TSN-FlexTest: Flexible TSN Measurement Testbed

Marian Ulbricht®, Stefan Senk®, Hosein K. Nazari®, How-Hang Liu®, Martin Reisslein®, Fellow, IEEE, Giang T. Nguyen®, Member, IEEE, and Frank H. P. Fitzek®, Senior Member, IEEE

Abstract—In order to provide consistent low-latency communication network services, Time-Sensitive Networking (TSN) unites a set of standards for time-synchronization, flow control, enhanced reliability, and management. We design the TSN-FlexTest testbed with generic commodity hardware and open-source software components to enable flexible TSN measurements. We have conducted extensive measurements to validate the TSN-FlexTest testbed and to examine TSN characteristics. The measurements provide insights into the effects of TSN configurations, such as increasing the number of synchronization messages for the Precision Time Protocol, indicating that a measurement precision of 30 ns can be achieved. The TSN measurements included extensive evaluations of the Time-Aware Shaper (TAS) for sets of Tactile Internet (TI) packet traffic streams. The measurements elucidate the effects of different scheduling and shaping approaches, while revealing the need for pervasive network control that synchronizes the sending nodes with the network switches. We present the first measurements of distributed TAS with synchronized senders on a commodity hardware testbed, demonstrating the same Quality-of-Service as with dedicated wires for high-priority TI streams despite a 200% over-saturation cross traffic load. The testbed is provided as an open-source project to facilitate future TSN research.

Index Terms—Ethernet, industrial communication, Quality-of-Service, testbed, Time-Sensitive Networking.

I. INTRODUCTION

FOR A wide range of applications and domains it is necessary to avoid unexpected communication behaviors, such as inconsistent (highly variable) packet latencies. Among others, the industrial, automotive, medical, and avionic domains have strict requirements on the underlying communication infrastructure [2], [3], [4]. To mitigate problems in the communication infrastructure, deterministic data transmission is preferred in these domains. Determinism in networks requires maintaining full knowledge and control of packet-based transmissions so as to achieve perfectly constant packet latencies. However, achieving determinism in data communication is very challenging, and commonly the more realistic goal is to strive to achieve consistent (nearly constant) and very short packet latencies. Time-Sensitive Networking (TSN) unites different mechanisms to achieve consistent low-latency packet transport over Ethernet networks. TSN is managed by the IEEE TSN Task Group (TG) which defines the underlying standards.

TSN mainly encompasses four categories: time-synchronization, latency and packet delay variation reduction through flow control, ultra-reliability, and resource management. Each category improves specific aspects of data communication. The standards can be flexibly combined to adapt to particular use cases. Although the standards can be used individually, the greatest benefit is obtained by a judicious combination of selected features. Recent studies have mainly focused on integrating TSN with wireless technologies, as well as frameworks for managing and optimizing TSN core functionalities, such as the Time-Aware Shaper (TAS) [5]. Also, enhancing low-latency end-to-end transmission [6], and the integration of TSN with the software-defined networking (SDN) paradigm has been pursued [7].

A. Limitations of Existing TSN Evaluation Frameworks

TSN focuses not only on the reliability of packet transmissions, but also on the timing aspects of the packet transmissions. It is crucial to precisely measure timing-related metrics, such as the one-way delay and the Packet Delay Variation (PDV). Three primary methods are used for evaluating TSN protocols and systems: theoretical mathematical analysis, simulation and emulation, and hardware testbeds. Mathematical analysis frameworks, such as [8], [9], have been developed to evaluate TSN. However, the mathematical analysis frameworks abstract the behaviors of TSN systems compared to real-world systems.
Simulation frameworks, such as OMNeT++ [10] and NS-3 [11], have been widely used in networking research, including TSN research. The advantages of simulations include flexibility, reduced cost, and scalability. A main drawback of simulations in TSN research is that they do not involve real experimental hardware networking components, making it impossible to showcase the applicability and demonstrate the developed TSN systems and protocols with real operating networks. Emulation frameworks and tools were developed to address the applicability issue of simulation software. However, the scale of the emulated system is limited by the computing resources of the PC running the emulation (whereby, computing resources are typically limited by cost constraints). Therefore, timing-related emulation measurements of large and complex systems are unreliable.

Using dedicated hardware is, in most cases, more precise while having the same complexity as real network deployments [12], [13], [14]. Even though scalability is a limiting factor for designing hardware-based TSN testbeds caused by limited resources, dedicated TSN devices offer the full range of scenarios to be considered [15], [16], [17]. However, testbeds have complexities, e.g., the measurement techniques and setup can skew the results [18], [19]. Therefore, building a testbed with high precision and flexibility is challenging, yet desirable and facilitates reproducible experiments.

As TSN is a family of standards, TSN-related testbeds have been built to study various isolated TSN aspects, namely i) packet processing, ii) evaluating the Precision Time Protocol, iii) communication over-the-air; iv) TSN performance; and v) TSN management. One of the primary goals of packet processing frameworks, such as MoonGen [20] and P4STA [21], is to precisely generate packets and to precisely timestamp both transmitted and received packets. Even though MoonGen requires Data Plane Development Kit (DPDK)-supported hardware, it offers a high degree of flexibility and high performance.

Nevertheless, a testbed for studying TSN needs more than just packet generation and processing. Several existing testbeds focus on time synchronization, primarily with the Precision Time Protocol (PTP), such as [22], [23]. However, they solely evaluate the synchronization precision without assessing TSN mechanisms. More comprehensive testbeds have been built to study the performance of TSN in combination with wireless communication, such as WiFi and 5G [24], [25], [26]. A few testbeds target specific application contexts, such as for benchmarking the TSN performance of the controller on multi-domain networks [27] and for intra-vehicle networking [28]. However, they either neglect the per-packet delay or have limited precision due to inconsistent hardware and software timestamping. Overall, the existing testbeds focus on specific narrow sets of Key Performance Indicators (KPIs). Also, the existing studies commonly consider limited precision levels and often neglect detailed investigations of the effects experienced by a TAS scheduled data stream.

