# Context Matters: Lessons Learned from Emulated and Simulated TSN Environments

Filip Rezabek*, Marcin Bosk†, Leander Seidlitz‡, Jörg Ott§ and Georg Carle¶
*Department of Computer Engineering, TUM School of CIT, Technical University of Munich, Germany*
Email: *rezabek@net.in.tum.de, †bosk@in.tum.de, ‡seidlitz@net.in.tum.de, §ott@in.tum.de
¶carle@net.in.tum.de

***Abstract*—Security and precise clock synchronization are crucial in modern Time Sensitive Networking (TSN) systems. However, deployment validation is challenging due to many different, incompatible, and inaccessible implementations of TSN and related standards and the complexity of involved system artifacts. This aspect makes ensuring proper functionality and performance difficult and requires thorough evaluation and validation to ensure a system's robustness. In this work, we analyze various platforms, using hardware and simulations as an evaluation tool.**

**We present the lessons learned during real-world deployments of the Precision Time Protocol (PTP) and MACsec. We identify the challenges of combining numerous sensors and propose how to possibly overcome the former using open-source solutions. To protect sensitive PTP traffic and other TSN protocols, we evaluate MACsec as a solution and determine its performance implications. Our findings clarify requirements for future deployments of TSN systems. Simulation as a supporting tool for TSN experimentation is discussed, and its limitations are explored. Lastly, we highlight how different evaluation approaches can provide an overall view of the system under test.**

***Index Terms*—COTS, Open-Source, TSN, MACsec, Simulation**

## I. Introduction

The growing complexity, scale, and throughput requirements in current real-time systems demand the exploration of new innovative architectures [1]. Nowadays, even specialized domain-specific networks are shifting toward Ethernet [2]. The technology alone might prove insufficient for specific applications, lacking deterministic bounds on latency, jitter, packet loss, and reliability. To overcome these limitations, the application of Time Sensitive Networking (TSN) standards[1] to Ethernet enables the missing real-time functionality. These solutions are widely adopted across diverse sectors, including Intra-Vehicular Networks (IVNs), aerospace, smart manufacturing, and professional audio and video solutions.

The TSN standards undergo extensive development and performance evaluation through simulation, emulation, and proprietary Hardware (HW) solutions. Each approach has its strengths and weaknesses. Simulation allows for easily configurable and reproducible experiments in a rapid development cycle without needing dedicated HW. However, it may deviate from real-world deployments by omitting certain artifacts such as operating system interactions or specific Network Interface Card (NIC) behaviors.

HW deployments, whether with dedicated or generic components, offer a more realistic representation of real-world scenarios. However, they must carefully consider both HW and Software (SW) aspects to meet desired real-time requirements. Although proprietary HW provides the most accurate results, researchers may encounter limitations due to vendor-locked solutions. Nevertheless, it is crucial to focus on the fundamental properties of the system to avoid only measuring a specific HW or SW implementation.

Building on previous work focusing on Commercial off-the-Shelf (COTS) HW and open-source SW [3], in this work, we explore the combination of specialized HW for the automotive industry, as introduced in [4]. We consider security extensions offered by the Media Access Control Security (MACsec) standard in combination with TSN standards. We also compare simulation and emulation of TSN, identifying their possible limitations and challenges. To facilitate replicable experiments with TSN systems, leveraging COTS HW and open-source SW, we use the Environment for Generic In-vehicular Networking Experiments (EnGINE) [5] and its latest extension, enabling usage of the Objective Modular Network Testbed in C++ (OMNeT++) discrete event simulator [6].

We start with Precision Time Protocol (PTP), which is a backbone of TSN, offering nanosecond-accurate [7] time synchronization. Such accuracy can be achieved if the network devices or sensors completely implement the IEEE 802.1AS [8] standard. To ensure the integrity and confidentiality of the PTP messages, we consider the MACsec standard [9]. MACsec protects Ethernet traffic exchanged among the peers. In this context, we evaluate the impact of MACsec on the TSN traffic. Additionally, we use simulation to enhance our understanding of TSN systems and evaluate their performance and capabilities. This approach lets us assess scenarios without the HW limitations and artifacts caused by SW or HW found in regular systems. Finally, based on [6], we rely on OMNeT++ and compare the results to HW deployments.

