A Flexible, Protocol-agnostic Latency Measurement Platform

A Flexible, Protocol-agnostic Latency Measurement Platform

Francesco Raviglione
CNIT - Politecnico di Torino
Torino, Italy
francesco.raviglione@studenti.polito.it

Marco Malinverno
CNIT - Politecnico di Torino
Torino, Italy
marco.malinverno@polito.it

Claudio Casetti
CNIT - Politecnico di Torino
Torino, Italy
claudio.casetti@polito.it

***Abstract*—Latency is one of the key parameters of any networked system, from vehicular networks to real time video streaming. Being capable of measuring such a parameter can be very important in assessing the performances of devices under test. In this paper, we discuss how we designed a lightweight, flexible, custom latency measurement protocol, LaMP, completely agnostic of lower-layer protocols. We also present the first open source tool leveraging LaMP, called LaTe, running on any Linux-based device, which has been validated through several tests, both involving general purpose laptops and embedded devices for vehicular communications, for which the most important results are presented.**

***Index Terms*—802.11, 802.11p, latency measurements, application layer protocol, LaMP, LaTe**

I. Introduction

Network latency measurement tools play a crucial role in the provisioning of reliable and efficient networked services, such as video streaming, online gaming, and social networking. Furthermore, the upcoming vehicular communication networks are going to heavily increase the demand for ultra-Reliable Low-Latency Communication (uRLLC), one of the tenets of 5G networks, bringing attention to the need of innovative tools and protocols capable of targeting the new network architectures.

In the literature, it is possible to find several examples of tools and protocols to measure the latency between endpoints, the most notable being the ICMP protocol, in particular for what the "Echo Reply" and "Echo Request" packets are concerned. Every IP compliant system should be able to reply to ICMP requests coming from another node in the network. This mechanism is exploited by tools like `ping` and, thanks to timestamps which are embedded in such packets or stored inside the application, latency measurements are made possible. Another example is Metherxis [1], a system leveraging Virtualized Network Measurements Functions (VNMFs) to measure network device latency with micro-second grade accuracy. The idea at the base is to use Linux Containers (LXC) to create a range of VNMFs, under a single Linux host, some of them instantiating a packet generator, others a packet analyzer. Other specifically targeted protocols, such as the recently proposed Two-Way Active Measurement Protocol (TWAMP) [2], works over a client/server architecture allowing one or two way delay measurements. Finally, a dedicated IP Timestamp Option, that can be used to determine the latency between single links, is also available, as highlighted by Sherry in [3]. After identifying the above solutions as the most common for this kind of measurements, we remark that they have two main drawbacks. Firstly, `ping` relies on ICMP. Even though it can be very practical to leverage something that is implemented inside almost every networking system (i.e., the ICMP echo reply mechanism), only ICMP can actually be used to transport the timestamp data needed to compute the latency between nodes. Moreover, by using ICMP, it is possible to more precisely compute the network latency, but tools like `ping` do not provide the user with a clear estimate on the latency experienced at the application level. Secondly, Metherxis, TWAMP, and other specific protocols, even though very precise in giving the desired measurement values, all require the tested device to be compliant to specific standards (or to specific options, such as in the IP case) and possibly need some additional capabilities to be implemented, which may not be the case for all the network nodes.

For these reasons, we introduce the Latency Measurement Protocol (LaMP), an application-layer protocol which can be encapsulated inside any lower-layer protocol.

The protocol addresses the need for a lightweight framework which encapsulates the basic information needed to perform accurate latency measurements that are completely agnostic of the communication sublayers. In addition, this paper introduces an open-source tool leveraging LaMP, i.e., LaTe (*La*tency *Te*ster), a flexible client-server application that currently supports LaMP over UDP/IP to perform measurements under different conditions. In particular, we validated the LaTe functionalities through several tests deploying different communication protocols and physical media, such as Ethernet, 802.11a and 802.11p in Outside Context of BSS (OCB) mode.

The remainder of this paper is organized as follows: in Section II, the LaMP protocol is described in detail, Section III is devoted to the description of LaTe in all its components and, finally, Section IV shows the validation results.