B. Contributions: Open-Source TSN-FlexTest Testbed

We address a root impediment in TSN research: the need for high-precision reproducible measurement studies in comprehensive TSN testbeds. We design and develop TSN-FlexTest, an open-source, flexible, high-precision, yet affordable measurement testbed framework to comprehensively study TSN. Our key innovation is a well-engineered combination of affordable Commercial-off-the-Shelf (COTS) hardware with an automated measurement workflow from the packet generation to the data collection and the visualization of the measurement results. Our open-source measurement framework facilitates the sharing of experiment scenarios and setups. Furthermore, the testbed workflow is organized into a centralized management and control entity, making the testbed workflow very easy to change. Last, introducing key abstraction entities, such as the Device-under-Test (DuT) and the traffic generator, make the entire testbed flexible to change. Striving towards high precision and affordability at the same time, we advocate the use of commodity hardware, which also facilitates reproducibility.

This paper makes the following two main contributions: Section III first reviews the general design principles for a TSN testbed, including the available options for hardware and software components and their trade-offs. Subsequently, we detail the TSN-FlexTest testbed architecture, design, and implementation, including the workflow of our measurement framework. We also share the lessons we learned while developing our measurement framework. In Section IV, we validate our TSN-FlexTest testbed, which produced results with nanosecond precision. A preliminary version of this contribution was presented in [1], focusing on the testbed design. We make the TSN-FlexTest framework [29] and the evaluation traffic streams [30] openly available.

The second main contribution of this paper is the investigation of critical aspects of TSN in Section V. This investigation leverages the developed and validated TSN-FlexTest testbed and is a significant novel contribution in this extended article compared to the preliminary conference version [1]. There are four main findings: 1) We evaluate a typical configuration in a net-neutral Internet, without specialized packet treatment and with limited resources in the backbone networks. We demonstrate that it is nearly impossible to ensure prescribed service quality in such a net-neutral Internet. 2) We investigate the impacts of soft Quality-of-Service (QoS) with Strict Priority Queuing (SPQ) and conclude that SPQ cannot enforce strict guarantees. 3) We measure the one-way packet latency for TAS. The results reveal inconsistent advantages of traffic shaping and hard scheduling. The TAS sometimes achieves latency reductions, especially when considering the average latency. 4) We demonstrate that synchronizing a generic application running on COTS hardware, i.e., the transmitter, with the TAS on the network switches achieves essentially the same packet latencies as a dedicated link.

II. RELATED WORK

This section reviews the related work on TSN network performance evaluation.

A. Formal Mathematical Analysis

Some of the TSN core functionalities, such as traffic shaping, are tractable in formal mathematical analysis.
Mathematical analysis can explore a wide range of configurations for an equation-based model of the TSN system. As a result, performance can be analyzed more thoroughly than with simulations or empirical measurements for specific configurations. Several studies have used formal mathematical analysis, including network calculus [31], to evaluate TSN, such as [8], [9], [32], [33]. Generally, formal mathematical analysis avoids the issue of configuring specific simulation or measurement scenarios, but the formal modeling may tend to reflect pessimistic timing behaviors [34].

B. Simulation and Emulation

Simulation tools have played a pivotal role in TSN evaluations since new protocols can be designed and validated quickly and with relatively low costs in simulation models. The OMNeT++ [10] and INET [35] frameworks have been used in several TSN studies, e.g., [36], [37], [38]. In particular, Falk et al. [36] implemented a standard-compatible simulation model based on OMNeT++ and INET, which includes VLAN tagging, as well as forwarding and queuing enhancements for time-sensitive streams [Credit-based Shaper (CBS)], Frame Preemption, and scheduled traffic (TAS). Moreover, Huang et al. [38] introduced an alternative to CBS called time-aware cyclic-queueing (TACQ), and employed OMNeT++ and INET to demonstrate the benefits of TACQ for scheduling mix-flows and reducing the PDV for isochronous traffic compared to CBS. Apart from OMNeT++, NS-3 [11] is a well-known network simulator into which TSN has recently been integrated [39], [40]. The powerful modeling features for wireless channels make NS-3 a promising candidate for combining 5G and TSN in future simulation models [39], [40]. However, [39], [40] mainly focus on TAS and neglect other TSN features, such as time synchronization.

None of the existing TSN simulators includes all TSN features. Although the latest INET library supports most TSN features, the correctness of combining them has not yet been validated. Moreover, the existing TSN simulators lack a comprehensive framework that accommodates large-scale heterogeneous network architectures.

Emulators provide a cost-efficient alternative to dedicated hardware testbeds by mimicking the underlying hardware and software environments to test code in a variety of settings. Thus, emulations are commonly an important intermediate evaluation and developmental phase towards real-world experiments with actual hardware. One example of software for emulation is Mininet [41]. Mininet uses Linux namespaces to emulate a set of nodes on a host system. The connections between the nodes are made via virtual interfaces, so that the behavior is similar to a Virtual Machine (VM)-based virtual environment but with less overhead. Mininet provides an ssh connection into each node, thus each node can be handled as a standalone Linux system. Ulbricht et al. [42] implemented TSN in Mininet. The scalability of network emulation is limited by the resources of the host system; therefore, the precision of emulated TSN networks typically deteriorates as the number of used nodes increases.

C. Measurements in Hardware Testbeds

Simulation, mathematical analysis, and emulation are important stepping stones on the way towards designing real-world testbeds. However, using dedicated hardware is in most cases more precise while having the same complexity as the targeted real network deployment. Whereas simulation models reduce the system complexity, dedicated TSN devices offer the full range of settings and parameters to be considered. The typically limited resources constrain the scale of hardware-based TSN testbeds. Employing hardware testbeds comes with its own set of complexities: the measuring techniques, setup, and other elements can strongly influence the outcome. For instance, simulation models usually have built-in tools for collecting evaluation data. However, aside from some exceptions, such as development kits or explicit measurement tools, commercial hardware is typically not designed with built-in evaluation tools.