In this paper, we present the following lessons we learned from the aforementioned evaluation methods:

- **L1** Challenges regarding PTP on proprietary hardware
- **L2** The impact of MACsec on TSN when using COTS HW
- **L3** Assessment of OMNeT++ for the evaluation of TSN

Filip Rezabek and Marcin Bosk contributed equally to this paper.
All links are valid as of 30 January 2024.
[1]https://www.ieee802.org/1/pages/tsn.html

These lessons highlight possible challenges of using various HW in time-critical systems, their possible implementation limitations, and optimistic simulation results in some scenarios. These should help researchers and engineers build their own TSN platforms using heterogeneous HW deployments and open-source SW or simulations.

## II. Background

To support TSN experiments with COTS HW and open source solutions, we introduced the *EnGINE* framework [5]. We enhanced it by a methodology [10] for TSN experiments and enabled its integration with the OMNeT++ simulator [6]. Our experiments focus on achieving the latency, jitter, and packet loss requirements outlined by [11] and [12] for IVNs, also applicable to other TSN capable systems, as well as the security goals covered in [13], [14]. In our approaches, we utilize Linux queuing discipline (qdisc) implementations for Time Aware Priority Shaper (TAPRIO) and Earliest Time First (ETF), as well as Credit-Based Shaper (CBS). Accurate time synchronization in the network is achieved by PTP.

In this section, we give an overview of the leveraged technologies, focusing on their relevant configuration features.

### A. Precision Time Protocol

In a TSN system, precise time synchronization can be achieved through the use of the PTP, defined by the *IEEE 1588* [15] standard. System clocks are individually synchronized via PTP instances, organized in a master-slave hierarchy. The exchange of messages between the master and slave occurs over the network (either at the Link Layer or Transport Layer), resulting in the synchronization of the slave clock to the master clock. The Grandmaster Clock (GM) establishes the reference time for the entire system clock, situated at the top of the hierarchy. The process of PTP clock synchronization involves the exchange of timing information between nodes. This information is then used to calculate the clock offset and path delay between the nodes.

Within the PTP network, three types of clock devices are defined: Ordinary Clock (OC), Boundary Clock (BC), and Transparent Clock (TRCL). The OC is the simplest PTP device, possessing exactly one port in either a master or slave state. A BC has two or more ports and serves to link different segments of a PTP topology. All but one port is in the master state, while the remaining slave port synchronizes the internal clock of the BC. The state of the internal clock is then propagated via the master ports. The BC also functions as a fully-featured PTP node, making its synchronized internal clock available to applications. In contrast, the TRCL does not synchronize itself to the time reference but forwards PTP messages and adjusts them according to their residence time. The use of TRCLs enhances synchronization accuracy compared to a network solely based on BCs [7].

In the context of TSN, PTP is defined by the IEEE 802.1AS standard [8], referred to as generic Precision Time Protocol (gPTP). This protocol restricts message exchange to Layer 2 with IEEE 802.3 MAC and confines packet exchange to instances of gPTP, ensuring that clocks use the same frequency. For Linux-based systems, the `linuxptp` project [16] facilitates time synchronization with PTP. Network-wide clock synchronization is achieved using `ptp4l`, while synchronization between clocks of a single machine is handled by `phc2sys`. Configuration of `ptp4l` involves deploying a `gPTP.cfg` profile file, specifying gPTP profile parameters for each PTP interface on the machine.

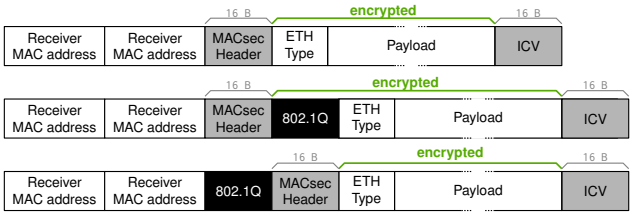



Fig. 1: MACsec header placement in the 802.3 frame. The 802.1Q header can be either encrypted (VLAN-encrypt) or placed outside (VLAN-plain) of the MACsec payload.