II. LaMP protocol description

LaMP has been developed as a flexible application layer protocol, which can be encapsulated inside any lower layer protocol, and it is designed to be as much self-contained as possible. For instance, it can be encapsulated, without requiring any modification to its rules and packet format,

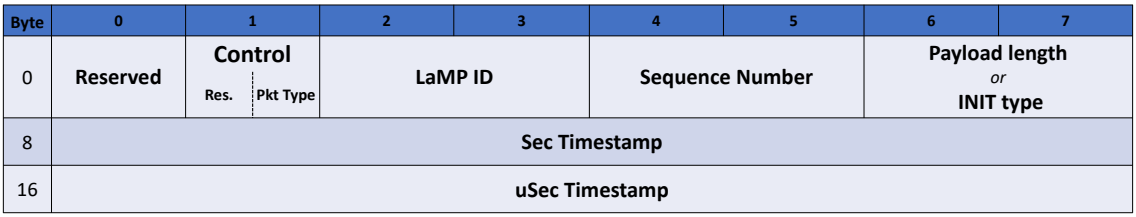

Fig. 1. Structure of the custom Latency Measurement Protocol header.

inside UDP over IPv4, or directly inside a local network raw Ethernet packet, or, to make an example related to the vehicular networking world, inside Wave Short Message Protocol (WSMP) packets which are transmitted over 802.11p.

The protocol works with a client-server paradigm: LaMP requests are sent by a client and received by a server, which replies back to the client. Then, in a typical scenario, LaMP computes the latency or Round Trip Time (RTT) as the time difference between "send timestamps" and "receive timestamps", which are managed by the applications participating in the measurement session. Typically, the "send timestamp" is inserted inside the LaMP packet by the client sending a request and the "receive timestamp" is instead gathered by the client when a reply, containing a copy of the "send timestamp" from the corresponding request, is correctly parsed. Before any session is started, a connection initialization packet is sent by the client and properly acknowledged by the server, in order to ensure that the measurements can start and the connection is stable enough.

Additional modes are also defined, including an experimental unidirectional mode, in which the client only sends requests to the server, which will be responsible for computing the latency and returning it to the client at the end of the test. This mode enables the computation of one-way latency, but, as of now, requires the clocks of the different devices to be precisely synchronized through the *Precision Time Protocol* (PTP) or through the *Network Time Protocol* (NTP).

The header size is equal to 24B, accounting for the need of a lightweight protocol but including all the necessary data, as shown in Figure 1. Every LaMP packet starts with a reserved header field, which is used by the nodes to identify if the application layer data is really a LaMP SDU, and with a control field, which is set depending on the LaMP packet type (e.g., a request, a reply, an acknowledgment, a connection initialization). The latter is further divided into a reserved sub-field and one indicating the packet type, accounting for a maximum of 16 packet types but increasing the length of the overall reserved field, making the LaMP packet detection more robust. All the 12 reserved bits are made of alternating `1` and `0`. Then, a 16 bit-long field is carried, containing an identification number which is used to identify each LaMP client-server measurement session, allowing a server to reply only to the client which sent the requests. This mechanism can be used as a basic system to identify each LaMP session and can be integrated, if necessary, with other protocol-specific session identification mechanisms, such as ports when UDP is used, or the *Provider Service ID* (PSID) when WSMP is used. A sequence number is then carried, enabling the association between each request and reply. The following 16 bit-long field is instead used to store the optional payload size or, if the packet is a connection initialization one, the type of connection which should be established (unidirectional or bidirectional). Finally, LaMP is designed to embed, together with two 64-bit precise second and microsecond timestamps, any additional payload, up to 65535 bytes, in which, if desired, other data and possibly other user-defined protocols can be encapsulated.

As each measurement session is completed, the client should gather some mandatory and possibly additional optional statistics and report them to the user. In the one-way mode instead, the server is responsible for the latency computation and for returning the report to the client.

The custom protocol specifications are open and we published them on the project website, hosted on GitHub [4].

III. LaTe tool description

LaMP has been used as a base to develop a flexible, custom, client-server command line tool to measure the latency between different devices running Linux, connected by means of wireless and/or wired physical media. This tool was written in C and released as open source software under the GPLv2 license [4].

LaTe (*La*tency *Te*ster) follows all the LaMP specifications mentioned before and it currently supports tests over a LaMP/UDP/IPv4 protocol stack. Nevertheless, additional protocols are going to be supported and we plan to insert new features as well, such as a way to measure latency in broadcast flooding situations, which can be of interest in the vehicular networking context. LaTe can measure different kinds of latency: in the current version two types of RTT (or one-way latency) measurements are supported. The first one (*User-to-user*) gathers a receive timestamp, to compute the latency, as the receiving entity (typically, a client receiving a reply) has completely processed the LaMP packet. The second one (which we called *KRT*, i.e., *Kernel Reception Timestamp*) instead obtains the reception timestamp as the time when the LaMP packet is being passed from the hardware to the kernel stack, thanks to the Linux kernel capability of generating packet timestamps.