Measurement evaluations of proposed TSN methods have been conducted on dedicated hardware testbeds for several purposes, as reviewed next.

1) Packet Processing: A set of software and hardware should be employed for conducting the intended evaluation in testbeds. For example, when utilizing time-aware gates, the precision of producing scheduled traffic can be crucial. We briefly review options for generating and time-stamping packets. MoonGen [20] is a high-speed packet generator that takes advantage of hardware capabilities to reliably regulate the pace of arbitrary traffic patterns on commodity hardware and to timestamp data packets with sub-microsecond precision. For efficient operation, MoonGen requires the Data Plane Development Kit (DPDK), which comprises libraries to speed up workloads operating on a broad range of CPU architectures.

P4STA [21] operates on programmable hardware, such as smartNICs and Field-programmable Gate Array (FPGA) [43], [44], [45], as well as P4-based technology [46] to achieve measurement precisions of a few nanoseconds. Programmable network hardware is important for flexible and precise measurements [47]. Runge et al. [48] and Beifuß et al. [49] focused on QoS aspects of packet processing in COTS hardware and investigated the effects of the Linux network stacks and related queuing issues.

2) Evaluating the Precision Time Protocol (PTP): Some hardware testbed studies concentrated on improving the hardware and software aspects of time-synchronization, mostly with PTP. Clock synchronization precision in smart-grid systems was investigated in [22]. For these systems to operate reliably, timestamping is necessary, and it is crucial to maintain performance standards even for limited communication bandwidth. To achieve an optimal PTP system for a Wide Area Network (WAN), Kassouf et al. [22] investigated the effects of various network factors and clock settings on synchronization precision. They report a 6.38 ns back-to-back master-slave deviation, compared to our 14 ns maximum deviation, see Figure 4. Ferencz and Kovácszégy [50] built a testbed using inexpensive COTS components, such as generally available CPUs with x86 architecture and typical Network...
Interface Cards (NICs). Using IEEE 1588v2 and hardware-assisted COTS devices, they aim to attain sub-microsecond precision. In a PTP-limited performance evaluation, they observed a synchronization precision of 482.7 ns. Similarly, Kyriakakis et al. [23] employed hardware-assisted clocks for synchronization. To increase the timestamp precision, they integrate a PTP hardware-assist unit into a multicore FPGA platform. Through studies on a testbed consisting of two FPGAs that implement the suggested design and are interconnected through a COTS switch, they examined the worst-case execution time, achieving a worst-case offset of 138 ns.

A completely centralized IEEE 802.1Qcc [51] architecture for configuring the generalized Precision Time Protocol (gPTP) profile is used by Thi et al. [52] in a Software-defined Networking (SDN)-based automotive and industrial application setting. Thi et al. examined specialized hardware from the manufacturer NXP and limited their evaluations to temporal synchronization, ignoring packet scheduling.

The existing hardware testbed studies on evaluating PTP typically evaluate only the synchronization precision. In contrast, this TSN-FlexTest study evaluates the PTP synchronization precision, and broadly evaluates TSN mechanisms.

3) Communication Over-the-Air: The performance and design of TSN in complex structures as well as their integration with other communication technologies, such as WiFi and 5G, have attracted the interest of both academia and industry. Sudhakaran et al. [24] designed an industrial testbed that combines traditional TSN in the wired domain with IEEE 802.11. Sudhakaran et al. [24] are primarily concerned with the feasibility and testing of an industrial collaborative robotics use case. Kehl et al. [25] built an Over-the-Air testbed on an industrial shop floor with a prototype of 5G-TSN integration to investigate a typical industrial use case with cloud-controlled mobile robots. Agarwal et al. [26] investigated the benefit of employing TSN in microgrid monitoring. They used a commercial Cisco switch to conduct measurements on a laboratory-scale microgrid with four nodes. Agarwal et al. [26] found that TSN can enhance QoS through increased data rates and traffic shaping while retaining connection reliability.

4) TSN Measurement: Böhm et al. [27] investigated the controller performance of TSN networks on a three-domain TSN testbed. They found that a negotiation mechanism for provisioning real-time end-to-end connections across various TSN domains can be beneficial. In contrast to the TSN-FlexTest measurement technique, Böhm et al. [27] did not measure the latencies of individual packets and did not consider oversaturated network conditions.

Bosk et al. [28] devised a methodology to assess TSN networks, specifically in the context of Intra-Vehicle Networking (IVN). Through systematic analysis of IVN traffic patterns for TSN and Best-Effort (BE) cross traffic, Bosk et al. [28] evaluate different TSN standards in a configuration using COTS hardware and open-source solutions. They try to optimize the system to meet the IVN application requirements. To achieve this, Bosk et al. employ the EnGine [53] framework, which provides a reproducible and scalable TSN experimentation environment. However, the measurement environment in [28], [53] mixes hardware- and software-based time stamping in combination with coarse-precision *iperf* packet generation, which results in relatively long cycle times for the TAS configurations and underscores the need of a well-evaluated TSN measurement methodology.

5) Time-sensitive Networking Management: Several testbed measurement studies investigated the impact of dynamic reconfiguration of the TSN network and optimizing TSN main functionalities, e.g., shapers, frame preemption, and stream reliability, as reviewed in the following. Jiang et al. [37] provided not only a TSN simulation but also built a hardware testbed. The simulation results verify that the real-world testbed matches the TSN specification, however, only for µs resolution. A self-configuring testbed for QoS management has been developed by Garbugli et al. [54] and employed for testing scenarios with varying packet sizes. However, the details of the testbed, such as the specifications of the testbed switches, are not disclosed. Groß et al. [55] proposed a specialized hardware extension for standard Ethernet controllers. The flexible and scalable architecture proposed in [55] can run solely in hardware, solely in software, or in both hardware and software.