### B. Credit-Based Shaper

CBS, as defined in the IEEE 802.1Q-2018 [17] family of standards, allows for bandwidth allocation to Stream Reservation (SR) classes. Such functionality is achieved with a scheduler defining which frames are to be dequeued next using a credit system. The credit level of a given class increases when one or more packets are present in its queue. The class may transmit only when its accumulated credit exceeds zero and the interface is otherwise idle. With an addition of priority enforcement mechanisms, and proper configuration, CBS offers soft guarantees for delay, jitter, and packet loss.

The credit level over time for a given class is governed by four parameters: $hiCredit$, $loCredit$, $idleSlope$, and $sendSlope$. The first two parameters limit the maximum and minimum credit. The latter two parameters define the credit change rates applied when the class is transmitting or idling.

### C. MACsec

MACsec (defined by the 802.1AE standard [18]) enables connectionless data integrity, data confidentiality, data origin authentication as well as replay protection. A flow consists of two unidirectional streams of data, each of which MACsec secures separately by enclosing them as Secure Channel (SC). MACsec represents the security parameters of an SC with an Security Association (SA).

The encryption and decryption of the frame payload is handled either by the operating system or by the HW directly. Hardware support depends on driver support — and to our knowledge, HW support is very limited to non-existent. While, e. g., Intel network cards selectively have hardware support for MACsec, and driver support is absent on Linux. Therefore, significant overhead is introduced as the operating system has to handle all MACsec operations in software.

Figure 1 shows the MACsec header placement in the 802.3 Ethernet frame. An additional 802.1Q VLAN and QoS header can be placed inside (VLAN-encrypt mode) or outside (VLAN-plain mode) of the encrypted MACsec payload. MACsec builds its security guarantees around AES-GCM as a cipher. Key establishment is done statically or via the MACsec Key Agreement (MKA), which derives keys through the 802.1x Extensible Authentication Protocol (EAP), enabling node authentication. The secure channels are unidirectional, with two establishments securing traffic in both directions.

### D. OMNeT++

OMNeT++ [19] is a discrete-event simulator that can be used to simulate any network type. It is open-source and easily extensible. Its simulation libraries are written in C++. The simulator is comprised of modules, each implementing specific functionality of the networked system, interconnected using gates and channels. OMNeT++ simulations are defined using two types of files, the NED files describing the network, and INI configuration files for each scenario.

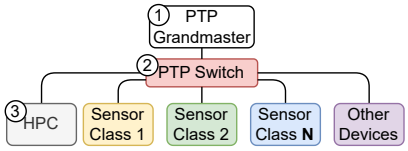
Various frameworks implement distinct features, which expand the generic OMNeT++ network model. Relevant in the scope of this work is the INET Framework [20] enabling the simulation of computer networks. INET implements protocols of all layers of the ISO/OSI stack. Starting with version 4.4.0, it also includes numerous TSN standards. Most notably, it provides an implementation of the PTP and CBS, which is used in Section V of this work.

## III. Precision Time Protocol Deployments

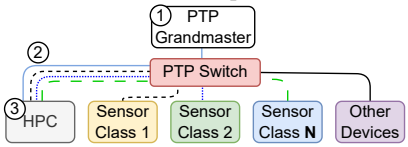
PTP plays a fundamental role in synchronous TSN standards like Time-Aware Shapers (TASs) [21], [22]. Beyond TSN, precisely timestamped data is essential for data fusion in applications like autonomous driving [4]. In such setups, sensors, e. g., cameras, LIDARs, or RADARs, need to support PTP to add precise timestamps to the message payload.

Figure 2 shows the PTP implementation architecture in the Excellent Driving GARching (EDGAR) project. Figure 2a illustrates the ideal scenario where ① serves as the PTP GM to which all devices synchronize. For local deployments, the GM does not need to sync to a wall clock but provides precise time to the vehicle's devices. However, for global communication, synchronization to a global source like GPS is relevant. PTP messages are distributed among devices supporting PTP via the PTP switch ② that is a TRCL. The High Performance Computer (HPC) ③ collects data from sensor families. Devices use their internal PTP clocks and should operate as OC.

