Demo: Environment for Generic In-vehicular Network Experiments - EnGINE

Marcin Bosk[*,1], Filip Rezabek[*,1], Kilian Holzinger[1], Angela Gonzalez[2], Abdoul Kane[2], Francesc Fons[2], Zhang Haigang[2], Georg Carle[1], and Jörg Ott[1]

[1]Department of Informatics, Technical University of Munich, Germany
[2]Huawei Technologies Düsseldorf GmbH, Germany
[1]{bosk | rezabek | holzingk | carle | ott}@in.tum.de,
[2]{angela.gonzalez.marino | abdoul.aziz.kane | francesc.fons | zhanghaigang}@huawei.com

***Abstract*—Self driving cars bring new challenges to intra-vehicular networks (IVNs), such as increased bandwidth and the need for more powerful computation. To solve these challenges, in recent years, the IVNs started shifting towards the use of Ethernet, which with IEEE Time-Sensitive Networking standards can support deterministic behavior required for these networks. To evaluate the application of Ethernet in IVNs, we introduce novel approach called *EnGINE* - an *En*vironment for *G*eneric *In*-vehicular *N*etworking *E*xperiments. The environment is based on commercial off-the-shelf components and open-source software. In this work we present *EnGINE* and showcase an exemplary use-case it can support.**

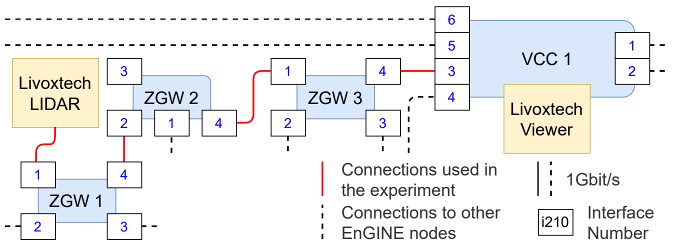

Figure 1: Excerpt of *EnGINE* network topology

I. Introduction

Autonomous driving and shared mobility are just two examples of modern trends within the automotive industry. These novelties are enabled by deterministic real-time communication in an intra-vehicular network (IVN). Due to rising demands on bandwidth, we start to see more Ethernet-based IVN solutions. Although Ethernet, by design, cannot offer deterministic behavior, with the Time-Sensitive Networking (TSN) family of standards it can provide real-time guarantees.

TSN has been a major research subject in recent years with most experiments being conducted using simulation. This approach has its advantages such as easy reproducibility and configurability. Still, it has many challenges, e.g., omission of clock-deviation and other real-deployment artifacts.

To combine the simulations' ease-of-use and behavior of real network deployments we introduce *EnGINE* [7] - an **En**vironment for **G**eneric **In**-vehicular **N**etwork **E**xperiments. Instead of using micro-controllers [4], *EnGINE* uses PCs which emulate zonal gateways (ZGWs) and vehicle control computers (VCCs). It builds upon the Linux networking stack and its queuing disciplines (qdisc) with initial focus on 802.1Qav, 802.1Qbv, and 802.1AS TSN standards [5] unlike previous work using OpenAvnu [8]. *EnGINE* is managed using a custom-built *Ansible*-based [1] orchestration tool which allows for configuration of the network and data sources/sinks. The setup follows recommendations of the AVNU Alliance and TSN Standards outlined in IEEE P802.1DG [6]. Furthermore, the framework enables extensive monitoring that can later be used for evaluation or traffic re-play. Using *EnGINE* the experiments are conducted autonomously, and can be easily repeated and reconfigured. Thanks to the framework's flexibility, novel IVN architectures can be developed.

[*]Marcin Bosk and Filip Rezabek contributed equally to this paper.

In this work, we show our design goals and their realization for an easily configurable and flexible IVN experiment infrastructure. Furthermore, we introduce one of the IVN use-cases and demonstrate how it can be evaluated using *EnGINE*.

II. System Design

To provide a framework for IVN research, we follow the ACM Policy [2] and define additional requirements for our system: repeatability, reproducibility, and replicability. With those requirements in mind, we build *EnGINE* using commercial off-the-shelf (COTS) network interface cards (NICs). More specifically, the 1 Gbit/s Intel® I210 NICs, which are used to interconnect our Linux-based ZGWs and VCCs as outlined in Fig. 1. All nodes utilize standard PC components. *Open vSwitch* is used to prepare the network topology for each experiment and forward traffic.

To orchestrate *EnGINE*, we built a framework using *Ansible* [1], an idempotent, descriptive language using YAML and Jinja. Fig. 2 shows a typical workflow, where the management host remotely executes commands on a node ①. Then ② respective node runs this code and ③ interacts with others. Next, the nodes store the collected artifacts on the management host ④, where the collected artifacts are processed ⑤.

Each experiment campaign consists of four phases: **install**, **setup**, **scenario**, and **process**. In **install**, the nodes required for the campaign are allocated and booted with an operating

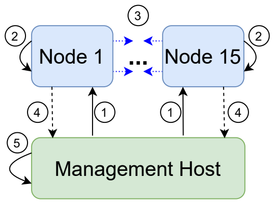

Figure 2: Communication workflow

system using *pos* [3]. **Setup** follows with installation of dependencies required for the experiments. Also, scripts and other prerequisites are copied from the management node. Individual experiments are performed in **scenario** phase. After each experiment, the generated artifacts are collected and transferred to the management node. The campaign is concluded with post-**processing** where those artifacts are evaluated and initial plots are generated.

