TSN Experiments Using COTS Hardware and Open-Source Solutions: Lessons Learned

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

***Abstract*—Time-Sensitive Networking (TSN) brings deterministic behavior to Ethernet-based systems, resulting in hardware and software supporting various TSN standards. Using TSN-capable Commercial off-the-Shelf (COTS) hardware and open-source software brings several challenges. These are especially visible while performing performance evaluation of various TSN standards. In this work, we present the most significant challenges we faced using such deployments. Starting with the Precision Time Protocol, we observe its implementation being incompatible with that of the Time-Aware Priority Shaper. We present several solutions on how to overcome the identified behavior and compare them proposing best fitting solution for any setup.**

Next, we focus on the Network Interface Cards (NICs) and their behavior in presence of various TSN standards. We observe that the hardware offload features aiming to improve performance sometimes introduce performance artifacts worthwhile of investigation. Further, even though the Credit-Based Shaper configuration parameters can theoretically be computed for various NICs, due to the internal optimization of some, the calculated parameters may not hold.

Our findings are intended to help the community improve observed results and solve challenges in using the COTS hardware and open-source software. We believe additional documentation detailing the implementation aspects of TSN standards in hardware would be beneficial in explanation of observed behavior.

***Index Terms*—TSN, COTS, Open-Source, PTP, Experiments**

I. Introduction

The increasing scale, complexity, and data throughput of today's networks [1] prompt their operators to look for new solutions. Even for specialized domain-specific networks, some of them shift towards Ethernet [2]. For certain applications, Ethernet by itself may not be sufficient as, by default, it does not offer deterministic bounds on latency, jitter, packet loss, and reliability. To overcome these shortcomings, the Time Sensitive Networking (TSN) standards[1] can be applied to Ethernet, enabling the missing real-time functionality. Nowadays, those solutions are more widely adopted in areas such as Intra-Vehicular Networks (IVNs), aerospace, smart manufacturing, or professional audio and video solutions.

The TSN standards are widely worked on, with their performance being evaluated in simulation, emulation, and using proprietary hardware solutions. Each of these approaches has its advantages and drawbacks. With simulation, experiments can be easily configured and reproduced in a rapid development cycle which does not require any dedicated hardware. Yet, simulations often differ from real-world deployments as they omit certain artifacts such as operating system interactions or Network Interface Card (NIC) behavior specifics. These drawbacks can be overcome using hardware deployments with dedicated or generic components realizing the required functionality. Undoubtedly, in this approach, the hardware and software components must be carefully considered to achieve desired real-time requirements. The system's performance needs to be thoroughly evaluated and its deficiencies must be well understood to achieve a well-performing experimental environment. Finally, proprietary hardware yields the most accurate results in the real-world. However, such experimentation may not be available to researchers when vendor-locked solutions are used.

To enable replicable experiments with TSN systems, Commercial off-the-Shelf (COTS) Hardware (HW) and open-source Software (SW) can be used. Such an approach, e.g., introduced in [3] and [4], naturally faces multiple challenges concerning hardware-software interactions. In this work, we want to share our lessons learned during the use and development of experimental COTS and open-source based TSN systems. We introduce the challenge of coexistence between the IEEE 802.1Qbv Time Aware Priority Shaper (TAPRIO) queuing discipline (qdisc) on Linux and `linuxptp`. Precision Time Protocol (PTP) is a crucial building block for synchronous TSN standards that require precise clock synchronization. Similarly, for the COTS HW we focus on the features offered by NICs such as the HW offloading of the Earliest Time First (ETF) qdisc. Lastly, we assess the usage of Credit-Based Shaper (CBS) on 1GbE and 10GbE NICs. Based on the evaluation we identified that the CBS parameters cannot be easily ported to higher throughput NICs. The lessons we present in the paper are summarized as follows:

- **L1** Usage of TAPRIO and PTP in Linux
- **L2** Evaluation of NIC hardware offloading capabilities
- **L3** Assessment of CBS standard on various NICs

The identified lessons should help researchers and engineers aiming to build their TSN systems platform using COTS HW and open-source SW.