While the ideal scenario has one hop between GM and devices, thereby minimizing clock deviation, challenges arose in the actual implementation due to configuration limitations, network-based inter-sensor interference, and only partial PTP protocol support in some cases. Our final deployment necessitates separation using dedicated physical or virtual links, as depicted in Figure 2b. We separate the sensor classes into different VLANs. The HPC ③ runs a dedicated `ptp4l`




(a) Envisioned Implementation




(b) Actual Implementation

Fig. 2: Envisioned implementation versus the actual implementation, showcasing the implementation Challenges.

instance for each VLAN. Each specific instance is tailored towards the compatibility of the respective sensor class, e. g. by distributing messages in PTP Layer 2 or Layer 4 modes. Splitting sensors into different networks was a necessary step to reduce the network noise each sensor class receives.

As a result, the HPC runs as a BC, ensuring synchronicity among other interfaces. The multiple PTP Hardware Clocks (PHCs) are synchronized using `phc2sys`. The separated network approach allows for easy system extension by additional device classes. Despite the additional hop, clock precision is still expected to be in the range of nanoseconds [7]. However, due to closed-source devices with limited documentation, external validation using, for example, an oscilloscope or other means would be necessary.

## IV. MACsec Integration with TSN

PTP requires message protection in order to provide reliable, unmodified information. MACsec helps to achieve such protection while encrypting the PTP traffic and any other traffic. This is not necessarily a problem, as additional applications, e.g., the automotive industry, require data security. Nevertheless, this must adhere to strict latency and jitter bounds [10] while ensuring data confidentiality and integrity requirements [13], [14]. MACsec provides both and thereby contributes to protection of the PTP and other traffic. This section discusses whether MACsec can provide security guarantees without violating the tight timing constraints.

In a series of experiments, we integrate MACsec into a TSN environment. We measure latency, jitter, and throughput for emulated high-priority, time-sensitive traffic. Our experiment setup consists of eight nodes in a line topology of seven hops, as shown in Figure 4. All links use 1 GbE Intel i210 NICs supporting TSN features. The HPC used as a source has an Intel Xeon D-1518 CPU with 128 GB of RAM. The remaining nodes (Low Performance Computers (LPCs)) feature a Xeon E3-1265L CPU and 16 GB RAM. Each intermediate forwarding node applies traffic shaping based on the 802.1Q header. In our evaluation, we focus on CBS.

We analyze the impact of MACsec using three scenarios:

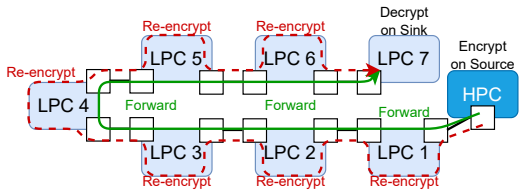



Fig. 3: EnGINE MACsec test architecture

1) Forward-only (FWD): Forwarding nodes do traffic shaping, but no MACsec is employed.
2) MACsec re-encrypt (MRE): The 802.1Q header is encrypted. Nodes shape traffic classes appropriately at ingress and egress interfaces. Each node decrypts, shapes, and then re-encrypts the traffic.
3) MACsec forward (MFW): The 802.1Q header is unencrypted. The traffic is encrypted by MACsec at the source node. The forwarding nodes only do traffic shaping, without handling any MACsec. Only the destination node decrypts. This mode reduces encryption and decryption operations to one each.

The outlined scenarios do not regard any test case in which the 801.1Q header remains encrypted on the nodes without its re-encryption. In this case, forwarding nodes cannot properly do traffic shaping since the priority classes stored in the VLAN header are not readable by them. An overview of the header structures is shown in Figure 1.

Figure 3 illustrates the corresponding scenarios, depicting in red the MACsec re-encrypt (MRE) and in green the MACsec forward (MFW) and Forward-only (FWD) scenarios. For FWD, the source does not perform encryption. The traffic is shaped using CBS on each interface. We configure it to account for the 32 B increase of the header size while maintaining the same throughput of the flows.

