experiment with new protocols definitions before large scale deployment [1] while real-life experiments operate in real-time but require expensive equipments.

A third way to study vehicular network would be network emulation. It offers a combination of both real protocol and application implementation as well as realistic network models. They allow users to reproduce realistic network behaviour in controlled environment in order to test application and protocols over particular network situations. However, it was rarely attempted to emulate realistic application traffic for cellular vehicular communications.

Therefore, this paper introduces a lightweight emulation environment for vehicular communications combining network emulator tool `Netem` [5] and Operating-system-level virtualization software `docker`. Autobot is easy to install and extend and provides accurate long-term-evolution network behaviour without the need to simulate the LTE infrastructure and core-network. It provides researchers the possibility to test real-life ITS applications under different networking properties and allows them to generate networking datasets without the need for high-end hardware. The rest of this paper is as follows, we present relevant work surrounding our solution in section II. Then, we introduce our environment that allows researchers to generate networking datasets dedicated to vehicular communications in section III. A performance study of the proposed emulation environment is done in section IV. Finally, we conclude this paper in section V and present relevant future work on Autobot.

II. Related Work

In [12], the authors present a testbed for evaluating the quality of experience (QoE) perceived by users for 3D video streaming over LTE. They also use `Netem` for the emulation of the network. In their study, they used two nodes that run 3D video streaming clients, one on a computer connected to a LAN, while the other one was a smartphone running android connected to a WiFi network. The node hosting the WiFi network as well as the LAN acted as a gateway to connect both nodes to a video streaming server. The gateway runs `Netem` an manipulate the communications coming from the WiFi network to emulate LTE networking behaviour. In our case, our environment is fully virtualized using `docker` and does not need additional hardware.

In [13] the authors developed a platform that provides students lab exercises on cybersecurity. Their work consists on the emulation of applications and networking services inside a contained environment using `docker` containers. It allows student to perform offensive tasks in order to take control of the emulated network. The key advantage in this situation is that the environment is entirely confined. We followed the same approach but we focused on vehicular communications and we emulate realistic LTE network behaviour.

III. Autobot

A. Overview

Autobot is a lightweight emulation environment for cellular vehicular networks. It was designed to emulate realistic network traffic of ITS communications between vehicles and the rest of the world. Users of Autobot can test the behaviour of their own applications under different network properties. In its default settings, our environment is able to emulate a server and vehicles communicating with each other. The connectivity of the vehicles is meant to emulate that of real LTE networks in terms of latency and packet loss.

Autobot allows multiple network topology for the communications. The one that is used in Autobot by default is defined as follows: All vehicles are in the same sub-network and are allowed to communicate with a server that is located in another virtual network. The vehicles have internet access through the same gateway that is used to communicate with the server similarly to how mobile devices are connected to the internet in LTE networks. In fact, in the LTE architecture, a user equipment (typically a mobile phone), connects to the evolved packet core (EPC) network by attaching itself to an antenna (EnodeB). The EPC is in charge of identifying the device and authorizing it to communicate with external IP networks through the packet data network gateway (P-GW).

Autobot is based on Operating-System-level virtualization. Therefore, each vehicle or server runs in an isolated environment. Moreover, we are able, theoretically, to emulate a considerable number of applications running inside these isolated environment as long as the host hardware is able to maintain the performances of the network. The role of each application is to generate communications between vehicles and the server, or between vehicles. This allows users to build their own applications in order to validate them under specific network behaviour.

B. Proof of Concept

a) Virtualization: Our implementation is using `docker`, a popular containerization software. Containers run and isolate processes, packaging their own libraries, dependencies and configurations. They avoid shipping an entire operating-system (OS) for each container and therefore provide a lightweight solution in terms of CPU consumption and storage space compared to virtual-machine-based solutions.

In Autobot, each container acts as a node in an emulated LTE network. Therefore, in order to emulate n vehicles we deploy n containers. Every container is connected to a virtual network using a virtual network interface. Finally, we emulate realistic network behaviour by using `traffic-control` (or `tc`)[1], specifically `NetEm` an enhancement of `tc` written by Stephen Hemminger.

b) Traffic manipulation: We chose `NetEm` over other available network emulation tools because of its usability on virtual interfaces. Several studies [14], [15] showed that `NetEm` has limitations related to packet delay variations. It affects mostly QoE, in video streaming for example, and is irrelevant to the applications that are currently emulated in our environment.