II. Background

To support TSN experiments with COTS HW and open source solutions, in previous work, we introduced the *EnGINE* framework [3], enhanced by a methodology [4] for such experiments. We focus on achieving the requirements outlined

Filip Rezabek and Marcin Bosk contributed equally to this paper.
[1]https://www.ieee802.org/1/pages/tsn.html, Access 25.01.23

by [5] and [6] for IVNs, also applicable to other TSN capable systems. We utilize Linux with qdisc implementations for TAPRIO and ETF, as well as CBS. On top of that we consider PTP to synchronize the time within the network. In the following sections, we give a brief overview of these technologies, focusing on their relevant configuration features.

A. Precision Time Protocol

The time in a TSN system can be synchronized using PTP described in *IEEE 1588* [7] standard. The individual clocks are synchronized via PTP instances running on each participating device structured in a master-slave hierarchy. Master and slave exchange messages over the network and the slave clock is synchronized to the master clock. The reference time for the whole system is determined by the Grandmaster Clock (GM) clock, which is placed on the top of the hierarchy.

The PTP clock synchronization requires timing information exchange between nodes. The timing information is used to compute the clock offset and path delay between them.

The PTP defines three clock device types - Ordinary Clock (OC), Boundary Clock (BC), and Transparent Clock (TRCL). The simplest PTP device is the OC. It has one port in either a master or slave state. A BC has two or more ports and is used to link parts of a PTP topology. All but one of the ports is in the master state. The remaining port is in the slave state and is used to synchronize the internal clock of the BC, which is in turn propagated via the master ports. The BC also acts as a full-featured PTP node and its synchronized internal clock can be used by applications that need it. On the other hand, TRCL, does not synchronize itself to the time reference. Instead, it forwards the PTP messages and adjusts the PTP message according to the residence time in the TRCL. Using TRCLs improves the synchronization accuracy over a purely BC-based network [8].

PTP is further applied to TSN with the IEEE 802.1AS [9] standard as a generic Precision Time Protocol (gPTP). The protocol only allows message exchange at Layer 2 with IEEE 802.1 MAC. It also constrains the packet exchange to gPTP instances, as the clocks need to use the same frequency.

In Linux-based systems, the `linuxptp` project [10] enables time synchronization with PTP. The clock synchronization across the network is achieved with `ptp4l`, and between clocks of one machine with `phc2sys`. `ptp4l` is configured using a `gPTP.cfg` profile file, specifying gPTP profile parameters for each PTP interface on the machine.

B. Time-Aware Priority Shaper

To ensure deterministic packet delivery as well as low latencies and jitter, *IEEE 802.1Qbv* [11], [12] amendment known as Time-Aware Shaper (TAS), or TAPRIO qdisc [13] in Linux, can be used. The qdisc supports synchronized packet scheduling between multiple traffic classes (TCLs) and queues of a single NIC, similarly as in a Time Division Multiple Access (TDMA) system. Synchronization is achieved using gates operating according to a user-specified cycle, with frames only being sent when a gate is open. Each TCL can get a dedicated window within the cycle when the transmission is allowed. Within a window, the packets are passed to a child qdisc that may additionally police and shape traffic. These child qdiscs could, e.g., include ETF of CBS, and can further influence the system behavior. Furthermore, in Linux, the TAPRIO qdisc uses packet priorities to map the TCLs to queues and cycle windows. The functionality of the qdisc can be influenced by the following configuration parameters:

- `base-time` enables alignment of schedules across the network
- `sched-entry S $MASK $DURATION` enables configuration of the gate window opening schedule
- `flags` with support for option `0x1` indicating TxTime mode, which can be used to specify sending time of a packet for an application not supporting such functionality natively. Further, it supports `0x2` option enabling offload of TAPRIO functionality to the NIC
- `txtime-delay` to account for the system delay

Generally, the TAPRIO qdisc utilizes features of the ETF qdisc configured alongside it. Therefore, the TAPRIO configuration needs to consider the interplay with ETF, which is especially relevant for the `txtime-delay` and associated ETF `delay` parameters.

