# Cross-Validating Open Source In-Vehicle TSN Simulation Models With a COTS Hardware Testbed

Darinela Andronovici*, Ion Turcanu*, Jannusch Bigge†, and Christoph Sommer†
*Luxembourg Institute of Science and Technology (LIST), Luxembourg
†TU Dresden, Faculty of Computer Science, Germany
{ darinela.andronovici, ion.turcanu }@list.lu
https://www.cms-labs.org/people/ { bigge , sommer }

***Abstract*—We tackle the challenges of accurately replicating modern automotive network architectures, particularly those reliant on automotive Ethernet and Time-Sensitive Networking (TSN), in simulation environments. We describe an open-source TSN simulation model tailored to match a real-world TSN testbed built using off-the-shelf hardware. We address key challenges such as hardware and software imperfections as well as varying data traffic patterns from automotive sensors. By incorporating real data traces from LIDAR and camera sensors, we improve simulation accuracy. Our findings underscore the importance of accounting for hardware and software nuances to ensure faithful simulation results, thus advancing the reliability of automotive network simulations.**

## I. Introduction

In-car networking is currently undergoing a radical shift [1], away from numerous dedicated buses and sub-buses connecting dedicated Electronic Control Units (ECUs); instead, in-car networking is moving to modern architectures like a Time-Sensitive Networking (TSN) based communication backbone running automotive Ethernet. This shift to automotive Ethernet is a central enabler [2] for consolidation of advanced vehicle functions and for vehicular cloud computing. In-car networking thus mirrors the trend of a wide array of applications from smart manufacturing [3] to spacecraft industries [4], [5]. Beyond this, research is also exploring the potential of TSN for enabling Vehicle to everything (V2X) communication over both guaranteed-service [6] and best-effort wireless networks [7].

The associated rise in network and scheduling complexity, however, means that the design and validation of in-car networks is becoming increasingly challenging [8] making simulation the new default approach to parameterization [9]. Thus, research and the automotive industry alike are increasingly looking towards digital twinning [10] for system design and validation.

Such a digital twin can be provided only by such computer simulations that are demonstrably able to accurately model the behavior of the in-car network. This, in turn, requires a careful parameterization of the simulation model and the use of realistic input data traces, as we demonstrate in this paper. Openly available data traces are, however, rare, and the parameterization of the simulation model is heavily dependent on the specific hardware and software configuration of the in-car network.

In this paper we fill this gap by:

- providing a detailed description of a publicly available TSN simulation model and configuration,
- matched to an easily assembled real-world TSN testbed composed of Commercial Off-the-Shelf (COTS) hardware.
- We also demonstrate the importance of realistic input data traces for the simulation model; and
- provide such traces for a camera and a LiDAR sensor.[1]

The remainder of this paper is structured as follows. In Section II we discuss related work in terms of all of TSN simulation, TSN testbeds, and digital twinning. In Section III we describe how off-the-shelf hardware is assembled into the real-world TSN testbed underlying this paper and how the testbed is configured. In Section IV we describe the publicly available TSN simulation model used in this paper. In Section V we discuss important aspects of the parameterization of the simulation model to accurately match the testbed. In Section VI we discuss input data modeling and the importance of realistic input data traces. The paper concludes with a summary given in Section VII.

## II. Related Work

TSN is a collection of standards being developed by the IEEE 802.1 TSN Task Group to enable deterministic real-time communication over Ethernet networks [11]. These standards define features that enable synchronization, bounded latency, communication reliability, and resource management. The growing need to provide deterministic communication in various industries and application domains has led to recent efforts by IEEE and 3GPP to extend the capabilities of wired TSN to the wireless domain [12]. In this context, evaluating the performance of end-to-end TSN systems with different levels of complexity is of utmost importance. Designing an accurate TSN evaluation framework presents numerous challenges, such as supporting precise timing requirements, capturing real-world network variability, and providing scalability and flexibility.

A common approach to validating TSN systems is to evaluate their performance on real hardware testbeds under realistic conditions. In a recent survey, Senk et al. [13] provide an overview of open source projects aimed at low-cost development and evaluation of integrated 5G-TSN architectures in a

[1]Available at https://github.com/LIST-LUXEMBOURG/vnc2024-traces



IEEE Vehicular Networking Conference (VNC 2024)

general context. In particular, the authors describe the current standardization status of 5G-TSN integration, present existing open-source 5G systems, and discuss available hardware for open-source 5G and TSN integration. An earlier work by Quan et al. [14] proposes OPENTSN, a generic open-source TSN testbed that allows prototyping and customization of TSN systems on Field Programmable Gate Arrays (FPGAs). The proposed system includes two main components, *TSNSwitch* and *TSNNic*, and supports several key features such as a Software-Defined Networking (SDN)-based network control mechanism, a time-sensitive management protocol, and a time-sensitive switching model.

Senk et al. [15] describe a testbed design that uses open source software and COTS hardware to measure the performance of generic TSN networks. The testbed design is publicly available and is used to gain insight into the performance of real-world TSN devices. In an extended version of this work, Ulbricht et al. [16] further analyze critical aspects of TSN with multiple stream sets and identify challenges in configuring the Gate Control List (GCL), especially for traffic with variable packet sizes.

Miranda et al. [17] evaluate the suitability of two cloud-based testbeds for TSN experimentation. They focus on key features supporting time synchronization, traffic scheduling, and filtering. Moreover, they also introduce a prototype SDN controller for TSN Centralized Network Configuration (CNC) and implement crucial modules for TSN network deployment and management.