This tool tries to exploit the philosophy at the base of LaMP, offering a flexible Linux tool in which various options can be specified by the user, such as the possibility to:

- Choose a transport protocol (although, as mentioned before, only UDP is supported at the time of this writing).
- Choose and compare raw sockets [5] and non-raw sockets, for supported transport protocols, such as UDP.
- Test over a specific interface (wired or wireless, including loopback).
- Select an Enhanced Distributed Channel Access (EDCA) traffic class to be used in both the client and the server. This choice can be of relevance in the vehicular use case, which uses EDCA in combination with the OCB mode and some specific EDCA parameters [6]. It is currently supported only when a patched kernel is available (such as the one included in the OpenC2X-Embedded platform [7] or in OpenWrt-V2X [8]).
- Choose the frequency and amount of request packets.
- Specify custom LaMP payload sizes and compare them.
- Use the experimental unidirectional mode supported by LaMP.

All the useful data can be logged onto a separate *.csv* file for further manipulation.

Furthermore, our tool can automatically compute the 90%, 95% and 99% confidence intervals, according to the Student's t-distribution, around the point average. This statistic is reported over the packets that are sent and received in a single test session, using a lookup table approach to increase the performance in low end devices.

As detailed below, our tests have proved, on the one hand, how LaTe can be used to measure the latency in distributed embedded Linux systems. On the other, that LaTe can be used as a basic measurement tool to characterize vehicular networking systems.

IV. Protocol and Tool Validation and Measurements

The tests we performed, summarized below, involved two different benchmarking setups. In the first one, we used two laptops, one mounting an Intel Dual Band Wireless-AC NIC and the other one a Qualcomm Atheros AR9460 NIC. The connectivity between the laptops was realized with the following options: (i) directly connected through an Ethernet crossover cable; (ii) connected through Ethernet with an 8-port 10BASE-2/T hub in between; (iii) connected through a 100 Mbit/s switch; (iv) communicating over Wi-Fi with a 5 GHz 802.11a access point in between, using a 20 MHz-wide channel, after verifying the absence of interference[1]. In the second test, we used two *PC Engines* APU1D embedded boards, mounting UNEX DHXA-222 WNICs and Realtek RTL8111 Ethernet Controller Cards, to recreate the same scenarios as before, with the following additional connectivity setups. (i) Directly communicating with 802.11p, over a 10 MHz wide channel and using OCB mode, as required by the standard [6]. Three physical data rates have been selected: 3 Mbit/s, 6 Mbit/s and 12 Mbit/s. Tests were performed both

[1]Tests were performed both by leaving the Linux rate adaptation algorithm as is (i.e., not trying to force any specific bitrate) and by selecting a data rate of 6 Mbit/s.

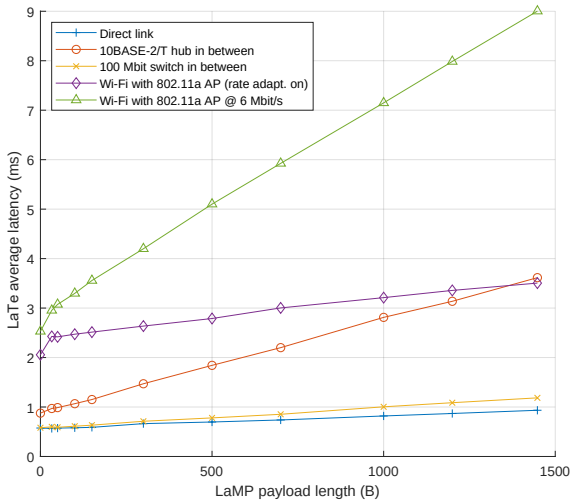

Fig. 2. Average user-to-user latency (RTT), using UDP and "SOCK_DGRAM" sockets, over direct Ethernet link, 10BASE-2/T hub, 100 Mbit switch and 802.11a, on the two laptops.

at a fixed distance, inside a laboratory, and outdoors, by varying the distance between the boards, from 0 m (i.e. the boards being placed very close to each other) to 190 m, using a transmission power of 18 dBm. (ii) Communicating with 802.11p, in OCB mode, selecting a LaMP payload size of 1448 B and using different EDCA Access Categories (ACs), with a parallel interfering traffic, generated by means of a patched version of the iPerf tool [8], capable of sending packets over different ACs. iPerf was set to generate a sizable interfering traffic, pushing data at 60% of the selected physical data rate.