C. Earliest-TxTime First

The ETF qdisc offers control over the transmission time (TxTime) of individual frames. It is generally configured on each HW queue of the relevant NIC. The TxTime can be specified via the socket SO_TXTIME option.

ETF supports two transmission modes, strict and deadline. With strict mode, the packets are transmitted exactly at TxTime, while in deadline mode, the frame can be dequeued anytime before TxTime. For some NICs, e.g., Intel® I210, the ETF functionality can be hardware offloaded.

D. Credit-Based Shaper

The algorithm for CBS was first defined with the IEEE 802.1Qav amendment, now part of the IEEE 802.1Q-2018 [14] family of standards. CBS enables bandwidth allocation and guarantees to pre-defined Stream Reservation (SR) classes. To achieve that, the algorithm defines the next frame to be transmitted on an interface from a set of SR classes and their associated queues. The bandwidth allocation for every class is ensured by a scheduling system based on credits. With that, CBS also offers some bounds on delay, jitter, and packet loss.

Using CBS, a frame can only be transmitted when the class's collected credit is ≥ 0 and the NIC does not transmit any other frames simultaneously. Credit is only accumulated when at least one packet is present in the class's queue and is spent during frame transmission. With no packets in the queue of a given class, the credit level does not change. Four parameters govern the credit level over time for a class X:

- $hiCredit_X$ - maximum credit level
- $loCredit_X$ - minimum, negative, credit level
- $idleSlope_X$ - credit replenish rate when idle
- $sendSlope_X$ - credit spend rate during transmission

The proper parameter setting for class X can be calculated using Equations (1) to (5) and the following information:

- B_X - class X bandwidth fraction
- MFS_X - class X maximum frame size, including Physical Layer (PHY) overhead
- PTR - NIC transmission rate
- MFS_0 - maximum frame size supported by the NIC, including PHY overhead
- Class Y denotes any other SR class using the NIC

For Linux implementation of CBS, these parameters are defined using B and kbit/s, but the IEEE 802.1Qav [14] standard uses bit or bit/s respectively.

$$idleSlope_X = B_X \cdot PTR \tag{1}$$

$$sendSlope_X = idleSlope_X - PTR \tag{2}$$

$$loCredit_X = MFS_X \cdot \frac{sendSlope_X}{PTR} \tag{3}$$

$$hiCredit_X = idleSlope_X \cdot (\frac{MFS_0}{PTR} + IF_X) \tag{4}$$

$$IF_X = \sum_{Y}^{Y<X} (\frac{hiCredit_Y}{-sendSlope_Y} + \frac{MFS_Y}{PTR}) \tag{5}$$

The CBS qdisc in Linux is usually used in conjunction with the Multiqueue Priority Qdisc (MQPRIO) qdisc. MQPRIO enables packets of various SR classes to be mapped to predetermined priorities and queues of the NIC. CBS is configured as a child qdisc of MQPRIO, enabling packets to be handled by the proper CBS qdisc. Of note, ETF can also be configured with MQPRIO.

III. Precision Time Protocol Synchronization and Co-existence with TAPRIO

PTP plays an important role for synchronous TSN standards, such as the IEEE 802.1Qbv, TAPRIO Linux qdisc. The synchronization of the clocks in the system plays a key role in ensuring the window alignment on each hop. Unfortunately, we identified a co-existence challenge of the `ptp4l` daemon and TAPRIO while configured on the same NIC. These results are confirmed using the Linux kernel versions 5.4 and 5.15 with Ubuntu 20 Focal and 22 Jammy Jellyfish, respectively. The `linuxptp` version is 3.1.1. We verified this behavior using several NICs, namely the Intel® I210, I350, I225, and X552. Importantly, all of the NICs we used for our experiments support the IEEE 802.1AS standard, which limits the variety of selections in the market. In case IEEE 802.1AS is not supported the required precision in nanoseconds cannot be achieved. We use BCs for our experiments as the nodes internally also require the clock information. In these experiments we do not police the PTP traffic, but in a presence of more network flows, policing of PTP messages would benefit the precision of the synchronization.