Focusing on the automotive domain, Xu et al. [18] describe a TSN testbed design using COTS hardware and open-source software, and use it to evaluate the performance of TSN in terms of latency and jitter. Bosk et al. [8] also propose a methodology for evaluating TSN performance in the context of in-car networks. The proposed methodology, which is based on the ENGINE framework [19], can also be generalized to other domains. However, the experimental evaluation only considers synthetically generated periodic traffic. Also, the engine only supports software timestamping, which does not meet the synchronization accuracy required by most TSN setups.

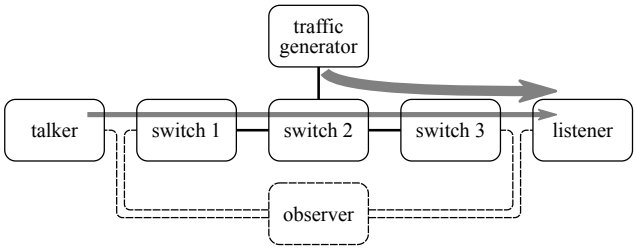
Despite the clear advantages in terms of real-world network performance evaluation, TSN validation in hardware testbeds remains a challenging task due to hardware cost, scalability, and deployment complexity. For this reason, simulation-based approaches have gained popularity by enabling rapid evaluation at scale. For example, Falk et al. [20] present a TSN simulation framework based on OMNeT++ for converged time-triggered and best-effort traffic. This framework, which includes several TSN simulation components such as Strict Priority Scheduling (SPS), a Time-Aware Shaper (TAS), a Credit-Based Shaper (CBS), and frame preemption, has been evaluated under different configuration settings. Zhou et al. [21] evaluate the performance of various TSN traffic shaping and scheduling mechanisms in a fully simulated automotive Ethernet context. A simulation-based approach to analyzing the potential of integrating multiple in-car wired TSN networks over best-effort wireless links for V2X communications is proposed in [7]. Here, the authors focus on a vehicular platooning use case and demonstrate the benefits of synchronizing the TSN schedulers of the platoon members in terms of end-to-end delay. However, these publications do not reveal the degree to which these have been validated in real TSN testbed environments.

Few TSN simulation papers are co-published with a treatment of validation in hardware. One example is the simulation model developed by Jiang et al. [22] in OMNeT++ and validated on a TSN hardware testbed. The proposed simulation model consists of four modules: (i) a configuration module, which defines the GCL; (ii) a TAS module, which executes the GCL configuration; (iii) a queue module, which stores and distributes the incoming traffic into different priority queues; and (iv) a transmission module, which simulates the actual data transmission on the physical media. Validation with the TSN hardware testbed demonstrates the accuracy of this simulation model in terms of end-to-end delay. However, the authors only considered synthetic data with a fixed transmission interval, which does not emulate a real-world application.

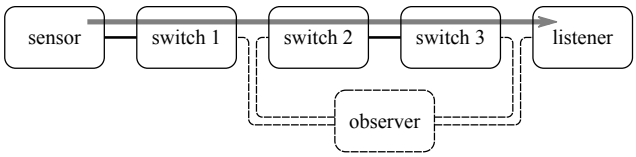
Another example is the experimental approach proposed by Bosk et al. [23], which allows mirroring the hardware-based TSN configuration in a simulation environment. Specifically, the authors extend the ENGINE framework [19] to support a simulation environment based on OMNeT++ that replaces the original hardware deployment. To this end, they describe a methodology that allows the ENGINE scenario configuration and physical hardware deployment to be replicated in OMNeT++. The simulation results are compared with the testbed experiments and generally show lower delays in simulation than in hardware experiments for both CBS and TAS. The authors conclude that this discrepancy is due to the fact that the processing delay in the hardware is not modeled in the simulation. We confirm their findings in our experiments and investigate them further, using more realistic data sources and gaining more insight into the importance of using real data traces to configure the simulation environment.

Karle et al. [24] present a holistic platform for autonomous driving research that includes the EDGAR research vehicle and its digital twin. The autonomous vehicle is equipped with a high-performance network switch that supports IEEE 802.1Qav and Precision Time Protocol (PTP)-based time synchronization. While time constraints are not currently guaranteed, the proposed system design allows for such guarantees using the TSN capabilities of the switch. To create the digital twin of the network, the authors rely on the extended ENGINE framework, which includes the OMNeT++ simulation tool [23]. While this solution introduces the first openly available holistic digital twin of an autonomous vehicle, it may be too complex and expensive for researchers working on novel TSN-based in-car network architectures to adopt it as their lab prototype.

In this paper, we focus specifically on in-car networks and describe a methodology for configuring and cross-validating an open-source TSN simulation model using a more accessible, COTS hardware testbed. Our proposed solution thus enables researchers to develop a digital twin of automotive TSN networks with many different levels of complexity.




(a) model parameterization experiments




(b) input data modeling experiments

Figure 1. Network topologies of the testbed.

## III. Testbed Configuration

Figure 1 illustrates the network architecture of the testbed used in the experiments. It models the commonly employed hub-and-spoke design with data streams passing through a central bridge [25]. Many of the components are *Relyum*[2] branded devices manufactured by *SoC-e* as listed in the following.

The testbed contains three switches with TSN bridge capabilities: *Switch1*, *Switch2*, and *Switch3* (Rely-Tsn-Bridge v20.1.11, Rely-Tsn-Bridge+/12 v22.2.0, Rely-Tsn-Bridge v22.3.0, respectively). An optional *Traffic Generator* (Rely-Traf-Gen v20.1.0) is connected to *Switch2* to generate interfering traffic. The *Observer* (Rely-Tsn-LAB v.21.1.1c2) is a transparent device with test capabilities to measure the end-to-end delay and bandwidth of a device or TSN network segment under various test conditions. The hardware components are connected with 1 Gbit/s Cat5 Ethernet cables, each 1 m long. Figure 2 illustrates the setup.