`NetEm` is controlled using `traffic-control` a tool that is part of the `iproute2` utilities. It allows users to manipulate the Linux Kernel packet scheduler that is responsible of the receive and transmit buffer of network interfaces.

It allows the user to define matching filters in order to manipulate packets according to a defined policy. Packets can be manipulated in the following ways:

- Shaping: to control the rate of transmission of packets.
- Scheduling and Dropping: to reorder and drop packets.

Therefore, we are able to manipulate the communications of vehicles and server of our emulation scenario in terms of latency, bandwidth and packet loss.

c) Emulated Applications: We emulate an application dedicated to fleet management services. The scenario is the following: a vehicle sends telemetry information about its status to a remote server. The application sends message following the specification of Sensoris[2], an industry defined exchange format for vehicles and cloud communications. The messages include but are not limited to:

- Speed, GPS coordinates, torque, pressure applied on the brakes, dimensions of the vehicle, the nature of the ground (paved, macadam) and others.

The data was extracted from a vehicle during a trip of 1.5 hour near the city of Toulouse, France, and is artificially replayed in our emulation to craft the Sensoris messages.

[1]http://man7.org/linux/man-pages/man8/tc.8.html
[2]https://sensor-is.org/

The messages are sent to the server in our emulation over the MQTT protocol. MQTT, for Message Queuing Telemetry Transport is a publish-subscribed-based ISO standard that is very popular in the Internet-of-Things for its low bandwidth requirement and small code foot-print. We believe that it is highly possible that it will be used by vehicles so send similar information to remote servers in ITS. In fact, even though mobile communications are getting cheaper, there is still a non negligible cost to send data on cellular networks. Furthermore, the reception quality is changing depending on the position of the vehicle, i.e. urban and rural do not experience the same bandwidth and latency variations. Therefore, we believe the industry will chose the protocol that is the most efficient in terms of bandwidth and cost.

d) Defining Network characteristics: In order to define accurate network characteristics for our emulation environment we used a raspberry Pi and connected an antenna equipped with 4G connectivity subscribed to a major French mobile operator. We measured the latency between the device and a famous DNS server (8.8.8.8). The captures were generated following different mobility scenarios (highway, countryside, ring road). Results of this experiment were used to set up the `netem` rules.

IV. Evaluation

We evaluated Autobot based on **CPU consumption**: to define the maximal number of vehicles that we are able to emulate on two hardware set up and **network behaviour**: to assert the ability of Autobot to emulate the network behaviour properties used to configure `Netem`.

We evaluate Autobot using two different hardware set-up. The first evaluation is done on a virtual machine running on a laptop. The host is running Windows 10 with an Intel Core i7-6820 (2.71 GHz) and 16 gigabytes of RAM. The virtual machine executing Autobot runs on Ubuntu (18.01) and is allocated 6 processors and 6 gigabytes of RAM. The second evaluation is done on a high end server running Ubuntu 16.04 with 32 processors (2.6 GHz) and 64 gigabytes of RAM.

In the emulation, every vehicle is sending Sensoris messages at varying frequencies using an MQTT client. We alert the reader on the fact that sending a Sensoris message might lead in multiple TCP frames being sent. Our goal is to emulate as many vehicles as possible while maintaining the behaviour of the containers unchanged as well as maintaining the expected network behaviour.

A. CPU consumption

We collected the memory and CPU consumption of every set up while running Autobot. During each run of Autobot, we defined the number of vehicles to emulate and the frequency at which they would send MQTT messages. The parameters for the evaluation on the server and laptop were defined as in table I, where the values correspond to:

- min and max #veh represent the minimum and maximum number of vehicles in the emulation;
- min f and max f represent the minimum and maximal frequency at which vehicles send MQTT messages;
- vincr and fincr represent respectively the increments of the number of vehicles and frequency used between each step during the evaluation;

Set up	min #veh	max #veh	vincr	min f	max f	fincr
Server	10	40	5	0.5	2.5	0.5
Server	10	40	5	3	8	1
Server	40	200	20	3	8	1
Server	200	400	50	1	11	0
Laptop	10	100	10	1	8	1

TABLE I: Parameters for the evaluation depending on Hardware set up, number of vehicles and frequency of messages.