Coleman et al. [56] addressed the performance impact of the degree of precision of network and CPU clock synchronization on a real-time TSN network. Coleman et al. presented an enhancement utilizing the PCIe bus interface to improve the network-to-CPU synchronization and assessed the improvement using COTS hardware. Vlk et al. [57] developed a method for increasing the schedulability and throughput of time-triggered traffic in IEEE 802.1Qbv. Miranda et al. [58] demonstrate the setup and operation of a cloud-based Linux testbed for TSN experimentation. Using a Central Network Controller (CNC) prototype, TSN bridges and nodes can be initialized and managed in the Linux testbed. The studies on designing and evaluating TSN switches based on FPGA, e.g., [59], [61], are interesting because FPGA can achieve high performance, programmability, and customization with a low entry cost. However, FPGA is not cost-efficient for mass production. Also, the FPGA design procedure is time-consuming and requires a relatively extensive effort for achieving high precision. In contrast, the proposed TSN-FlexTest testbed is based on COTS hardware, which can
be purchased and set up with relatively low development effort, while achieving a high degree of precision.

Overall, the existing testbed studies focus on specific narrow sets of KPIs. Also, the existing studies commonly consider only coarse precision levels and often lack detailed investigations of the effects that are experienced by a TAS scheduled data stream. With our TSN-FLEXTest testbed, we evaluate the measurement methods against the PTP synchronization precision, and we extend the measurement method with exact time stamping to a resolution that is in the same range as the PTP sojourn to enable comprehensive detailed measurements of the TAS mechanisms, such as TAS.

III. TSN-FLEXTest Testbed Design

This section provides insights into the hardware and software selected for the testbed. Also, the procedure flow and the used test data are described.

A. Testbed Overview

The purpose of the described TSN testbed is the examination of the behavior of different network stream patterns in a given DuT. The TSN-FLEXTest testbed accommodates, in principle, arbitrary network configurations with complex network topologies [62], [63] as DuT. For illustration, we consider a single TSN switch as DuT in this study, see Figure 1.

Five dedicated nodes are connected to the TSN switch with 1 Gbit s\(^{-1}\) links of negligible length. A node is an end-station in a TSN domain, i.e., talker or listener, and is comprised of COTS hardware, including an x86 CPU and multiple NICs. Nodes 1, 2, and 3 replay and transmit prioritized (TSN) packet streams through the DuT to one sink node (node0) which captures all received traffic. Node4 generates two streams of interfering BE cross traffic. All five generated packet streams from the transmitting nodes share the same bottleneck by traversing the output port of the DuT to the sink node (node0). On this bottleneck, different scheduling and shaping strategies can be applied.

The topology is designed for the unidirectional measurement of the packet latency from the time instant when a transmitting node (node1, node2, or node3) transmits the first bit of a packet onto the physical link to the time instant when the first bit of the packet is received by the receive-NIC on the receiving node (node0). The testbed thus enables the measurement of the sojourn (residence) time of a packet inside the switch DuT, which always operates in cut-through mode in our evaluations, with a precision of a few tens of nanoseconds (see Section IV-A). The packet sojourn time in the switch DuT accounts for the switch packet processing delay and packet queuing delay, and the packet transmission delay for the cut-through portion of the packet (see Section IV-C); in our setup, the link propagation delays are negligible due to the short physical cables. For the measurement, all devices are synchronized using PTP, whereby one-step timestamping is used for one-way delay measurements. The synchronization traffic is transferred in-band to be closely aligned with real TSN scenarios where the main benefit of TSN is the convergence of link technologies.

B. Analysis of Available and Utilized Hardware

In preparation for the testbed design, we conducted an extensive market analysis of the available TSN hardware components, see [64] for full details, which we cannot include due to space constraints. We considered TSN Intellectual Property Cores (IP-Cores), System on Chips (SoCs), Switched Endpoints (SEPs), switches, and NICs. Some vendors provide combined setups as TSN kits. The available Intel NICs do not have a dedicated hardware TAS, but support the hardware-accelerated transmitter (TX)-transmission time, which enhances all software schedulers with additional hardware support and improves their precision.

Unlike industry-grade products, basic COTS (generic commodity) hardware components in combination with Linux facilitate the flexible extraction of measurement data. On the other hand, industry-grade hardware products provide typically state-of-the-art reliable performance. For example, the TrustNode switch from InnoRoute GmbH, has an industry-grade version and a research version. The research version, which we purchased, has highly precise reliable packet time stamping, obviating additional time synchronization for source and sink [42]. Unfortunately, with enabled timestamping of every packet, the device became unstable for two 1 Gbit s\(^{-1}\) MoonGen streams (i.e., sporadic reboots occurred, likely due to unsafe memory behaviors), which makes the TrustNode switch unsuitable for our test setup.

The integration and debugging of IP-Cores is very complex and resource-consuming and requires hardware description language skills. Therefore, the IP-Core solution should be considered as a last resort if simpler methods are not applicable. Intel has proposed a special TSN Central Processing Unit (CPU) with Time Coordinated Computing (TCC) [65] which will probably reduce the effects of CPU load on timing and measurements. Unfortunately, the device was not yet available for order. Thus, the testbed hardware selection is a trade-off of feature completeness, measurement capability, stability, and availability. As summarized in Table I, we selected Intel NICs, e.g., the I210 NIC, as flexible COTS devices, and the stable and available Falcon-RX/G switch from FibroLan.
Emmerich et al. [20] investigated the synchronization capabilities of the Intel 82599 NIC. The internal clock of the Intel 82599 is 64 bit wide, whereby the Least Significant Bit (LSB) represents 1 ns. A timer event periodically increments the clock value. Unfortunately, this period depends on the link speed and for 1 Gbit s⁻¹ is fixed to 64 ns [68]. Emmerich et al. [20] measured that the reported clock values from the NIC are consistently multiples of 64 ns.