The laptops were running Linux Mint 19.1, while the APU1D boards were instead executing a patched build of OpenWrt 18.06.1 [8] [9], in order to enable the communications over 802.11p. Every single test lasted for 60 seconds, over which the average, minimum and maximum latency values reported by LaTe were collected. The same measurement session was then repeated for different values of LaMP payload sizes, up to 1448 B, which is the maximum currently supported by LaTe, in order to never exceed the Ethernet MTU. As periodicity, we decided to send a request packet every 100 ms for the whole test duration (thus with a total of 600 packets for each test), which is also the maximum *Cooperative Awareness Message* (CAM) frequency foreseen by the ETSI standards for ITS-G5 [10]. The most important results are reported in Figures 2, 3, 4, 5 and 6.

Figure 2 and Figure 3 report some results we mainly used for validation purposes. In particular, the curves obtained for the latency tests involving the laptops are depicted in Figure 2. The values reported on the x axis represent the LaMP payload size: therefore, in order to obtain the full UDP payload size, 24 B have to be added, which represents the length of the LaMP protocol header. As expected, the higher is the transmitted payload, the higher is the observed RTT, because of the increase in transmission time. The increase is

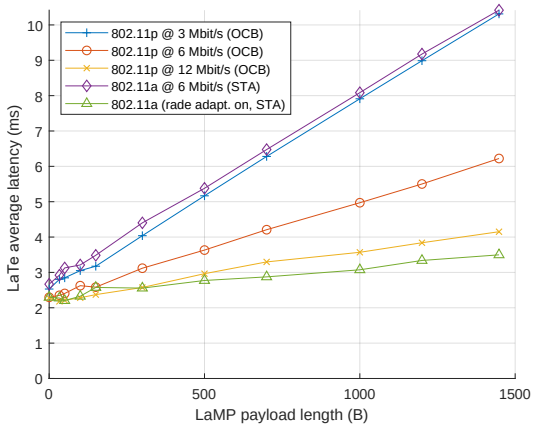

Fig. 3. Average user-to-user latency (RTT), using UDP and "SOCK_DGRAM" sockets, over 802.11p and 802.11a, on the two APU1D boards.

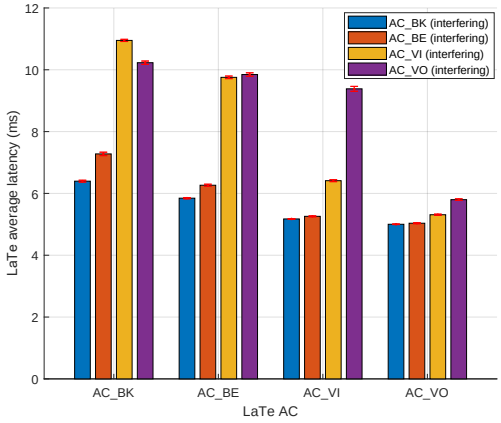

Fig. 4. Average user-to-user latency (RTT), using UDP and "SOCK_DGRAM" sockets, over 802.11p and different Access Categories, with a parallel interfering traffic, over a certain AC, generated by iPerf.

quite low when using a fast channel, i.e., a direct Ethernet connection or a 100 Mbit/s switch. Conversely, it is more evident, for instance, when using a 10BASE-2/T hub, which limits the maximum transmission speed over a shared medium. Finally, the results show how communicating over a wireless, contention-based, medium causes the latency to be higher.

Figure 3 compares the average user-to-user latency (RTT) measured with the APU1D boards under different configurations, when communicating over 802.11[2]. In particular, two scenarios have been considered: infrastructured IEEE 802.11a, in which the boards act as clients stations, with an additional Access Point (AP) to which they are connected, and its vehicular evolution, IEEE 802.11p, in which the boards directly exchange packets in OCB mode [6]. The LaMP payload size is shown on the x axis and the measured latency values, in milliseconds, are depicted on the y axis. The results are in line with expectations: starting from similar latency values for low payload sizes, its value increases linearly with the payload length, proportionally to the data rate associated with each modulation.