Virtual Local Area Networks (VLANs) are implemented to match real-world deployments for performance, security, and network management. Each VLAN sees only the traffic that is intended for it. In the context of TSN, VLANs allow for the efficient separation of traffic streams with different quality of service requirements. Bridges use the VLAN priority information to prioritize and manage network traffic. The TSN components are easily configurable through their internal web server, the *Web Manager*. The tool allows configuration of various hardware parameters, including the stream processing rules at each port and the supported TSN standards.

In our testbed, we use two key TSN features: time synchronization and traffic shaping. Time synchronization is performed using IEEE 802.1AS-2020, also known as the Generalised Precision Time Protocol (gPTP). Within the gPTP domain, all time-aware systems synchronize their clocks with a single

[2]https://www.relyum.com/

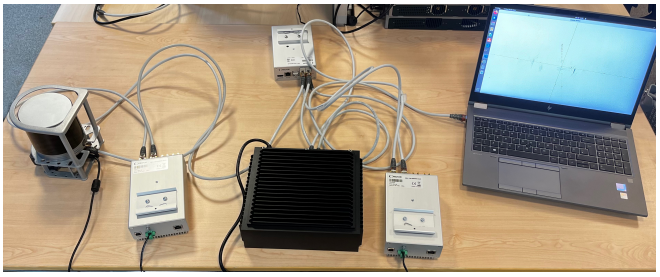


Figure 2. Photo of the testbed. From left to right: LiDAR sensor, auxiliary *Switch1*, central *Switch2* (black), auxiliary *Switch3*, *Listener*; top: *Observer*.

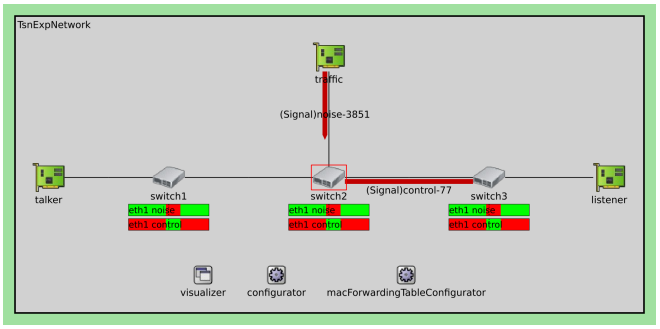



Figure 3. Screenshot of the simulation; components arranged and labeled to represent the physical topology.

Grandmaster (either a bridge or an end station [26]). Traffic shaping is performed using the IEEE Std 802.1Qbv Time-Aware Shaper (TAS). The TAS schedule configuration is based on assigning a time slot to one or multiple of the eight Ethernet priorities, which are encoded as Priority Code Points (PCPs) in the Ethernet frame header.

## IV. Simulation Framework

The simulation uses OMNeT++ version 6.0.1 as a discrete event simulator in combination with INET version 4.5.2 as a network simulation framework. INET provides a basic end node (*TsnDevice*) and a bridge node (*TsnSwitch*) for TSN with preconfigured submodules. It also provides a dedicated clock node (*TsnClock*) that can be used as a gPTP master clock and is also TSN capable. The nodes do not have any TSN features enabled by default, but each node has multiple parameters that can be used to enable them.

The simulated network consists of three end nodes (of type *TsnDevice*): *Talker*, *Listener*, and *Traffic Generator*, as well as three bridge nodes (of type *TsnSwitch*) to connect them. All network nodes are connected via an *Eth1G* channel with each channel set to a length of 1 m. The network topology is shown in Figure 3 and is identical to that of the testbed – except for the absence of the *Observer*, whose role is fulfilled by logging data directly from the simulation.

INET uses the stream classification and identification feature to add VLAN tags to the outgoing frames. Each frame is assigned to a stream based on specific criteria in the bridging module of the talker. VLAN tagging is then performed by

Table I
TRAFFIC CONFIGURATION FOR THE MODEL PARAMETERIZATION EXPERIMENTS.

| Traffic Type | PCP | Frame Size | Inter-Frame Gap |
|---|---|---|---|
| Control data | 4 | 300–1500 Byte, step 300 | 1 ms |
| Noise data | 2 | 1500 Byte | 13.33, 20, and 40 µs |

Table II
TAS CONFIGURATION FOR THE MODEL PARAMETERIZATION EXPERIMENTS.

| slot | duration | PCP0 | PCP1 | PCP2 | PCP3 | PCP4 | PCP5 | PCP6 | PCP7 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 100 µs | · | · | · | · | ✓ | · | ✓ | · |
| 2 | 450 µs | ✓ | ✓ | ✓ | ✓ | · | ✓ | ✓ | ✓ |

the stream classification in the bridging module of the talker based on the stream to which the frame belongs. To correctly tag and interpret incoming frames, it is necessary to enable the incoming and outgoing *stream* feature in the end nodes. INET uses the concept of *tags* to add additional information to frames in upper layers to inform lower layers to add them. Therefore, it is necessary to exclude the VLAN tag from the *PacketDirectionReverser* module in the bridge nodes to prevent loss of tags.

To use a TAS in the bridge nodes, the *hasEgressTrafficShaping* parameter must be enabled. This replaces the default queue module with the *Ieee8021qTimeAwareShaper* module. By default, no scheduling is performed and all queues are always open. Scheduling can be defined manually or an automatic scheduler can be used. To ensure consistency between the simulation and the testbed, scheduling is done manually. To define the scheduling, one must define the cycle time, the slot time for each gate, and the offset. Each queue is initially open by default. Each slot includes the guard band. The mapping of the PCP to the gate is based on the "recommended priority to traffic class mappings" table of IEEE 802.1Q [27, page 217, table 8-5], but can be modified to meet the network requirements.