Newer NICs feature improved clock resolution. The I210 NIC, which evolved from the Intel 82599, has an internal 96 bit wide timer which is incremented every 8 ns, independent of the link speed [69]. Due to the larger clock register, the I210 has 32 bit in the sub nanosecond range, allowing the clock to be adjusted in fine-granular steps. Every 8 ns, the NIC adds an increment value to the clock register. This increment value can be slightly greater or smaller than 8 ns. The value itself is provided during the clock synchronization process from the Linux PTP `ptp4l` [70] via the `ptp_adifreq()` kernel interface callback function. By manipulating the increment value, the clock can be forced to run slightly faster or slower. The clock control is thus controlled by the `ptp4l` clock servo [70]. Due to this implementation, the investigated clock values reported from the I210 NIC are not multiples of the clock granularity, but can differ in multiples of 1 ns.

Several effects influence the PTP precision: `ptp4l` periodically measures the path delay and adjusts the local hardware clock based on the received master clock values corrected by the path delay. Thus, the jitter of the path delay and the corresponding servo-loop settings are important factors for the PTP precision.

2) Packet Generation: There are basically two types of traffic generators: a) (raw) socket based, and b) DPDK based. Both approaches have pros and cons: using sockets and generating packets in user space is highly flexible, allowing packet generation and other network usage (e.g., PTP) at once. The disadvantage of sockets are the limited performance for generating enough traffic to fully load an Ethernet link. Emmerich et al. [71] and Wong et al. [72] found that not all packet generators can saturate a link, especially if small packets are used. If precision is not important, the usage of Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) sockets within `iperf` is a cost-efficient solution for generating traffic [28]; although, `iperf` is not an optimal packet generator [72].

A more powerful approach is provided by DPDK, by making the configuration memory space of the NIC accessible from user space. This enables the user to write Ethernet data into memory and to pass the memory pointer directly to the NIC’s TX queue without utilizing the Linux network stack. The problem with DPDK is that it replaces the NIC’s driver with a user space pass-trough driver and disconnects the NIC from the Linux network stack. Thus, common software, such as `ptp4l`, cannot be used on this interface anymore.

Next to the achievable data rate, the generator’s precision is an important KPI. A fixed cyclic stream should transmit the packets periodically exactly according to the predefined cycle time. However, due to CPU load or other software effects, the packet transmission times can have random or constant offsets. DPDK based traffic generators achieve higher periodicity precision than socket-based generators [71]. MoonGen [20] (which was compared with `tcpreplay` in [1]) provides a lua framework for generating high data rate packet load with the help of DPDK. We use MoonGen for generating high-load cross traffic on a dedicated node (node4) that does not require PTP synchronization. For the replay of our test traffic we employ `tcpreplay` [1], which uses raw sockets to replay arbitrary traffic from prerecorded files.

3) Scheduler – GCL Configuration: Since version 5.4, the Linux kernel includes an implementation of the time-aware shaper and an (additional) transmission timestamp. In contrast to the TX timestamp, this additional transmission timestamp is the time when the frame should be transmitted, i.e., a value in the future. In theory, a packet with an additional transmission timestamp is handed over from the network driver to the NIC hardware, which stores the packet in the transmission queue and will release the packet when the transmission time is reached.

Due to limited resources in the hardware layer of the NIC, this procedure needs to be supported by the higher layers of the network protocol stack. The NIC hardware
cannot store a large amount of data for transmission since the internal NIC memory is limited to a few packets. The NIC queue is a First-In-First-Out (FIFO) buffer; therefore, the packets cannot be reordered. To respect the limited hardware resources, the Linux network stack has to be aware of these characteristics. The $qp_{disc}$ environment provides an interface in the Linux network protocol stack to insert and manage queuing and packet scheduling. One loadable module is the Earliest Time First (ETF) scheduler which reorders incoming packets according to their transmission timestamp. To respect the limited hardware queue size, the packets need to be temporarily stored in a software queue and handed to the hardware just shortly before the transmission time. This time offset needs to be determined experimentally for each NIC and system configuration. To set the transmission time of each packet, one possible module is $ta$-$prio$ which implements a software TAS for scheduling packets.

### D. Testbed General Procedure Flow

The TSN-FlexTest testbed supports the automatic test of the packet transmission capabilities of a dedicated DuT. Several nodes are controlled to replay measurement traffic and send it through the DuT. The NIC of each transmission node inserts a precise TX timestamp into the payload section of TX packets. Additionally, nodes can be configured to replay a cross traffic to simulate a busy link. One node (node0 in Fig. 1) captures the measurement packets after they have passed the DuT. By comparing the TX and RX timestamps, the transit time of each packet through the DuT can be evaluated. A detailed description of the testbed setup is provided in [1], [64].

1) Packet Header/Our Streams: To determine the packet handling behaviors of the DuT, the selection of characteristic test (packet traffic) data is important. In the current scenario, we use three types of publicly available test data [30]:
- a) generic generated cyclic streams,
- b) generated data streams which reflect typical I4.0 characteristics, and
- c) real captured data of a robot control environment, as summarized in Table II.

The generic streams (stream set $\Theta$) send maximum-sized Ethernet frames [1472 B application layer payload + headers of 8 B (UDP), 20 B (IP), 22 B (Ethernet, incl. VLAN tag and frame check seq.), i.e., 1522 B Ethernet frames at link layer], with distinct packet generation cycle times to represent three cyclic TSN streams.

The stream set $\Psi$ corresponds to a typical I4.0 scenario where a machine delivers tactile data in combination with video and audio streams for human control and mechanical error monitoring [73]. The cyclic video frames are lossy compressed, resulting in bursty compressed video frame sizes.

For stream set $\Omega$, we captured the data streams of the well-known robot spot [74]. The streams contain the acyclic control data stream of the spot robot in combination with the spot camera stream. For audio, we included a lossy compressed voice audio file.

Throughout, we transmitted the audio and video traffic streams (with fairly long cycles) jointly with the tactile stream (with the short 1 ms cycle) or the acyclic Spot control stream so as to assess the impact of cycle time heterogeneity.

To be replayed in the TSN-FlexTest testbed, the packet header needs to be modified. Figure 2 shows the packet structure of the TSN streams. The testbed uses several fields in the packet header for matching. The source Medium Access Control Layer (MAC) address is used for matching the source node name of a received packet. The DuT is configured to apply different queuing and scheduling strategies depending on the packet’s virtual LAN (VLAN) PCP ID. Each transmitted packet is timestamped automatically by the TX network hardware, which adds a 64-bit timestamp into the packet payload.