Figure 1 indicates the expected behavior of TAPRIO and `linuxptp`, where each interface runs the `ptp4l` daemon and sends the synchronization messages on the link. Depending on the number of hops we want to evaluate, each hop must have synchronized clocks. Based on our findings, this approach works as expected for other qdiscs (e.g., CBS and ETF with MQPRIO). We observe time synchronization failures only when TAPRIO is configured alongside PTP. To note, similar issues were observed by others as can be seen from the discussions in the open issues[2,3].

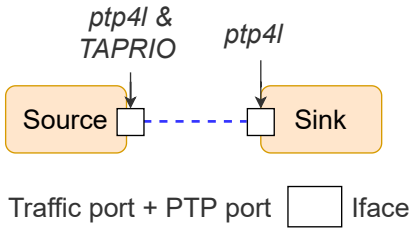


Fig. 1. Setup of ptp4l daemon with TAPRIO on a source and sink

We were able to identify a single configuration allowing for PTP to work alongside TAPRIO where `txtime-assist` mode is unused. However, better solutions are needed as the aforementioned mode is essential to specify the TxTime of a packet without the usage of custom traffic-generating applications. Further, based on our testing, this approach only works in one-hop configurations. With more hops, the PTP needs to function properly on the further hops and synchronize all clocks in the networks to the GM.

To overcome this challenge, we identified three possible solutions shown in Figure 2. Figure 2a shows **Solution I** and relies on two nodes (servers) that have a large number of ports that allow for multi-hop experiments. The nodes are connected with a dedicated link running the `ptp4l` daemon that ensures the time synchronization between them. Locally each node runs `phc2sys` to synchronize the individual PTP Hardware Clocks (PHCs). With this approach, as expected, we can use the TAPRIO on the other available ports of the machines and ensure the window alignment across the different hops.

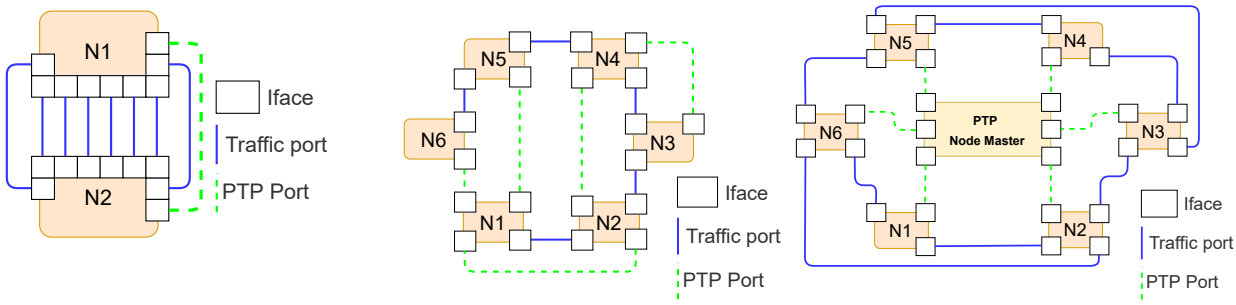
Figure 2b introduces **Solution II**, enabling use of multiple nodes to build a TSN system. However, this approach requires using additional, redundant paths in the system. The solution uses at least the same number of nodes we evaluate in a given experiment. The exact amount depends on the node wiring. We call this approach a PTP 'overlay' network where additional dedicated ports are used to synchronize the time across all nodes and rely on the `phc2sys` to synchronize all PHCs. The additionally used ports must be at least in active state if PTP on the link layer is used.

Lastly, **Solution III** shown in Figure 2c uses a dedicated PTP master node connecting to each node via a dedicated link. Such solution is fitting if a server with a sufficient amount of ports is available, bringing additional advantages concerning clock precision. As shown in [8], generally, a higher number of hops results in a larger clock deviation.