The gPTP module must be configured for time synchronization. INET provides a basic gPTP module that can act as a master or slave clock with modifiable sync and delay measurement intervals. To simulate clock drift, the parameterizable *ConstantDriftOscillator* module can be used for each clock provided by INET.

In the sink module, there is one *UdpSinkApp* for each type of traffic in the network, each listening on a different port. The application on the *Talker* depends on the experiment.

## V. MODEL PARAMETERIZATION

### A. Testbed

For the model parameterization experiments, the *Talker* and *Listener* of the testbed are off-the-shelf Linux machines used to send and receive data.

Two types of traffic are used in this set of experiments, as shown in Table I. The first type of traffic is time-sensitive control traffic. The *Talker* is configured to generate Layer 2 control traffic with a specific frame size, VLAN ID, PCP, inter-frame gap, and outbound VLAN interface. The control traffic has a PCP of 4 and an inter-frame gap of 1 ms. Each experiment consists of sending 20 000 frames through the TSN network and to the *Listener*. In addition, several experiments are performed with different frame sizes for the control traffic: 300, 600, 900, 1200, and 1500 Byte.

The second type of traffic is the congestion traffic (hereafter referred to as *noise*) which is transmitted by the *Traffic Generator*. This traffic is configured with a fixed frame size of 1500 Byte and a PCP of 2. In addition, this traffic is configured to have different *noise levels*, i.e., to occupy different portions of the available bandwidth: 30 %, 60 %, and 90 %. The absence of noise is also considered as part of the study. This is done in order to observe if there is any impact on the control traffic.

Regarding the TSN mechanism, we use a simple TAS configuration consisting of two slots, as shown in Table II. The first slot is for the control and gPTP traffic and the second slot is for all eight priority queues except the one with PCP 4. The length of the first slot is 100 µs, the length of the second slot 450 µs, for a total cycle time of 550 µs.

### B. Simulation

The testbed setup is reproduced in simulation by configuring the *Talker* as a standard *UdpSourceApp*. It produces User Datagram Protocol (UDP) datagrams at a configurable interval. The datagram length is configured to match the frame lengths used in the testbed experiment (accounting, of course, for header lengths).

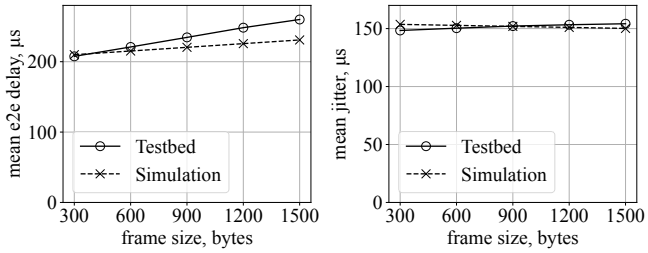
The *Traffic Generator* uses a standard *UdpSourceApp* from INET to produce UDP datagrams with a length of 1446 Byte, that is, Ethernet frames with a length of 1500 Byte. The sending interval depends on the data rate of the experiment. For 30 %, it is 40 µs, for 60 % it is 20 µs, and for 90 % it is 13.33 µs. For the case with 0 % noise, an offset beyond the end of the simulation time is configured to effectively disable the *Traffic Generator*.

Frames are classified based on the destination source port and mapped to either a control or noise traffic stream. As in the testbed, the control traffic stream receives PCP 4, while the noise traffic receives PCP 2. Each outgoing Ethernet port has two traffic classes per queuing module. The first class is for noise traffic and the gate opens for 450 µs with an offset of 100 µs into a 550 µs cycle. Therefore, the other gate governs the control traffic and remains open for 100 µs with no offset, followed by 450 µs of being closed.

For time synchronization, *Switch2* is chosen as the master node, with *Switch1* and *Switch3* as slave nodes. All other parameters are left at their default values.

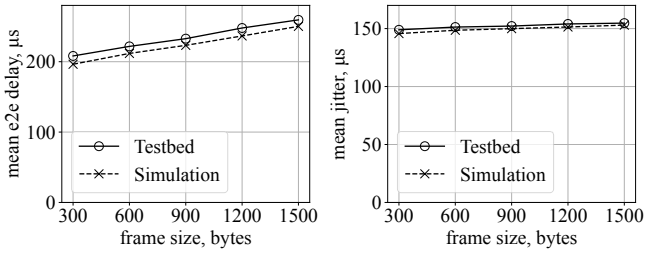
### C. Result Analysis

We now analyze the numerical results obtained from the model parameterization experiments in the TSN testbed and compare them with those obtained from the simulation. The metrics studied are the *end-to-end (e2e) delay* and *jitter* under various combinations of noise level and frame size. To measure



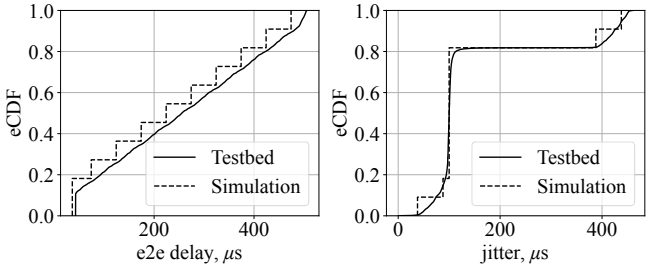

(a) end-to-end (e2e) delay (b) jitter

Figure 4. (Non-) matching metrics for different frame sizes and 90 % noise.



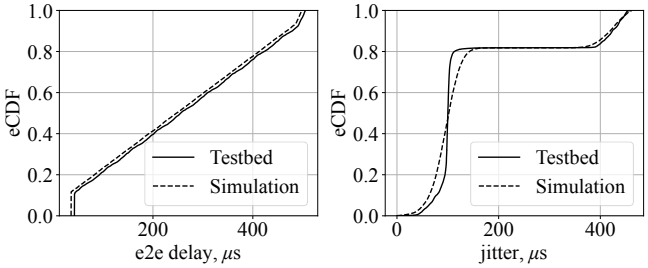

(a) e2e delay (b) jitter

Figure 6. Like Figure 4, but after inter-frame gap distribution matching.