The collected data show the greater performance, in terms of latency, of 802.11p with respect to 802.11a, when similar data rates are used: this is evident when comparing the 802.11p curve at 6 Mbit/s with the 802.11a one, at the same data rate; having a direct communication reduces the measured latency, due to the fact that each packet can directly reach its destination, without the need of an additional hop through the AP. Interestingly, the 6 Mbit/s 802.11a curve proved to be very similar to the one related to 802.11p at 3 Mbit/s, i.e., half of the data rate. This is correct, since each request and reply, when using 802.11a, has to pass through the AP, doubling the device-to-device transmissions. The difference between the two can be interpreted as the AP computation time, showing how latency measurements can also be used to estimate internal network delays.

[2]The use of different hardware with respect to results in Figure 2 explain why homologous curves do not exactly match.

Figure 4 depicts the values obtained for the latency tests involving the communication over different EDCA traffic classes, using 802.11p at 12 Mbit/s. LaTe is set to send LaMP packets with 1448 B of payload, while iPerf sends 1470 B-long UDP datagrams. In this plot the average user-to-user latency (RTT), in milliseconds, is reported on the y axis for each AC used by LaTe. Every section of the bar plot is then divided into four distinct bars, each showing the traffic class used by iPerf to generate the parallel interfering traffic. Each single test, providing the average latency over 600 packets, was performed 15 times. The average value among these attempts was then considered as reference. The 95% confidence intervals around the average value are shown using error bars, even though they are quite small, as all the tests provided similar results.

The values we obtained are coherent with the theory, showing how increasing the interfering traffic AC causes the measured latency to increase as well, for each AC used by LaTe. Additionally, it is possible to highlight how using a high priority AC, such as AC_VO or AC_VI, allows the sending device to experience a lower latency as opposed to when AC_BK or AC_BE are selected. When using AC_BK, it was possible, however, to highlight a marginal decreasing trend, as iPerf was transmitting an interfering traffic over the *Video* and *Voice* traffic classes. This can be explained by observing that the channel usage is quite unbalanced towards iPerf, which generates a higher amount of prioritized traffic than LaTe.

Figure 5 shows instead the results we obtained when using LaTe to test the RTT ("user-to-user latency") between the two *PC Engines* boards, as they were placed at increasing distances, keeping them in line-of-sight. Each single 1 minute-long test was repeated 10 times, with a pause of 10 seconds between each test, and the resulting values were then averaged, computing also the 90% confidence intervals. The results show that the embedded boards can communicate at least until 190 m, with all the selected data rates, and with nearly a 0% packet loss in all the cases, as reported by LaTe. As the distance is increased, however, it is possible to notice

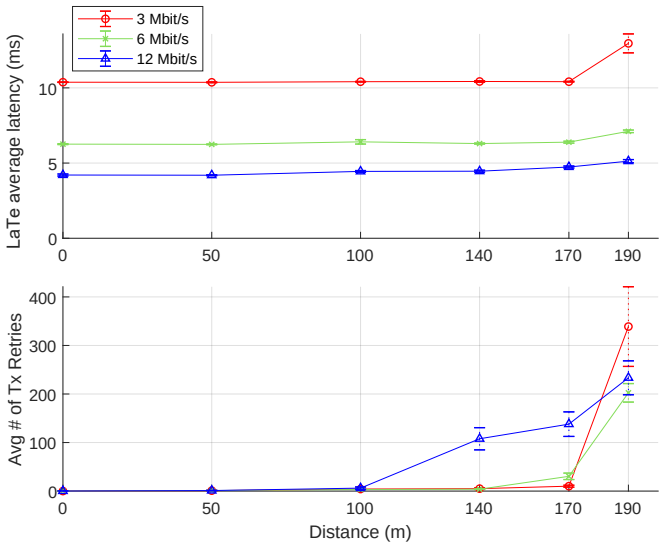

Fig. 5. Average user-to-user latency (RTT) and number of retransmissions, measured on the *PC Engines* boards, averaged over 10 tests, using UDP and "SOCK_DGRAM" sockets, over 802.11p and at different distances between them (outdoor tests). 90% confidence intervals are represented too.

an increase in the measured values, due to more frequent retransmissions caused by a lower received signal level. This is evident when looking at the *Tx Retries* curves, showing a correlation between the average number of retransmissions and the increased latency.