2) Measurement Procedure: The flowchart in Figure 3 illustrates the automated testbed measurement procedure. The testbed software, including data stream generators, are openly available [29]. The measurement procedure is structured into configuration, traffic replay, and record results for plotting. In the configuration phase, the control Personal Computer (PC) resets all network connections and sets the nodes into a dedicated state. Depending on the test configuration, the software TAS is configured at the transmitting nodes. To finish the preparation, the control PC transfers the TX pcaps to the nodes.

In the measurement phase, the control PC starts the capturing process at the receiving node and initiates the pcap replay at the transmitting nodes. If the replay or capture has finished, the received data is recorded and transferred to

### Table II

| Stream Set | Traffic Type | PCP | Peri- | $t_{ex}$ | Appl. Data Size | Bitr. |
|------------|--------------|-----|-------|---------|----------------|-------|
| $\Theta$   | Generic 1    | 6   | cyclic| 0.2     | 1472 (0.0)    | 61 M  |
| $\Theta$   | Generic 2    | 5   | cyclic| 0.3     | 1472 (0.0)    | 41 M  |
| $\Theta$   | Generic 3    | 4   | cyclic| 0.5     | 1472 (0.0)    | 24 M  |
| $\Psi$     | Tactile      | 6   | cyclic| 1       | 1012 (0.0)    | 1.15 M |
| $\Psi$     | Audio CBR    | 5   | cyclic| 24      | 480 (0.0)     | 177 k |
| $\Psi$     | Video VBR    | 4   | cyclic| 16.67   | 8336.4 (24283.8) | 4.1 M |
| $\Omega$   | Spot CTRL    | 6   | asycle| –       | 3016 (794.2)  | 775 M |
| $\Omega$   | Audio VBR    | 5   | cyclic| 20      | 1539 (46.6)   | 82 k  |
| $\Omega$   | Video VBR    | 4   | cyclic| 25      | 62412.5 (60355.9) | 20 M |
| Cross Traffic | 0 | asycle | –    | 1472 (0.0) | 2 G  |
the control PC for further processing. We noticed that some DuTs drop the PTP connection sporadically. Because the PTP synchronization is the backbone of the testbed we check the log for PTP errors or excessive clock derivations during the measurement and repeat the measurement if an error was detected.

We report the statistics of the measured one-way packet latencies with boxplots which represent the interquartile range from the first quartile $Q_1$ to the third quartile $Q_3$ as a box with the median marked by a horizontal line; the lower (upper) whisker indicates the minimum (maximum) latency that was measured within the range $[Q_1, Q_1 - 1.5(Q_3 - Q_1)]$ ($[Q_3, Q_3 + 1.5(Q_3 - Q_1)]$); outliers below (above) this range are indicated by dots.

IV. TSN-FLEXTEST TESTBED VALIDATION

Generally, a measurement is only as good as the metering setup. This section evaluates the measurement precision of our TSN-FlexTest testbed.

A. Evaluating the Precision Time Protocol (PTP)

As noted in Section III-C1, we designed the TSN-FlexTest testbed as a distributed architecture. In the distributed setting, clock synchronization via PTP is a key factor for precise delay measurement. For validating the testbed precision, we conducted measurements for several servo settings and examined the influence of the cross traffic on the precision of the LinuxPTP ptp4l [70], which we operated in the L2 multicast transport mode. To extract the clock synchronization metrics, we patched ptp4l to provide the actual clock deviation values and not only a root mean square (RMS) value. Previous measurements demonstrated that the synchronization metrics of ptp4l are as precise as physical measurements with an oscilloscope [75]. Figure 4 shows the results of the clock precision measurement. For simplicity and to avoid clutter in this evaluation, we manipulate only one ptp4l servo parameter, namely the synchronization frequency with the number of synchronization (sync) messages per second in the range from $[8, ... , 128]$.

We measured the clock deviation between one slave node and the DuT switch for several configurations with and without cross traffic in order to investigate the independence of the clock deviation from the cross traffic. Due to the activated hardware time stamping, the transmission of additional (cross traffic) packets increases the queuing delay of the PTP packets.

However, this additional queueing delay is compensated for by the timestamps (which are applied after the TX queue and before the RX queue). Thus, cross traffic does not affect the testbed precision. Generally, the physical transmission channel (including the physical layer chips) between a node and the switch has a jitter, which needs to be compensated by the servo. The results in Figure 4 indicate that the synchronization frequencies of 8 to 128 sync messages per second result in a reasonably small maximum clock deviation of less than $\pm 14$ ns, indicating a precision of approximately 30 ns. We use 8 sync messages/s, which is the proposed value in the standard, for the remainder of this article.

B. Sending Cyclic Data Precisely

As described in Section III-C2, different packet generation approaches have different advantages and precision levels. In Table III we extend the measurements from [1]. The results in Table III clearly indicate that all transmission techniques that use the Linux network stack (socket based and TA_PRIO) are less precise than sending packets via DPDK based generators, e.g., MoonGen. The substantial difference between the two tcpreplay versions is due to a bug in the lower version, which results in a constantly too short cycle. We generally use tcpreplay 4.3.4 for the TSN streams (and TA_PRIO in Section V-F).