(a) e2e delay (b) jitter

Figure 5. (Non-) matching eCDFs of e2e delay and jitter for 1500 Byte frames and 90 % noise.




(a) e2e delay (b) jitter

Figure 7. Like Figure 5, but after inter-frame gap distribution matching.

the e2e delay, the *Observer* records a timestamp each time a frame starts to be transmitted by the *Talker* and *Switch3*. The difference between these two values is recorded as the e2e delay per packet. For the jitter, we calculate the difference in e2e delay between two consecutive received frames.

Figure 4 illustrates the mean e2e delay and jitter for different frame sizes and for a noise level of 90 %. Note that, as described, we performed the experiments for multiple noise levels, but only show the results for 90 % because we confirmed that – as expected – the level of noise has no substantial bearing on either the e2e delay or the jitter. Only using a larger frame size does increase the e2e delay. As can be seen, a noticeable discrepancy between simulation and testbed results can be observed.

Figure 5 helps to illustrate the underlying reason via empirical Cumulative Distribution Functions (eCDFs) of e2e delay and jitter for a frame size of 1500 Byte: The operating system of the *Talker* in the testbed does not guarantee a fixed production interval, leading to randomness in the inter-frame gap. In order to reproduce this behavior in the simulation, we analyzed the distribution of the actual inter-frame gap in the testbed and noticed that it conforms closely to a normal distribution. Therefore, we run another set of simulations where the UDP datagrams are produced at an interval chosen from a normal distribution with a mean of 1 ms and a standard deviation of 22.8 µs based on measurements of the testbed hardware.

Both effects mentioned above are captured very well by both the testbed and the simulation, as can be seen in Figure 6. Figure 7 further confirms these findings via the eCDFs of e2e delay and jitter when using a frame size of 1500 Byte for the control traffic and a noise level of 90 %. Again, we only show the eCDFs for this particular configuration because the variation in frame size and noise level has very little effect on these distributions.

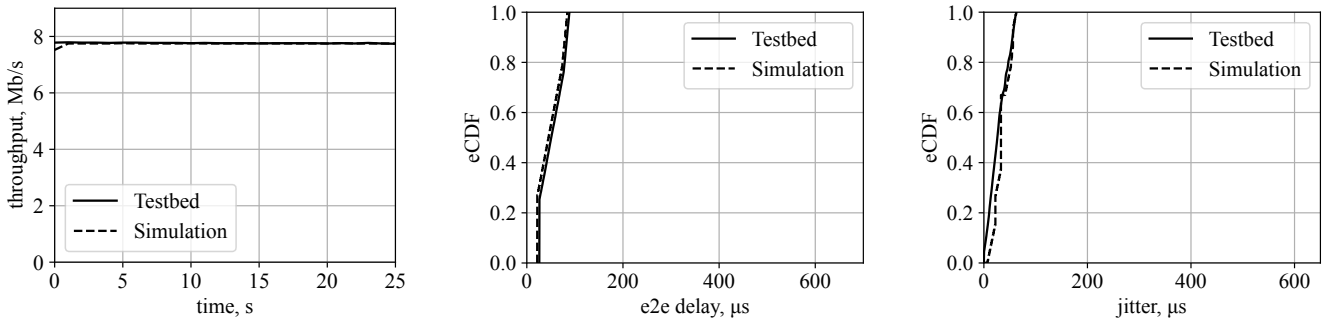
The only remaining difference between the simulation and testbed results is a constant offset in the mean e2e delay of about 7 µs and in the jitter of about 3 µs. We speculate that this is due to the processing time and hardware imperfections, which are not modeled in the simulation, paralleling the findings of Bosk et al. [23], who identified a similar discrepancy between simulation and hardware experiments and also concluded that this is due to the processing delay that is not modelled in software. This discrepancy could be reduced by adding a hardware-dependent processing delay component to the simulation model. However, even without such hardware-dependent tuning, we can conclude that for the selected COTS components, the simulation model can be brought into good agreement with measurements from the testbed.

## VI. Input Data Modeling

### A. Testbed

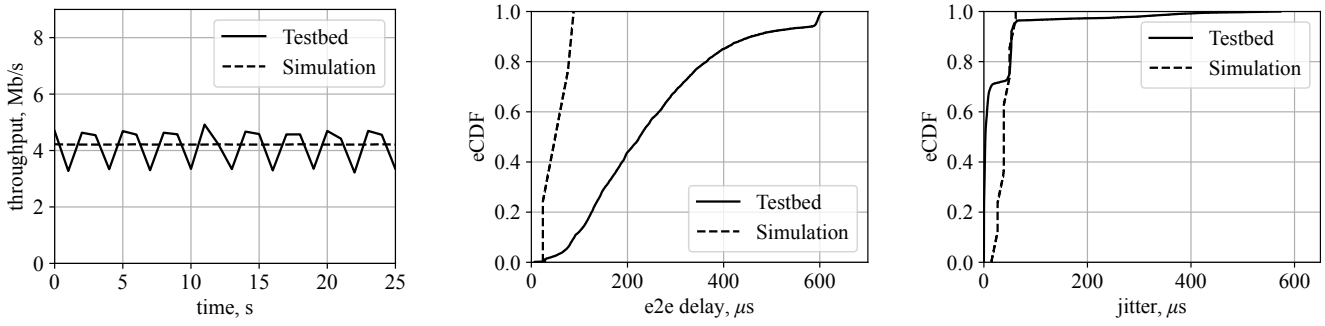
The input data modeling experiments use real sensors as *Talker* nodes that might be part of a vehicular network. Therefore, the Linux desktop acting as the *talker* is replaced with a *Sensor* – either a LiDAR or a camera. In this setup, *Switch1* is used to add the VLAN tag in the Ethernet frames sent by the sensors. As a result, the *Observer* monitors traffic on the smaller network segment between *Switch1* and *Listener*.