III. Experiment Setup

To show the capabilities of *EnGINE* we define a use-case where a LIDAR trasmits a point-cloud to one of the VCCs. For that, we use a Livoxtech LIDAR Mid40 connected to ZGW1 that sends data towards the VCC1 where a Viewer application outputting a live picture of the point-cloud is placed. The LIDAR sends approx. $1000\,\mathrm{packet/s}$ (depending on configuration), each with a payload of $1318\,\mathrm{B}$ for a total of $1388\,\mathrm{B}$ on the physical layer (PHY). The data is transferred over the network consisting of ZGW2 and ZGW3 placed in a line topology between ZGW1 and VCC1 as shown in Fig. 1.

We configure the network for two experiment types. The first one uses the 802.1Qav Credit-Based shaper (CBS) qdisc configured for the PHY bandwidth of $11\,104\,\mathrm{kbit/s}$ which corresponds exactly to that of the Livoxtech LIDAR. The second one utilizes the 802.1Qbv Time-Aware shaper (TAS/TAPRIO) qdisc and Earliest Time First (ETF) as a child qdisc configured in the deadline mode. We calculated the TAPRIO window sizes and cycle time corresponding to the link speed and packet spacing, which is approx. $10\,\mu\mathrm{s}$ and $100\,\mu\mathrm{s}$ respectively. The qdiscs are configured on each node and interface of the used topology. Furthermore, in each experiment we introduce competing cross-traffic that follows the same path as LIDAR traffic. The testbed also allows the competing traffic to follow any other available path. In case no qdiscs are configured, the time guarantees for the LIDAR traffic are not met and due to processing delay on each hop we see high fluctuation. This behavior is resolved by using properly configured CBS and TAPRIO along the path.

IV. Demonstration

Section III outlines the experiment performed using *EnGINE* deployed on COTS hardware as presented in Fig. 3. The Livoxtech LIDAR is connected to ZGW1 (right) and Livoxtech Viewer displayed on a monitor connected to VCC1 (left).

For each experiment we obtain a live point-cloud preview as shown on the monitor in Fig. 3. Furthermore, after each experiment the data transmitted by the LIDAR is analyzed

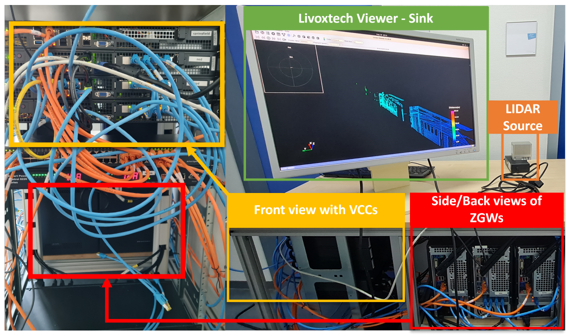

Figure 3: *EnGINE* hardware deployment

and the end-to-end delay, jitter, and throughput are visualized. We observe that the misconfigurations of CBS and TAPRIO result in the live output of the LIDAR starting to lose frames and thus signify the importance of proper shaper configuration for mission critical traffic.

V. Conclusion

In this paper, we introduce *EnGINE*, a framework for repeatable, reproducible, and replicable IVN and TSN experiments. The framework is built using open-source solutions coupled with COTS hardware and supports TSN standards as recommended by P.802.1DG [6]. Furthermore, we show that *EnGINE* can support IVN experiments, demonstrating a LIDAR point-cloud data subjected to various network conditions. Naturally, the framework inherits some challenges from its open-source nature, for example the inherent complexity of the Linux kernel or of other utilized software artifacts. We overcome those challenges with our framework structure and implementation to ensure real-time system performance. As the framework is still in development, the source code is not yet publicly available and will be published in the future.

References

[1] *Ansible is Simple IT Automation*. https://www.ansible.com.
[2] *Artifact Review and Badging - Current*. https://www.acm.org/publications/policies/artifact-review-and-badging-current.
[3] S. Gallenmüller et al. “High-performance packet processing and measurements”. In: *2018 10th International Conference on Communication Systems Networks*. 2018, pp. 1–8.
[4] J. Hwan Seo and J. W. Jeon. “Comparison of IEEE802.1Q and IEEE802.1AVB in multi switch environment in embedded system”. In: *2017 17th International Conference on Control, Automation and Systems (ICCAS)*. 2017.
[5] “IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks”. In: *IEEE Std 802.1Q-2018* (2018), pp. 1–1993.
[6] D. Pannell et al. *Use Cases - IEEE P802.1DG V0.4*. https://www.ieee802.org/1/files/public/docs2019/dg-pannell-automotive-use-cases-0919-v04.pdf. Sept. 2019.
[7] F. Rezabek et al. “EnGINE: Developing a Flexible Research Infrastructure for Reliable and Scalable Intra-Vehicular TSN Networks”. In: *2021 3rd International Workshop on High-Precision, Predictable, and Low-Latency Networking*. 2021.
[8] T. A. Xu et al. “Poster: Performance Evaluation of an Open-Source Audio-Video Bridging/Time-Sensitive Networking Testbed for Automotive Ethernet”. In: *2018 IEEE Vehicular Networking Conference (VNC)*. 2018.