C. Cut-through Switching: Effects of Ethernet Frame Size

Figure 5 shows the DuT one-way packet delay for cut-through switching. We generated a random homogeneous
TABLE III
MEASUREMENT RESULTS OF ACHIEVABLE PRECISION OF TRAFFIC STREAM CYCLE DURATION: MINIMUM, MEAN, MEDIAN, STANDARD DEVIATION, 99% PERCENTILE, AND MAXIMUM OF CONFIGURED CYCLE TIME (1 ms) MINUS MEASURED CYCLE TIME IN NANOSECONDS; WITH 8 SYNC MESSAGES/SEC. TYPE CHARACTERIZES THE SOFTWARE THAT GENERATES (OR SUBSEQUENTLY SHAPES) THE PACKETS ON SENDING NODES 1, 2, AND 3, NAMELY LINUX NEW API (NAPI), DPDK, OR SENDER TAS (WITH 15 \( \mu s \) OR 500 \( \mu s \) TIME SLOT DURATION IN GCL CYCLE)

| Type            | Tool   | Version | Min        | Mean        | Median       | Stddev       | P99%        | Max        |
|-----------------|--------|---------|------------|-------------|--------------|--------------|-------------|------------|
| Linux NAPI      | tcppreplay | 4.3.2    | -997176    | 149881      | 160763       | 29630        | 166339      | 447074     |
| Linux NAPI      | tcppreplay | 4.3.4    | -98312     | 4           | 48           | 11001        | 15960       | 2254136    |
| DPDK            | MoonGen | 25c61ec   | -6848      | 19          | 0            | 138.7        | 392         | 6895       |
| Sender TAS (15 \( \mu s \)) | TA_PRIO            | Kernel 5.15 | -997904    | 18          | -126         | 19205        | 24822       | 2958199    |
| Sender TAS (500 \( \mu s \)) | TA_PRIO            | Kernel 5.15 | -998784    | 336         | 2            | 1023658      | 2467535     | 6460670     |

Fig. 5. Packet latency of cut-through switch DuT as a function of packets size. The switch forwards link-layer Ethernet frames smaller than 337 B when they have been completely received; for larger frames, only 337 B are accumulated before forwarding.

data-set with frame sizes ranging from 64 B to 1518 B (at the link layer, w/o frame check seq.) and sent the packets through the DuT. With the testbed, we measured the packet residence time in the DuT and plot the measured delay as a function of the packet size. Packets of size 64 B to 337 B experience a linearly increasing delay. This is because the FibroLAN switch completely receives and stores these small packets before forwarding, incurring the equivalent of the transmission delay (which the packet experienced as the sending node transmitted the packet bits into the physical wire) as the packet bits are accumulated at the switch. For larger packets, the considered FibroLAN switch accumulates 337 B of data before starting to forward the packet; thus, incurring a constant equivalent transmission delay component for accumulating 337 B of data. The cut-through experiment should be conducted for every new DuT which is considered in the testbed, because the forwarding characteristics for different packet sizes depend on the DuT’s internal architecture and will affect all delay measurements [76].

V. TSN EVALUATION WITH TSN-FLEXTEST TESTBED

In this section, the TSN-FlexTest testbed is used to evaluate Time-Sensitive Networking (TSN). Due to space constraints, we focus on a few selected features of the measurements and refer the interested reader to [64] for the full details and results.

TABLE IV
CONSIDERED GATE CONTROL LIST (GCL) CONFIGURATIONS (1, 2, AND 3, FROM LEFT TO RIGHT) WITH OPTIONAL GUARD BANDS. EACH GCL CYCLE CONSISTS OF TIME SLOTS (TIME-AXIS, 15 \( \mu s \) TIME SLOTS AND 3 \times 15 \( \mu s \) = 45 \( \mu s \) TIME SLOTS ARE CONSIDERED) AND DIFFERENT STREAMS WITH ASSOCIATED PRIORITIES (PRIORITY-AXIS). THE TIME WHEN EACH STREAM IS PERMITTED TO SEND IS INDICATED BY A DARK-COLORED BOX. GCLS 2 AND 3 DO NOT NEED A DEDICATED TIME SLOT FOR A GUARD BAND, SINCE THE TO-BE-PROTECTED STREAM IS ALWAYS ALLOWED TO SEND DATA

| Time | Priority | GCL 1 | GCL 2 | GCL 3 |
|------|----------|-------|-------|-------|
| t1   | X        | X     | X     | X     |
| t2   | X        |       |       |       |
| t3   | X        |       |       |       |
| t4   | X        |       |       |       |
| t5   | X        |       |       |       |
| t6   | X        |       |       |       |
| t7   | X        |       |       |       |

A. Evaluation Setup

As illustrative examples for the Gate Control List (GCL), typically computed by scheduling algorithms [77], [78], [79], we consider the GCLs in Table IV: GCL configuration 1 (GCL 1 for brevity) is a simple round-robin. GCL 2 always opens the gate to the highest priority traffic, and other traffic holds the transmission time equally. In GCL 3, the GCL gate duration time corresponds to the priority, whereby the higher the priority, the longer the gate remains open.

We summarize the clock and synchronization settings as follows. The switch takes the role of the GM and is configured to transmit the synchronization and peer-delay measurement messages every 125 ms, which equals 8 messages per second.

Each generic stream measurement scenario was run for 12 Million GCL cycles. For selected spot robot scenarios, we conducted pilot measurements for 12 Million GCL cycles and for 1.2 Million GCL cycles. After confirming that the measurement runs for 1.2 Million GCL cycles and for 12 Million GCL cycles gave equivalent results, we ran each spot robot measurement scenario for 1.2 Million GCL cycles.

In order to provide QoS assurances for network streams and to reduce the PDV with TSN, there are multiple possible options: First, the architecture can be modified, e.g., a dedicated connection can be established. With dedicated connections, the delay increases caused by other network participants (streams) can almost completely be avoided, which provides a baseline benchmark for the following improvements. Second, if additional dedicated connections are either
physically infeasible or economically not viable, then existing technologies for data prioritization can be used. For instance, the PCP field of the Ethernet VLAN tag has 3 bit to distinguish 8 packet priorities, which can be utilized with SPQ to prioritize packets. We subsequently refer to the PCP approach as soft QoS. A third choice is to enhance network control: With a TAS, it is possible to achieve consistent packet transmission behavior, which we refer to as hard QoS.