The first sensor option is a *Robosense Helios 16* high-precision 3D LiDAR using 16 laser beams. This device uses



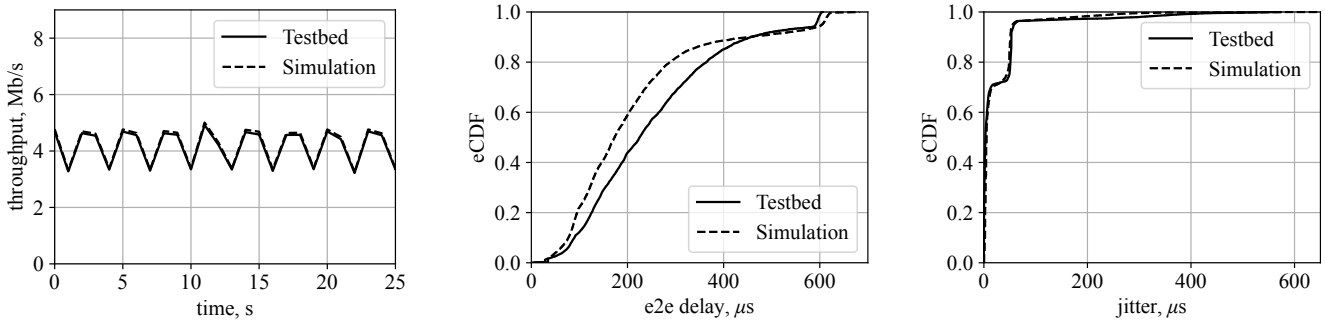

(a) throughput (b) end-to-end (e2e) delay (c) jitter

Figure 8. Matching metrics of the LiDAR sensor between testbed and simulation.




(a) throughput (b) end-to-end (e2e) delay (c) jitter

Figure 9. (Non-)matching metrics of the camera sensor between testbed and simulation when modeled as a constant bitrate source: Simulated delay is too low, jitter too high.




(a) throughput (b) end-to-end (e2e) delay (c) jitter

Figure 10. Matching metrics of the camera sensor between testbed and simulation when modeled using a real data trace: The distributions of both the delay and the jitter are now captured more closely by the simulation.

Ethernet as data transmission protocol for in-vehicle use and supports time synchronization methods such as GPS, PTP, and gPTP. In more detail, the LiDAR sends data via Ethernet using UDP datagrams. This data uses both the Main Data Stream Output Protocol (MSOP) – which contains point cloud data (laser scanning data, including distance, angle, and reflection intensity) – and the Device Information Output Protocol (DIFOP) – which outputs various configuration information about the current state of the LiDAR.

The second sensor option is a computer (*Jetson AGX Orin*) streaming video from a *Logitech C920 HD PRO* camera supporting 1080p video at 30 Frames Per Second (FPS) with autofocus and light/color correction. As the camera is connected via USB, we use the computer to encode and transmit the video signal via UDP. For this, the live camera capture is streamed over the network utilizing the GStreamer framework. In more detail, the GStreamer graph captures video from the Video for Linux API version 2 (V4L2) device of the camera at a resolution of 1280x720 and 10 FPS, burning in timecode information. It then uses the NVIDIA *nv v4l2 h264enc* H.264 encoder block with its Instantaneous Decoder Refresh (IDR), Sequence Parameter Set (SPS), and Picture Parameter Set (PPS)

Table III
TRAFFIC CONFIGURATION FOR THE INPUT DATA MODELING EXPERIMENTS.

| Traffic Type | PCP | Payload Size | Mean Data Rate |
|---|---|---|---|
| LiDAR data | 4 | 1248 Byte | 7.65 Mbit/s |
| Video data | 3 | 16–1400 Byte | 4 Mbit/s |

Table IV
TAS CONFIGURATION FOR THE LIDAR.

| slot | duration | PCP0 | PCP1 | PCP2 | PCP3 | PCP4 | PCP5 | PCP6 | PCP7 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 48 µs | · | · | · | · | ✓ | · | · | · |
| 2 | 52 µs | ✓ | ✓ | ✓ | ✓ | · | ✓ | ✓ | ✓ |

Table V
TAS CONFIGURATION FOR THE CAMERA.

| slot | duration | PCP0 | PCP1 | PCP2 | PCP3 | PCP4 | PCP5 | PCP6 | PCP7 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 48 µs | · | · | · | ✓ | · | · | · | · |
| 2 | 52 µs | ✓ | ✓ | ✓ | · | ✓ | ✓ | ✓ | ✓ |

interval set to 15 frames to allow fast (re-)synchronization to the stream. It then payload-encodes the H.264 video into Real-time Transport Protocol (RTP) packets, streaming these via UDP to the *Listener*. The result is a variable-bitrate video stream, as is currently being investigated for automotive contexts in the scientific literature [28].

Table III summarizes information on the traffic priorities, Ethernet payload sizes, and the mean transmission data rate of both sensors. Similar to the previous set of experiments, a simple TAS configuration with two slots is used. The cycle and slot duration are the same for both sensors. The first slot is used for the *Sensor* traffic and the second slot is used for the rest of the traffic except the *Sensor*. Tables IV and V show a detailed configuration of the scheduling mechanism.