Figure 4 illustrates the impact of MACsec on throughput over seven hops. The overhead is already introduced on the node that generates the traffic. When MACsec is inactive, the network achieves its maximum 1 Gbit/s bandwidth capacity, aligning with expectations. In contrast, when employing MACsec we see a decrease in throughput to around 450 Mbit/s, with slightly superior performance in the MFW scenario. This reduction, exceeding 50 % for both modes, underscores the drastic impact of MACsec on the link performance. Figure 5 illustrates the end-to-end delay and jitter experienced in the three scenarios. We measure these metrics based on two flows, respectively assigned TSN SR Classes A and B and observe whether they can satisfy their respective requirements of 2 ms and 10 ms for delay and 125 µs and 1000 µs for jitter.

Due to the significant computational overhead introduced by MACsec we experience a saturation of available CPU capacity even at lower data rates. We therefore limit ourselves to the data rates of 7.5 Mbit/s and 15 Mbit/s for our experiments, in the expectation that the testbed can handle such low rates without violating timing constraints.

We observe that in the FWD scenario, we can satisfy these requirements. In the other two scenarios, due to the expected

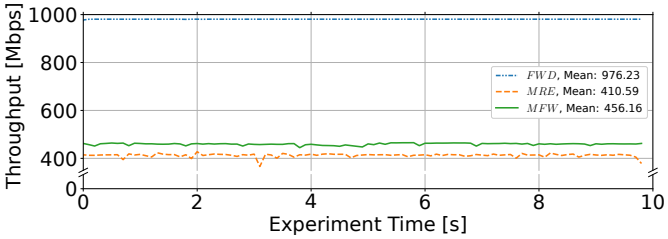
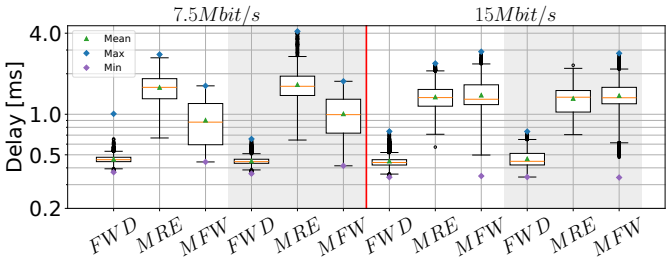
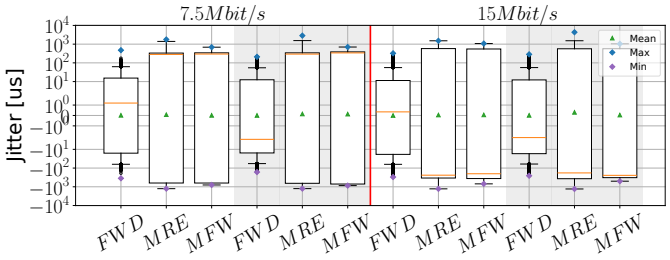



Fig. 4: Throughput achieved by Iperf3 over seven hops in the test cases FWD, MRE, and MFW




(a) CBS - Delay, 2 Flows SR Class A and B (grey background)




(b) CBS - Jitter, 2 Flows SR Class A and B (grey background)

Fig. 5: Comparison of CBS Delay and Jitter for two Flows with SR Class A and B with 7.5 and 15 Mbit/s throughput, for the scenarios FWD, MRE, and MFW.

MACsec overhead, we observe an increase in latency and jitter for both, SR class A and B. The means for both scenarios satisfy the latency requirements, but maximum values violate the requirements. For jitter, only the requirements of SR class B are satisfied. Overall, we observe a significant delay increase for both flows for the higher throughput of 15 Mbit/s. Our results point towards the overhead introduced by encrypting data in software as the source of the overhead. We conclude that there are significant challenges regarding the integration of MACsec together with TSN on hardware, as even low data rates already violate the timing constraints posed.

## V. Simulation to the Rescue?

Using HW-based setups for experimentation is quite complex and usually requires up-front investment and extensive setup. Researchers often rely on COTS HW and open-source SW to perform TSN experiments. In such setups, the compatibility of selected components and functionality of the associated drivers and SW needs to be thoroughly evaluated.