One additional test was then performed to compare the results obtained by means of `ping` with the ones gathered through our tool and by a cross-compiled version of `twping`, an open source implementation of TWAMP included in the perfSONAR project [11], as shown in Figure 6. The boards, communicating over 802.11p at a physical rate of 3 Mbit/s, were used. As mentioned in Section I, it is possible to notice how `ping` always reports a lower latency than LaTe, giving a more precise estimate of the network latency, but not of the application layer one, which is the latency a user would really experience when using UDP to communicate with a given application. Furthermore, this plot also compares the two latency types provided by LaTe, showing how using a *KRT latency* normally results in lower measured values, as the client-side time needed by the kernel (and then by LaTe itself) to handle each reply is not considered.

TWAMP instead provides comparable results with respect to the ones yielded by our tool, when the KRT latency is considered. This comparison can actually be performed as TWAMP test packets are encapsulated inside UDP, which is also the case of the LaMP packets managed by LaTe.

V. Conclusions

Network latency measurement tools are extremely important in the provisioning of reliable networked services.

In this paper we presented a new application layer protocol, LaMP (Latency Measurement Protocol), and the first application developed on top of it, i.e., LaTe (Latency Tester).

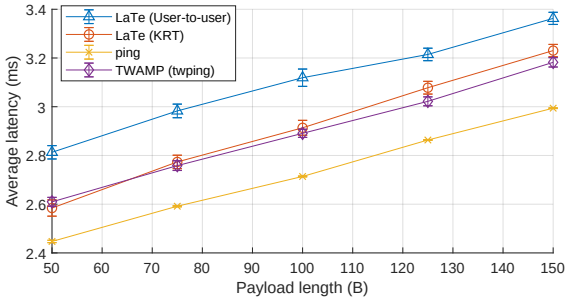

Fig. 6. Average user-to-user latency (RTT), measured on the *PC Engines* boards, averaged over 20 tests, using UDP and "SOCK_DGRAM" sockets, over 802.11p at 3 Mbit/s. 95% confidence intervals are represented too.

LaMP addresses the need of a simple protocol, completely agnostic of the communication sublayers, capable of carrying information useful to perform latency measurements. LaTe is a flexible client-server application relying on LaMP, that can be used to measure network latency in almost every linux-based communication system. LaMP and LaTe have been successfully tested with different setups and the results of the tests validated our tools by showing coherent results in every scenario, with a linear dependency between the packet payload length and the user-to-user latency.

Acknowledgment

This work has been performed in the framework of the EU Horizon 2020 project 5G-CARMEN co-funded by the EU under grant agreement No. 825012. The views expressed are those of the authors and do not necessarily represent the project. The Commission is not liable for any use that may be made of any of the information contained therein.

References

[1] D. Rossi Mafioletti *et al.*, "Metherxis: Virtualized Network Functions for Micro-second Grade Latency Measurements," in *ACM LANCOMM'16*, August 2016, pp. 22–24.

[2] K. Hedayat *et al.*, "A Two-Way Active Measurement Protocol (TWAMP)," Internet Requests for Comments, RFC Editor, RFC 5357, October 2018. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5357.txt

[3] J. Sherry, "Applications of the IP Timestamp Option to Internet Measurement," December 2010.

[4] LaTe + LaMP home page. [Online]. Available: https://francescoraves483.github.io/LaMP_LaTe/

[5] Linux programmer's manual. [Online]. Available: http://man7.org/linux/man-pages/man7/packet.7.html

[6] *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)*, pp. 1–3534, December 2016.

[7] S. Laux *et al.*, "Demo: OpenC2X - An open source experimental and prototyping platform supporting ETSI ITS-G5," in *2016 IEEE Vehicular Networking Conference (VNC)*, December 2016, pp. 1–2.

[8] GitHub - francescoraves483/openwrt-v2x. [Online]. Available: https://github.com/francescoraves483/OpenWrt-V2X

[9] F. Raviglione *et al.*, "Characterization and Performance Evaluation of IEEE 802.11P NICs," in *Proceedings of the 1st ACM MobiHoc Workshop on Technologies, mOdels, and Protocols for Cooperative Connected Cars*, ser. TOP-Cars '19. ACM, July 2019, pp. 13–18. [Online]. Available: http://doi.acm.org/10.1145/3331054.3331548

[10] "Intelligent Transport Systems (ITS); Vehicular Communications; Part 2: Specification of Cooperative Awareness Basic Service," *ETSI EN 302 637-2 V1.3.2 (2014-11)*, pp. 1–44, November 2014.

[11] perfSONAR Home. [Online]. Available: https://www.perfsonar.net/