### B. Simulation

To simulate the sensors, the *Talker* application remains the same, *UdpSourceApp*. We configure the *Talker* to generate frames with a fixed inter-frame gap and frame size to match the average values of inter-frame gap and frame size as measured in the testbed. Although the documentation of the LiDAR seems to infer an inter-frame gap of 666.67 µs, we decided to use the average of the measurements, that is, 1333 µs, as an input parameter to the simulation model and set its *productionInterval* parameter to this value instead. For the camera, the *packetLength* is set to 1442 Byte and *productionInterval* to 2738.5 µs.

The control traffic is mapped to PCP 3 or 4, depending on the sensor. The values were assigned based on the characteristics and quality of service requirements described in [26]. Two priority classes are used in each queuing module. The gate for the control traffic is opened for 48 µs and then closed for 52 µs. The *Traffic Generator* is disabled by configuring a time offset beyond the end of the simulation time. Regarding the time synchronization, *Switch2* serves as the master clock, while *Switch3* acts as a slave node.

### C. Result Analysis

We first analyze the traffic sent by the LiDAR through the TSN network. Figure 8 shows how the open source model allows us to model the behavior of the LiDAR sensor in simulation and to validate the TAS configuration. We plot a time series of the throughput and an eCDF each for the recorded e2e delay and jitter values. As can be seen, the throughput, e2e delay, and jitter measurements in the testbed all agree well with the simulation. No frame loss was recorded in either hardware or simulation.

We follow the same approach to validate the camera sensor in hardware and simulation. However, there is an important difference between the LiDAR and camera sensors. While the LiDAR produces a constant bitrate data stream and is therefore easier to replicate in simulation with a fixed inter-frame gap, the camera produces variable-bitrate data and using the same approach as before leads to inconsistent results in hardware and in simulation. This is illustrated in Figure 9, where we have run the simulation with a fixed inter-frame gap and frame size, as described in the simulation setup.

To reproduce the real behavior of the camera sensor in the simulation, we replaced the *Talker* application with a custom-built trace player application layer that generates frames according to the real data trace captured by the testbed. Each record in this data trace is a tuple ⟨*Timestamp*, *FrameSize*, *InterFrameGap*⟩ that is used to generate frames with variable sizes and inter-frame gaps in the simulation. In addition, we configured all the switches in the simulation to use limited queue sizes of 33 166 Byte each, i.e., to allow a maximum of 23 packets of 1442 Byte each.

The results obtained with this new configuration are shown in Figure 10. They show an almost perfect match in terms of throughput and jitter, and a very good match in terms of e2e delay. This demonstrates the importance of accurately modeling the input data to obtain well-calibrated simulation models.

We also noticed a packet loss of 5 % in hardware and 4 % in the simulation. This is due to the fact that we manually calibrated and used a fixed TAS configuration, which is not well suited for variable-bitrate traffic. Future work could include the design of adaptive TAS configurations, which could be optimized by a digital twin of the in-car network.

For reproducibility, we share the trace files of LiDAR and camera with this paper.[1]

## VII. CONCLUSION

In this paper, we addressed the problem of creating accurate digital replicas of modern automotive network architectures based on automotive Ethernet and Time-Sensitive Networking (TSN). To this end, we provided a detailed description of an open-source TSN simulation model specifically designed to match an accessible real-world TSN testbed that is composed of Commercial Off-the-Shelf (COTS) hardware. We identified several key issues that need to be considered when designing

such simulation models, and discussed potential problems that could lead to inaccurate TSN simulation results.

In particular, we demonstrated that hardware and software imperfections, such as those related to per-hop processing delay or non-real-time operating systems, are often hidden by the simulation abstractions and must be carefully considered to accurately reflect real-world conditions. We have shown that the data traffic generated by different real-world automotive sensors can vary and have a substantial impact on the accuracy of the simulation model. We published the real data traces of two sensors (LIDAR and camera) that we used to calibrate the TSN simulations.

As future work, we intend to take a further step towards closing the loop between the real network and its digital counterpart. To this end, we plan to implement and evaluate a translation module that will be responsible for replicating the TSN testbed configuration in the simulation module in real time, for example by interacting with the Centralized Network Configuration (CNC) module.

## ACKNOWLEDGMENTS

This research was funded in whole, or in part, by the Luxembourg National Research Fund (FNR), grant reference DEFENCE22/IS/17800623/LEONE. For the purpose of open access, and in fulfilment of the obligations arising from the grant agreement, the authors have applied a Creative Commons Attribution 4.0 International (CC BY 4.0) license to any Author Accepted Manuscript version arising from this submission.

## REFERENCES

[1] J. Peeck and R. Ernst, "Improving Worst-case TSN Communication Times of Large Sensor Data Samples by Exploiting Synchronization," *ACM Transactions on Embedded Computing Systems*, vol. 22, no. 5s, Sep. 2023.

[2] V. Bandur, G. Selim, V. Pantelic, and M. Lawford, "Making the Case for Centralized Automotive E/E Architectures," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 2, pp. 1230–1245, Feb. 2021.

[3] J. John, M. Noor-A-Rahim, A. Vijayan, H. V. Poor, and D. Pesch, "Industry 4.0 and Beyond: The Role of 5G, WiFi 7, and TSN in Enabling Smart Manufacturing," *arXiv preprint arXiv:2310.02379*, 2023.

[4] P.-J. Chaine, M. Boyer, C. Pagetti, and F. Wartel, "Comparative study of Ethernet technologies for next-generation satellite on-board networks," in *IEEE/AIAA 40th Digital Avionics Systems Conference (DASC)*, IEEE, Oct. 2021.

[5] J. Sanchez-Garrido et al., "Implementation of a Time-Sensitive Networking (TSN) Ethernet Bus for Microlaunchers," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 57, no. 5, pp. 2743–2758, Oct. 2021.