Table I compares the advantages and disadvantages of the **Solutions I, II**, and **III**. Depending on the scenario type

[2]https://github.com/Avnu/tsn-doc/issues/24, Access 28.11.22

[3]https://github.com/nxp-archive/openil_linuxptp/issues/19, Access 28.11.22



(a) Solution I (b) Solution II (c) Solution III

Fig. 2. Solutions I, II, and III showcasing how to overcome the coexistence challenge of PTP and TAPRIO

TABLE I
SOLUTIONS I, II, AND III COMPARISON, ✓ - PREFERABLE SOLUTION, ○ - FEASIBLE APPROACH WITH SIDE EFFECTS

Solution	PTP Precision	Number of Connections	Processing Overhead
I	✓	✓	○
II	○	○	✓
III	✓	○	✓

and available hardware, different scenarios are suitable. If the generated traffic can overload the system, **Solution I** is not feasible as both machines are limited by their CPU and RAM resources. The processing overhead in such a system could result in artifacts affecting the performance, e.g., increased delay, jitter, and packet loss. In contrast, the first approach requires the least wiring and achieves good clock precision thanks to the single PTP hop. Also, **Solution I** is less realistic as various types of topologies are used in real deployments. Instead, the realism is supported by **Solutions II** and **III**. These allow for load distribution among many nodes for the price of additional connections, enabling clock synchronization among multiple nodes. **Solution II** might be impacted by clock precision if too many hops are used [8]. Finally, **Solution III** improves the synchronization precision by introducing a central node into the TSN system. Nevertheless, not every deployment may desire or even can support a dedicated PTP clock master, more links, and NICs.

For our experiments [3], [4], we used **Solution II** as it allows for flexible wiring among the nodes. Further, with the number of hops we use, this approach is not impacted by the clock deviation. An alternative solution, absent in detail within this work, could include dedicated NICs supporting GPS e.g., Cisco Nexus GM. Nevertheless, this approach requires additional HW, which is only feasible in some cases.

Based on our analysis, the observed coexistence challenge between PTP and TAPRIO is most likely caused by the software component. We make such an assumption since we investigated the behavior on various aforementioned NICs with the same outcome. Further, additional investigation using various kernel versions (5.4 and 5.15 in normal and low-latency variants) did not yield better results.

IV. EARLIEST-TXTIME FIRST CONFIGURATION

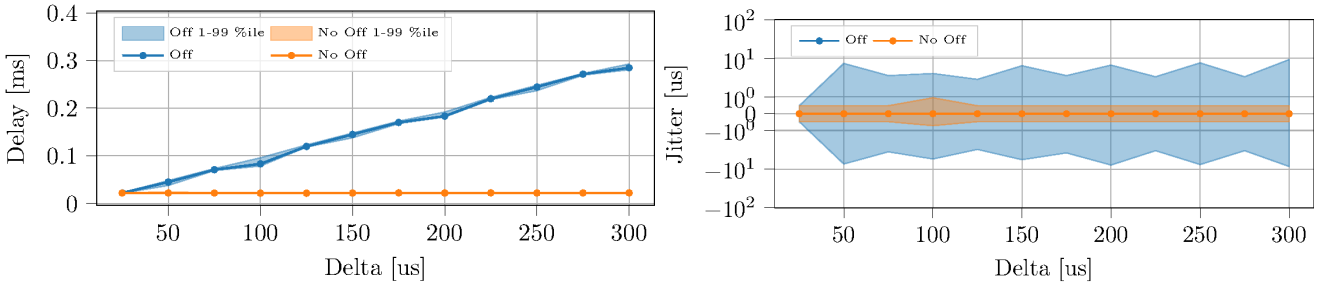
The Intel® I210 NIC used in the EnGINE [4] setup supports the so-called Launch Time Feature (also called ETF), which enables packet dequeuing at a TxTime. As a part of the experiment setup evaluation, we investigated the impact of the ETF behavior in SW and using the HW offload. For that, we performed a parameter study of the `delta` values between 25-300 µs, Figure 3. For the evaluation, a direct connection between two nodes was used, similar to the Figure 1, with the configuration of ETF as a child and MQPRIO as a parent qdisc. The application generating the packets is based on the udp_tai.c[4] which can specify the packet's TxTime. The experiment ran with Ubuntu 20.04 LTS and kernel version 5.4. The figures show mean values along with 1st and 99th percentiles for delay and jitter. With the software version of the qdisc, the delay and jitter behave similarly with minimal fluctuations caused by the changing `delta` parameter, as shown in Figure 3a and Figure 3b respectively. However, using the offload feature of the NIC shows a constant increase of the delay with rising `delta` parameter value. The same holds for the jitter values, which fluctuate between -10 to $10\,\mu s$, being higher than for the case with no offload.