The results of the evaluation are depicted in figure 1. We notice that while the number of vehicles tends to have little impact on the CPU consumption, the frequency of the messages has a more important impact. In fact, as stated before, the application we emulate is actually sending Sensoris messages based on data captured during a real-life trip of a vehicle. Thus, the application reads the data from a file at the frequency defined at the beginning of the emulation, and then sends the message to the server. Thus, as the frequency grows it requires more and more CPU to be allocated to the container of each vehicle. We found that the maximal frequency to which we were able to send MQTT messages that way was approximately 10Hz independently of the number of vehicles.

We did not increase the number of vehicles further as the starting time of the emulation, especially on the laptop, was exceeding 3 minutes after 100 vehicles (resp. 400 on the server). This is due to the docker environment that has to create every container but also the fact that we apply the `netem` rules sequentially to each virtual network interface of each container once its starts.

We did not notice any correlation between the frequency of messages and memory usage. In fact, the memory usage is mostly due to the capture file (88MB) that each vehicle uses to send messages.

B. Network Behaviour

In order to evaluate the network behaviour of the environment we used `tcpdump` to collect traffic on the virtual interface of one vehicle and we analysed the traces in order to assert that the delay interval between each sensoris message of the communication of the vehicle were respected as well as the latency of each packet.

The delay aspect allows us to assert that while the emulation is running each container has enough resources to send its messages on time based on the frequency parameter. While the latency aspect, on the other hand, allows us to verify that `netem` manipulates the traffic according to the rules we defined.

a) Latency: We used the highway mobility latency distribution to configure `netem`. The emulation was running 400 vehicles sending messages at 2 Hz for 30min. We note that the highway mobility distribution is relatively well respected

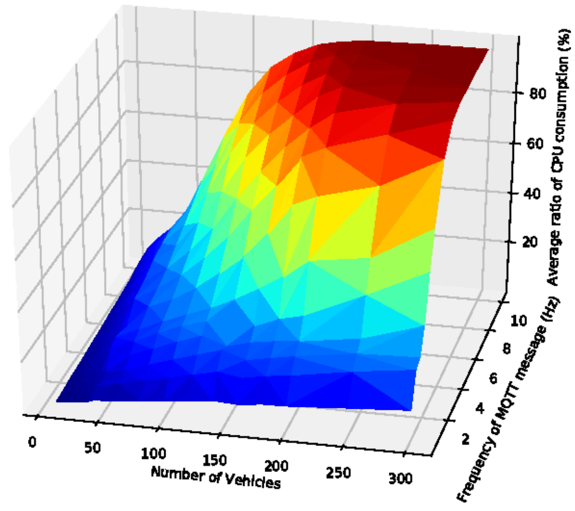

Fig. 1: Average CPU consumption ratio of Autobot on a High-end set up with varying amount of vehicles and frequency of messages.

by `netem` considering the high amount of traffic to handle at the same time (35 kBps on each interface).

b) Message delay interval: Based on the same capture file, we extracted the intervals between each messages sent from a vehicle. We noted that the vehicle was not able to send all its messages on time. In fact, the delay was only respected for the first 600 messages. We ran the same experiment but lowered the frequency of messages to 1Hz and noticed that in this case the delay between messages was respected.

C. Concluding the Evaluation

Based on the detailed analysis we provided in this section we are able to conclude that Autobot behaves as expected even when dealing with large amount of vehicles. Furthermore, we also demonstrated that it is easy to deploy on cheap hardware while still being able to emulate a significant number of vehicles. However, there is a trade-off to maintain between the frequency of messages and the number of vehicles in order to maintain the expected behaviour. The maximum amount of vehicles sending messages at 10Hz is 140 for the server and 60 at 8Hz for the laptop set up. Beyond these limits the frequency must be diminished.

V. Conclusion and Future Work

In this paper we presented Autobot, a realistic emulation environment dedicated to cellular vehicular networks. We showed that it is able to emulate large amount of vehicles with a high frequency of messages while maintaining expected network behaviour. The application integrated to the emulated vehicles depicted a telemetry service in charge of sending real-life data extracted from a vehicle to a remote server. In order to represent realistic LTE network properties, we gathered latency measures during three different mobility scenarios and integrated the results in `netem` as a distribution table.