[6] R. Debnath, M. S. Akinci, D. Ajith, and S. Steinhorst, "5GTQ: QoS-Aware 5G-TSN Simulation Framework," in *2023 IEEE 98th Vehicular Technology Conference (VTC2023-Fall)*, IEEE, Oct. 2023.

[7] I. Turcanu and C. Sommer, "Poster: Potentials of Mixing TSN Wired Networks and Best-Effort Wireless Networks for V2X," in *13th IEEE Vehicular Networking Conference (VNC 2021), Poster Session*, Virtual Conference: IEEE, Nov. 2021, pp. 135–136.

[8] M. Bosk et al., "Methodology and Infrastructure for TSN-Based Reproducible Network Experiments," *IEEE Access*, vol. 10, pp. 109 203–109 239, 2022.

[9] P. Keller and N. Navet, "Approximating WCRT through the aggregation of short simulations with different initial conditions: application to TSN," in *30th International Conference on Real-Time Networks and Systems (RTNS '22)*, Paris, France: ACM, Jun. 2022, pp. 196–206.

[10] U. Bordoloi, S. Chakraborty, M. Jochim, P. Joshi, A. Raghuraman, and S. Ramesh, "Autonomy-driven Emerging Directions in Software-defined Vehicles," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, Apr. 2023.

[11] N. Finn, "Introduction to Time-Sensitive Networking," *IEEE Communications Standards Magazine*, vol. 6, no. 4, pp. 8–13, Dec. 2022.

[12] M. K. Atiq, R. Muzaffar, Ó. Seijo, I. Val, and H.-P. Bernhard, "When IEEE 802.11 and 5G Meet Time-Sensitive Networking," *IEEE Open Journal of the Industrial Electronics Society*, vol. 3, pp. 14–36, Dec. 2021.

[13] S. Senk, H. K. Nazari, H.-H. Liu, G. T. Nguyen, and F. H. P. Fitzek, "Open-Source Testbeds for Integrating Time-Sensitive Networking with 5G and beyond," in *2023 IEEE 20th Consumer Communications & Networking Conference (CCNC)*, IEEE, Jan. 2023.

[14] W. Quan, W. Fu, J. Yan, and Z. Sun, "OpenTSN: an open-source project for time-sensitive networking system development," *CCF Transactions on Networking*, vol. 3, no. 1, pp. 51–65, Aug. 2020.

[15] S. Senk, M. Ulbricht, J. Acevedo, G. T. Nguyen, P. Seeling, and F. H. P. Fitzek, "Flexible Measurement Testbed for Evaluating Time-Sensitive Networking in Industrial Automation Applications," in *IEEE 8th International Conference on Network Softwarization (NetSoft)*, Milan, Italy: IEEE, Jun. 2022, pp. 402–410.

[16] M. Ulbricht et al., "TSN-FlexTest: Flexible TSN Measurement Testbed," *IEEE Transactions on Network and Service Management*, Oct. 2023.

[17] G. Miranda et al., "Time-sensitive networking experimentation on open testbeds," in *IEEE Conference on Computer Communications Workshops (INFOCOM) 2022, 9th International Workshop on Computer and Networking Experimental Research Using Testbeds (CNERT 22)*, IEEE, May 2022.

[18] T. A. Xu et al., "Poster: Performance Evaluation of an Open-Source Audio-Video Bridging/Time-Sensitive Networking Testbed for Automotive Ethernet," in *IEEE Vehicular Networking Conference (VNC)*, IEEE, Dec. 2018.

[19] F. Rezabek et al., "EnGINE: Developing a Flexible Research Infrastructure for Reliable and Scalable Intra-Vehicular TSN Networks," in *17th International Conference on Network and Service Management (CNSM)*, IEEE, Oct. 2021, pp. 530–536.

[20] J. Falk et al., "NeSTiNg: Simulating IEEE Time-sensitive Networking (TSN) in OMNeT++," in *International Conference on Networked Systems (NetSys)*, IEEE, Mar. 2019.

[21] Z. Zhou, J. Lee, M. S. Berger, S. Park, and Y. Yan, "Simulating TSN traffic scheduling and shaping for future automotive Ethernet," *Journal of Communications and Networks*, vol. 23, no. 1, pp. 53–62, Feb. 2021.

[22] J. Jiang, Y. Li, S. H. Hong, M. Yu, A. Xu, and M. Wei, "A Simulation Model for Time-sensitive Networking (TSN) with Experimental Validation," in *24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, IEEE, Sep. 2019, pp. 153–160.

[23] M. Bosk et al., "Simulation and Practice: A Hybrid Experimentation Platform for TSN," in *IFIP Networking Conference (IFIP Networking)*, IEEE, Jun. 2023.

[24] P. Karle et al., "EDGAR: An Autonomous Driving Research Platform – From Feature Development to Real-World Application," *arXiv preprint arXiv:2309.15492*, 2023.

[25] J. Zou, X. Dai, and J. A. McDermid, "reTSN: Resilient and Efficient Time-Sensitive Network for Automotive In-Vehicle Communication," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 3, pp. 754–767, Mar. 2023.

[26] C. Correa, J. Simon, M. Gubow, and S. Bhaqwat, *Automotive Ethernet: The Definitive Guide*, 2nd ed. Intrepid Control Systems, 2022.

[27] "IEEE Standard for Local and Metropolitan Area Networks – Bridges and Bridged Networks," IEEE, Std 802.1Q-2022, 2022.

[28] Y. Wang, P. H. Chan, and V. Donzella, "A two-stage h. 264 based video compression method for automotive cameras," in *IEEE 5th international conference on industrial cyber-physical systems (ICPS)*, IEEE, Jul. 2022, pp. 01–06.