In contrast, the expected outcome of the HW offload would be a performance improvement, as the logic and decisions happen directly on the NIC. However, our results show the opposite. Therefore, even if the NIC supports such a feature, it is required to consider its evaluation to see its impact on the system's performance. Unfortunately, it is challenging to find proper documentation that could better explain why we are observing such behavior. Currently, we assume that the observed delay increase is caused by how the measurements are performed. With the software version of the qdisc, the timestamp is taken after traffic shaping is complete, while with HW offload, the shaping takes place at the NIC, after the recording of the timestamp.

V. CREDIT-BASED SHAPER CONFIGURATION

We perform experiments with CBS using a 3-hop line topology consisting of 4 nodes interconnected using 1 Gbit/s

[4]https://gist.github.com/jeez/bd3afeff081ba64a695008dd8215866f#file-udp_tai-c, Access 25.01.23



(a) Delay

(b) Jitter

Fig. 3. ETF `delta` values with and without offload, delay and jitter mean with 1th and 99th-percentiles for various `delta` values, and packet received and droprates. Colored area represents the range between 1st and 99th percentiles values. Cf. [4]

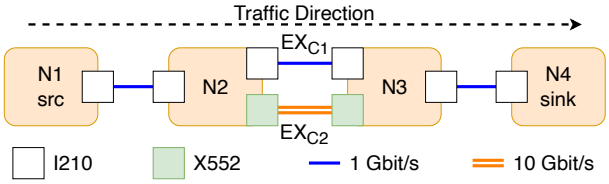


Fig. 4. Network topology used in EX_{C1} and EX_{C2}

(via Intel® I210 NIC) and 10 Gbit/s (via Intel® X552 NIC) links, as shown in Figure 4. Using this topology, we propose two types of experiments: EX_{C1} where all three used links are 1 Gbit/s, and EX_{C2} where links connected to source and sink are 1 Gbit/s and the remaining link between N2 and N3 is 10 Gbit/s. On each of the interfaces used in those experiments, we apply a CBS qdisc appropriately configured for the expected traffic pattern and link bitrate. Each UDP flow is assigned a different priority, with traffic going from the source node N1 to the sink node N4. We consider two of those flows, F1 and F2, to be time-sensitive and place those on the highest priorities 3 and 2. Of note, a higher number indicates higher priority. Both F1 and F2 transmit 1250 B packets every 100 µs, resulting in a bitrate of 100 Mbit/s. The remaining flows F3 and F4 are considered best-effort, use priorities 1 and 0, and transmit UDP packets saturating the 1 Gbit/s link.

For both types of experiments, we apply the CBS qdisc for F1 and F2 with the configuration being derived using Equations (1) to (5). For EX_{C1}, this results in the following CBS configuration parameters on the 1 Gbit/s links with a guaranteed bitrate of 100 Mbit/s for flows F1 and F2.