Our evaluation shows that Autobot models fairly accurately emulates the packet latency even when emulating large amount of vehicles.

We plan extending the emulation environment to include other relevant vehicular services such as infotainment applications and update over-the-air. We also plan on extending `netem` to include fluctuating bandwidth based on a recently gathered dataset [18].

References

[1] E. Lochin, T. Perennou, and L. Dairaine, "When should i use network emulation?" *annals of telecommunications-annales des télécommunications*, vol. 67, no. 5-6, pp. 247–255, 2012.

[2] G. F. Riley and T. R. Henderson, "The ns-3 network simulator," in *Modeling and tools for network simulation*. Springer, 2010, pp. 15–34.

[3] A. Varga, "Discrete event simulation system," in *Proc. of the European Simulation Multiconference (ESM'2001)*, 2001.

[4] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. 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.

[5] S. Hemminger *et al.*, "Network emulation with netem," in *Linux conf au*, 2005, pp. 18–23.

[6] C. Sommer, R. German, and F. Dressler, "Bidirectionally coupled network and road traffic simulation for improved ivc analysis," *IEEE Transactions on mobile computing*, vol. 10, no. 1, pp. 3–15, 2010.

[7] B. Schünemann, "V2x simulation runtime infrastructure vsimrti: An assessment tool to design smart traffic management systems," *Computer Networks*, vol. 55, no. 14, pp. 3189–3198, 2011.

[8] M. Rondinone, J. Maneros, D. Krajzewicz, R. Bauza, P. Cataldi, F. Hrizi, J. Gozalvez, V. Kumar, M. Röckl, L. Lin *et al.*, "itetris: a modular simulation platform for the large scale evaluation of cooperative its applications," *Simulation Modelling Practice and Theory*, vol. 34, pp. 99–125, 2013.

[9] M. A. To, M. Cano, and P. Biba, "Dockemu–a network emulation tool," in *2015 IEEE 29th International Conference on Advanced Information Networking and Applications Workshops*. IEEE, 2015, pp. 593–598.

[10] T. Molloy, Z. Yuan, and G.-M. Muntean, "Real time emulation of an lte network using ns-3," 2014.

[11] A. Fouda, A. N. Ragab, A. Esswie, M. Marzban, A. Naser, M. Rehan, and A. S. Ibrahim, "Real-time video streaming over ns3-based emulated lte networks," *Int. J. Electr. Commun. Comput. Technol.(IJECCT)*, vol. 4, no. 3, 2014.

[12] M. Solera, M. Toril, I. Palomo, G. Gomez, and J. Poncela, "A testbed for evaluating video streaming services in lte," *Wireless Personal Communications*, vol. 98, no. 3, pp. 2753–2773, 2018.

[13] M. F. Thompson and C. E. Irvine, "Individualizing cybersecurity lab exercises with labtainers," *IEEE Security & Privacy*, vol. 16, no. 2, pp. 91–95, 2018.

[14] A. Jurgelionis, J.-P. Laulajainen, M. Hirvonen, and A. I. Wang, "An empirical study of netem network emulation functionalities," in *2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2011, pp. 1–6.

[15] J. Sliwinski, A. Beben, and P. Krawiec, "Empath: Tool to emulate packet transfer characteristics in ip network," in *International Workshop on Traffic Monitoring and Analysis*. Springer, 2010, pp. 46–58.

[16] R. Lübke, P. Büschel, D. Schuster, and A. Schill, "Measuring accuracy and performance of network emulators," in *2014 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*. IEEE, 2014, pp. 63–65.

[17] L. Nussbaum and O. Richard, "A comparative study of network link emulators," in *Proceedings of the 2009 Spring Simulation Multiconference*. Society for Computer Simulation International, 2009, p. 85.

[18] D. Raca, J. J. Quinlan, A. H. Zahran, and C. J. Sreenan, "Beyond throughput: a 4g lte dataset with channel and context metrics," in *Proceedings of the 9th ACM Multimedia Systems Conference*. ACM, 2018, pp. 460–465.