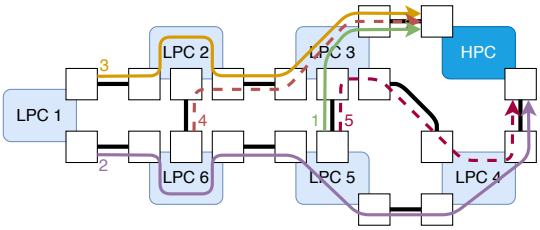



Fig. 6: Network with indicated packet flows used for the comparison of HW-based and simulated experiments

While the EnGINE simplifies experiment setup, configuration, and execution, it still needs access to HW. A possible solution with easier access to TSN experimentation, can be simulation. Thus, in recent work [6], we introduce a hybrid experimentation scheme, extending the HW-based experimentation capabilities via simulation using OMNeT++. While the simulator still requires quite complex configuration, EnGINE provides an abstraction that utilizes the same configuration scheme for simulated and HW-based experiments.

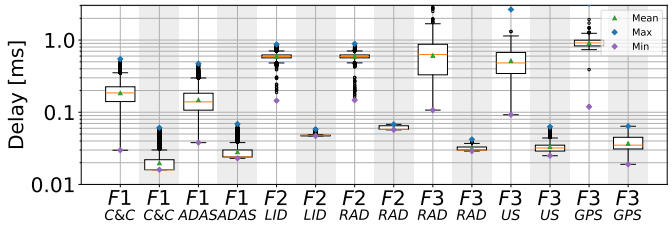
In our initial validation of the solution presented in [6], we investigated synthetic test cases consisting of a single traffic flow along a line network topology of seven hops. In those experiments, we observed quite significant discrepancies in the results of simulated and emulated runs. To gain additional insights into the performance of OMNeT++ TSN simulation compared to HW-based experimentation, in this work, we devise a more realistic scenario based on experiment $EX_{UCC}$ from [10]. The experiment consists of three classes of policed flows along paths 1, 2, and 3, as indicated in Figure 6. These flows follow different routes along the network. We further place interfering flows along paths 1, 2, and 3, with additional ones on paths 4 and 5, containing best-effort traffic.

Along path 1, we place flows corresponding to command & control, and Advanced Driver Assistance Systems (ADAS) traffic, belonging to SR class A with $F1_{C\&C}$ and $F1_{ADAS}$. Similarly, along path 2, we place SR class A traffic flows $F2_{LID}$ for LIDAR- and $F2_{RAD}$ for radar-generated traffic. Path 3 contains flows belonging to SR class B, encompassing radar - $F3_{RAD}$, ultrasound sensor - $F3_{US}$, and GPS-related - $F3_{GPS}$ traffic.

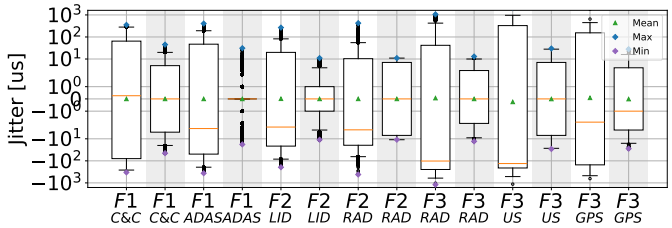
The aforementioned flows are policed on each interface along their paths using properly configured CBS qdisc. Flows belonging to SR class A have the highest priority, with those of class B being configured as the second highest. The best-effort traffic belongs to the lowest priority and is not policed.

We then perform two runs of the same scenario using a HW-based setup and simulation in OMNeT++. Figure 7 compares the results of the emulated and simulated experiments. We do not investigate the results of interfering, best-effort, flows.

Figure 7a shows the delay recorded in both experiment types. We observe a general correlation in both runs, with the delay being strongly dependent on the length of each path. However, the HW-based experiments show an order of magnitude higher delay when compared to simulated counterparts. The highest delay is observed along path 2 with an average




(a) Delay. Simulated results highlighted in grey.




(b) Jitter. Simulated results highlighted in grey.

Fig. 7: Comparison of HW-based and simulated experiments.

for both flows in HW-based experiment being $0.59\,\mathrm{ms}$, while simulation measures an average of only $0.05\,\mathrm{ms}$ and $0.06\,\mathrm{ms}$ for $F2_{LID}$ and $F2_{RAD}$, respectively. We observe a similar discrepancy in the jitter measurement shown in Figure 7b. The maximal absolute value for HW-based experiments is $2819.243\,\mu\mathrm{s}$ for flow $F3_{GPS}$. In contrast, we observe maximal jitter of only $45\,\mu\mathrm{s}$ for flow $F1_{C\&C}$.