F1 $idleSlope_{F1} = 100000$, $sendSlope_{F1} = -900000$, $hiCredit_{F1} = 155$, $loCredit_{F1} = -1125$

F2 $idleSlope_{F2} = 100000$, $sendSlope_{F2} = -900000$, $hiCredit_{F2} = 297$, $loCredit_{F2} = -1125$

Consequently, we derive the configuration for CBS applied on the 10 Gbit/s link used in EX_{C2}. Here, we consider the actual bitrate supported by the Intel® X552, based on the pacing settings of the NIC [15]. The CBS configuration defined for EX_{C1} is applied on the 1 Gbit/s links in EX_{C2}.

F1 $idleSlope_{F1} = 100000$, $sendSlope_{F1} = -9194196$, $hiCredit_{F1} = 17$, $loCredit_{F1} = -1237$

F2 $idleSlope_{F2} = 100000$, $sendSlope_{F2} = -9194196$, $hiCredit_{F2} = 31$, $loCredit_{F2} = -1237$

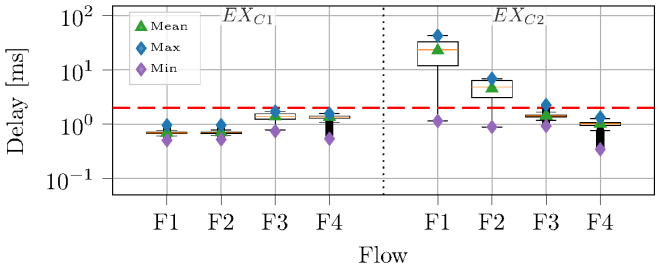


Fig. 5. Boxplot of the end-to-end delay observed in EX_{C1} and EX_{C2}

We then measure the end-to-end delay for each flow in EX_{C1} and EX_{C2}. We run the experiments for 2 s with all four flows being simultaneously active. Figure 5 shows the boxplot of the end-to-end delay for each flow in both experiments. Of note, the Y-axis shows end-to-end delay in logarithmic scale, and the red dashed line indicates the end-to-end target delay of 2 ms for high-priority flow F1. In EX_{C1} we see that the delay of flows F1 and F2 is consistent throughout the experiment and does not exceed the target 2 ms. Best-effort flows F3 and F4 also observe a similarly low delay, with increased jitter. In contrast, in experiment EX_{C2} which uses the 10 Gbit/s link with appropriate CBS configuration, we observe an increased end-to-end delay for the policed flows, exceeding the target 2 ms for flow F1. The delay of best-effort flows is similar to that observed in EX_{C1}. Such increase in latency for flows F1 and F2 indicates incorrect cooperation of the CBS qdisc with the 10 Gbit/s NIC, when appropriate CBS configuration derived based on Equations (1) to (5) is used.

To further assess the CBS configuration on the 10 Gbit/s NICs, we perform a small parameter study, increasing the guaranteed bitrate for flows F1 and F2 to 101 Mbit/s, 102 Mbit/s, and 103 Mbit/s. The 1 Mbit/s increase aims to show the general trend of the discrepancy between theory and actual behavior. This change resulted in an increased *idleSlope* and consequently decreased *sendSlope* for the flows. Similarly to Figure 5, Figure 6 shows a boxplot of the delay of flows F1 and F2 in EX_{C2} with the modified parameters. We observe that with increasing guarantee, the observed end-to-end delay is decreasing. Furthermore, in the experiment with the idleSlope set for 103 Mbit/s, the latency for both flows is below the target 2 ms.