The perfect nature of the simulation causes this significant discrepancy between the two types of results. It does not model many of the real-world aspects of the system, such as CPU/memory usage, memory access, communication access times via the PCI bus, or vendor-specific standard implementation. However, it includes a realistic clock model[2] with potential for, e.g., clock drift. Even with proper clocks and timing synchronization, as indicated in the results shown in Figures 7a and 7b, across the flows following the three paths, the discrepancy between results is not linear per each hop traversed by a packet. Therefore, the delay and jitter difference between the two experiment types cannot be modeled via, e.g., a processing offset added on each node.

Further, our HW-based experimentation approach presents a worst-case outcome, especially when using open-source SW and COTS HW. In contrast, the simulation shows a perfect best-case scenario that is hard to achieve in the real world. With dedicated TSN HW, the results should lay somewhere in between. Despite its shortcomings, the simulation can still give valuable insights into network performance and its configuration. However, its results should always be verified in hardware, before any solution is deployed, especially when system-dependent parameters are needed for configuration.

## VI. Discussion

COTS and proprietary HW combined with open-source SW come with particular challenges. In this work, we focused on the deployment of PTP and performance MACsec standards

[2]https://inet.omnetpp.org/docs/users-guide/ch-tsn.html

on Linux, with a further look at the use of OMNeT++ for support. The **L1** covered PTP deployments in an industrial setting with a variety of sensors that support the PTP standard. Considering the importance of PTP in the TSN, we evaluated the performance impact of MACsec as a solution for PTP and other data traffic security. The findings show a significant performance penalty to the system as a part of the **L2**. Lastly, considering the limitations of HW deployments, we considered using OMNeT++ simulator to provide a relevant evaluation platform for TSN standard behavior and bring scalable experiment capabilities. Certain findings cannot be easily ported to the physical devices. However, **L3** outlines the possible discrepancies between the performance of the simulator and the implementation in HW. Finding a suitable function to close the gap between the simulation and emulation infrastructure is challenging and requires further work.

Various industrial applications, e.g., data fusion or TSN standards, require precise configuration and rely on solutions such as PTP to bring such precision. However, due to the complexity and variety of HW on the market, the system configuration may be complex. Fortunately, the open-source solutions' versatility can also help mitigate certain challenges, as we can see in the case of PTP implementations. Using our findings, we want to motivate that TSN deployments with a heterogeneous set of HW components bring new challenges. When ensuring traffic security in transit, we must observe its impact on latency, jitter, and packet loss. MACsec, even though considered a standard for industrial applications, is currently supported only on a very limited number of HW devices along with possible TSN capabilities. Our findings show that an acceleration in HW is crucial to achieve a suitable performance. Otherwise, the system will struggle to keep up with higher loads. A few current NICs support MACsec offload capabilities using frameworks like the Data Plane Development Kit (DPDK), but the use of the latter would require a custom re-implementation of the Linux TSN capabilities.

A simulated digital twin of the system should enhance our understanding of hardware-software interactions and enable a better baseline for expected results in HW-based experimentation. However, as we identified in our work, this brings its own challenges and requires further investigation. For instance, extensions to the simulator could minimize the discrepancy observed between the two approaches and contribute models for various experimentation types. Such modeled results with a proper translation function to map to the real system's behavior could help better understand the complex interactions between components of a TSN system. Further enhancements of the simulation could enable an accurate evaluation of various TSN specifications at scale until more capable HW is available.

## Acknowledgment

This work is partially funded by the German Federal Ministry of Education and Research (BMBF) under the projects 6G-life (16KISK001K) and 6G-ANNA (16KISK107). We also received funding by the Bavarian Ministry of Economic Affairs, Regional Development, and Energy as part of the project 6G Future Lab Bavaria, and by the EDGAR project, DFG grant approval according to Art. 91b GG with DFG-number INST 95/1653-1 FUGG. We thank the reviewers and colleagues for their comments. Last, we thank Davide Alessi for his support with integrating MACsec into the EnGINE framework.

## References

[1] W. Zeng, M. A. S. Khalid, and S. Chowdhury, "In-Vehicle Networks Outlook: Achievements and Challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 1552–1571, 2016.

[2] S. Tuohy, M. Glavin, C. Hughes, E. Jones, M. Trivedi, and L. Kilmartin, "Intra-Vehicle Networks: A Review," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, 2015.

[3] F. Rezabek, M. Bosk, G. Carle, and J. Ott, "TSN Experiments Using COTS Hardware and Open-Source Solutions: Lessons Learned," in *2nd International Workshop on Negative Results in Pervasive Computing (PerFail 2023)*, Atlanta, USA, Mar. 2023.

[4] P. Karle *et al.*, *Edgar: An autonomous driving research platform – from feature development to real-world application*, 2023. arXiv: 2309.15492 [cs.RO].

[5] F. Rezabek *et al.*, "Engine: Flexible research infrastructure for reliable and scalable time sensitive networks," *Journal of Network and Systems Management*, vol. 30, no. 4, p. 74, 2022.

[6] M. Bosk *et al.*, "Simulation and Practice: A Hybrid Experimentation Platform for TSN," in *22nd International Federation for Information Processing (IFIP) Networking Conference*, Spain, Jun. 2023.

[7] F. Rezabek, M. Helm, T. Leonhardt, and G. Carle, "PTP Security Measures and their Impact on Synchronization Accuracy," in *18th International Conference on Network and Service Management (CNSM 2022)*, Thessaloniki, Greece, Nov. 2022.

[8] "IEEE Standard for Local and Metropolitan Area Networks–Timing and Synchronization for Time-Sensitive Applications," *IEEE Std 802.1AS-2020*, pp. 1–421, 2020.

[9] *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, IEEE, Jun. 2020.

[10] M. Bosk *et al.*, "Methodology and infrastructure for tsn-based reproducible network experiments," *IEEE Access*, 2022.

[11] "ISO/IEC/IEEE International Standard - Information technology – Telecommunications and Information Exchange between Systems – Local and Metropolitan Area Networks – Specific Requirements – Part 1BA: Audio video bridging (AVB) Systems," *ISO/IEC/IEEE 8802-1BA First edition 2016-10-15*, pp. 1–52, 2016.

[12] "IEEE Standard for a Transport Protocol for Time-Sensitive Applications in Bridged Local Area Networks," *IEEE Std 1722-2016 (Revision of IEEE Std 1722-2011)*, pp. 1–233, 2016.

[13] "ISO/SAE 21434:2021: Road Vehicles - Cybersecurity Engineering," *Vehicle Cybersecurity Systems Engineering Committee*, 2021.

[14] "UN Regulation No. 155 - Cyber security and cyber security management system," *UN Regulation No. 155*, Mar. 2021.

[15] "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems," *IEEE Std 1588-2019*,

[16] R. Cochran, *linuxptp*, Last accessed on 2022-11-26. [Online]. Available: https://sourceforge.net/projects/linuxptp/.

[17] "IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks," *IEEE Std 802.1Q-2018*, pp. 1–1993, 2018.

[18] "IEEE Standard for Local and metropolitan area networks-Media Access Control (MAC) Security," *IEEE Std 802.1AE-2018 (Revision of IEEE Std 802.1AE-2006)*, pp. 1–239, 2018.

[19] A. Varga, "OMNeT++," in *Modeling and Tools for Network Simulation*, K. Wehrle, M. Güneş, and J. Gross, Eds., Heidelberg: Springer, 2010, pp. 35–59.

[20] L. Mészáros, A. Varga, and M. Kirsche, "Inet framework," A. Virdis and M. Kirsche, Eds., pp. 55–106, 2019.

[21] "IEEE Standard for Local and Metropolitan Area Networks – Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic," pp. 1–57, 2016.

[22] I. S. Association *et al.*, "IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks—Amendment 25: Enhancements for Scheduled Traffic," *Amendment to IEEE Std*, vol. 802, pp. 1–57, 2016.