While we are still determining the exact reason for the

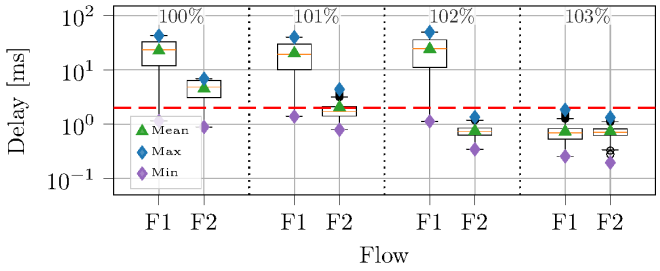


Fig. 6. Boxplot of the end-to-end delay observed in EX_{C2} parameter study

increased delay observed in EX_{C2}, we consider it to be a result of a misconfiguration due to missing information on the X552 NIC. With the increasing guarantees, we observe that the delay decreases, indicating a disparity between the expected bitrate and the actual PHY layer bitrate observed at the NIC. This could be a result of missing information on the interaction between the PHY layer and Linux CBS qdisc. One of the possible reasons for this mismatch could be the system configuration of the NIC considered in this work. The X552 NIC utilized in this work uses packet pacing to operate at a slightly lower speed than the $10\,\mathrm{Gbit/s}$. Pacing is achieved by padding packets with additional bytes. It is unclear how much changing of the packet size influences the operation of the CBS qdisc and its interaction with the NIC.

VI. Discussion

Using COTS HW and open-source SW comes with certain challenges. As identified in this work, we focused on the results of TSN standards on the Linux and NIC that support TSN. The **L1** covered the aspect of PTP and TAPRIO co-existence challenge, for which we presented possible solutions and their pros and cons. Next, we assessed the HW offloading capabilities that, to our surprise, impacted the performance as a part of the **L2**. Finally, even though the CBS standard defines how to compute its configuration parameters, certain NICs come with their own features for which the computation does not hold. As a part of the **L3**, we outlined the possible impact on the performance and how to assess such behavior. Nevertheless, finding a suitable countermeasure is challenging, even though it is easy to spot.

TSN standards require precise configuration and rely on other solutions to offer certain behavior, e.g., PTP. Unfortunately, as shown in this work, the system and hardware do not always bring the expected outcomes. Using our findings, we want to motivate the proper testing and investigation of various TSN deployments with respect to latency, jitter, and packet loss. The ETF and TAPRIO have strictly system-dependent parameters that must be tested, e.g., `txtime-delay` and `delta` values. Similarly, the CBS requires taking all NIC features into consideration when configuring the parameters. Additional testing and a larger community could help with evaluating various NICs and creating performance profiles based on the observed behavior.

Another approach to enhance the understanding of hardware-software interactions could involve a digital twin of the system to obtain a better baseline of expected results. This baseline could enable the identification of artifacts induced by the system itself or the TSN specification. Such a tool could be implemented using a simulation environment. Results modeled in this way, with all drawbacks of simulations, could help better understand the complex interactions between components of a TSN system. The digital twin could also further support the thorough benchmarking required with the increasingly complex specifications observed in these networks.

Acknowledgment

This work has been supported by the Algorand Centres of Excellence (ACE) Programme (https://www.algorand.foundation/ace) and Federal Ministry of Education and Research of Germany (BMBF) project 6G-Life (16KISK002). We thank the reviewers and colleagues for their comments.

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, pp. 534–545, 2015.

[3] F. Rezabek, M. Bosk, T. Paul, K. Holzinger, S. Gallenmüller, A. Gonzalez, A. Kane, F. Fons, Z. Haigang, G. Carle, and J. Ott, "Engine: Flexible research infrastructure for reliable and scalable time sensitive networks," *Journal of Network and Systems Management*, vol. 30, no. 4, p. 74, 2022.

[4] M. Bosk, F. Rezabek, K. Holzinger, A. G. Marino, A. A. Kane, F. Fons, J. Ott, and G. Carle, "Methodology and infrastructure for tsn-based reproducible network experiments," *IEEE Access*, vol. 10, pp. 109 203–109 239, 2022.

[5] "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.

[6] "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.

[7] "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems," *IEEE Std 1588-2019*, pp. 1–499, 2020.

[8] 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.

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

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

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

[12] 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.

[13] "TAPRIO(8)," Last accessed on 2022-11-26. [Online]. Available: https://man7.org/linux/man-pages/man8/tc-taprio.8.html

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

[15] "Datasheet - Volume 4 of 4: Intel Xeon Processor D-1500 Product Family LAN Controller," *Intel Xeon Processor D-1500 Product Family*, Nov. 2018, Last accessed on 2022-11-26. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/datasheets/xeon-d-1500-datasheet-vol-4.pdf