We have conducted extensive measurements in the TSN-FlexTest testbed to elucidate the advantages and disadvantages of the different options. Furthermore, we examine for the first time the TAS with synchronized senders using COTS hardware. While TAS with synchronized senders has been employed with industry-grade equipment; to the best of our knowledge, we are the first to conduct detailed measurements for TAS with synchronized senders in a testbed built from basic COTS hardware. (Conventionally, sender and network switches are not synchronized.) Our measurements demonstrate that TAS with synchronized senders improves the performance of conventional “switch-only” TAS (i.e., without synchronized senders) to achieve near-optimal performance (that is comparable to the baseline) on generic commodity (basic COTS) hardware, while allowing for the sharing of the network links by multiple streams.

B. Baseline

The baseline measurement considers only one sender (actually, three senders, but only one is sending in a given measurement run) and one receiver, with the DuT connecting both. The baseline scenario replays all streams in Table II, one by one, without cross traffic. The baseline measurement thus provides the absolute minimum delay for traversing the DuT, whereby this minimum DuT traversal delay is mainly governed by the queuing delay, while switch processing delay and transmission delay are small and nearly constant for the constant frame size. Collisions (resource contention) can arise due to the in-band PTP synchronization, whereby PTP frames are sent with the highest priority (7, which is higher than the highest test stream PCP of 6), and thus the PTP frames can delay the data frame transmissions. The one-way packet latencies for the three streams within each stream set are combined together in Figure 6(a), although in fact three separate measurements have been performed. It is expected that the packet latencies correspond to the stream priorities, i.e., higher priority streams should have lower packet latencies. Additionally, the boxplots for the high-priority streams should indicate a minimal consistent latency, since the GCLs are optimized towards the highest priority.

Figure 6(a) shows the measured one-way packet latency, i.e., packet residence time inside the DuT. An identical residence time for all packets in stream set \( \Theta \) can be visually observed in Figure 6(a) from the minuscule boxplot spans. Quantitatively, we measured means of 4967 ns, 4967 ns, and 4969 ns, and standard deviations of 43.5 ns, 41.5 ns, and 38.9 ns for streams \( \Theta_{\text{high}}, \Theta_{\text{med}}, \text{and} \Theta_{\text{low}} \), respectively. Thus, the one-way delay distributions are very narrow (the values are in the range of the Physical Layer (PHY) tolerances), an almost perfectly consistent behavior.

In contrast, stream sets \( \Psi \) and \( \Omega \) have wider boxplots. This can be explained by the frame size: Whereas the streams in stream set \( \Theta \) have a fixed (link-layer) frame size of 1522 B, and the streams \( \Psi_{\text{high}} \) and \( \Psi_{\text{med}} \) have constant (link-layer) frame sizes of 132 B \( [-82 \text{ B payload } + 8 \text{ B (UDP), 20 B (IPv4), 22 B (Ethernet.)}] \) and 530 B, respectively (see Table II), all other streams have variable frame sizes. Therefore, the boxplots of stream set \( \Psi \) and stream set \( \Omega \) exhibit the actual distributions of the frame sizes and the corresponding transmission delays. For instance, a few frames of stream \( \Psi_{\text{low}} \) are smaller than 337 B (see Section IV-C) and therefore are forwarded faster. Additionally, the video streams have packet bursts, i.e., multiple Ethernet frames may be needed to transport a large compressed video frame, which causes a filled queue. Although the DuT can handle the stream, the in-band time-synchronization causes a slightly longer delay for a fraction of the video packets.

C. No Prioritization (Internet Scenario)

After having established a baseline, we investigate the one-way delay in a typical Internet scenario, i.e., the data streams are transmitted simultaneously with the same priority, i.e., the DuT ignores the VLAN PCP. In this case, the streams compete equally for resources in the DuT. Figure 7(a) shows two measurements: in the scenario without cross traffic, only the three TSN streams of stream set \( \Theta \) are transmitted simultaneously, while the scenario with cross traffic considers the more typical case with added interference caused by best-effort cross traffic. In all following measurements, the cross traffic consists of full 1522 B (link-layer) Ethernet frames, with a throughput of 2 Gbit s\(^{-1}\) (200% loaded link), and a variable inter-packet delay based on a Poisson distribution. In this Internet scenario, the packets of the three simultaneously transmitted TSN streams will have collisions and are affected by queuing latency at the egress port towards node0 at the DuT. This Internet scenario
reveals the scheduling behavior of the DuT if packets from several streams need to be transferred through the same output port (our synthetically created bottleneck). Thereby, we can observe the queuing mechanisms of the DuT in an over-saturation scenario that forces the switch to drop packets.

We observe from Figure 7(a) that the one-way delay is not aligned with the priority set in the packet headers. In additional measurements, we found that without cross traffic, 98% of all sent packets have the same one-way latency as in the baseline measurement (see Figure 6(a)). Above the 98% percentile, we found a latency increase for all stream priorities. This latency increase is caused by statistical collisions of frames, i.e., arriving frames compete for resources. These collisions (resource contentions) need to be considered for applications with hard real-time requirements. Statistical collisions of frames result in latency increases (with a staircase distribution) because sometimes a frame is forwarded immediately, whereas sometimes it has to wait, mainly because long frames which are already in transmission, are not interrupted if a high priority frame gets ready for transmission. This blocking issue could be solved by frame preemption. Additional cross traffic affects all streams equally and increases the packet delay in general (see Figure 7(a)); the mean packet latency increases from around 10 μs without cross traffic to around 50 μs with cross traffic.

The no prioritization configuration is typical for a net-neutral Internet, without specialized packet treatment and with limited resources in the backbone networks. This is especially true due to rush hours in the Internet, e.g., in the evening or at large events (Black Friday, popular broadcasting events). For Tactile Internet (TI) scenarios [80], [81], it is nearly impossible to ensure a prescribed service quality in such a net-neutral Internet.

**D. Soft Quality-of-Service With Strict Priority Queuing (SPQ)**

In a first step towards providing QoS, we examine the improvement of the service quality achieved by applying queuing disciplines that take the stream priorities into consideration. One common approach is Strict Priority Queuing (SPQ), defined by IEEE 802.1Q, where always the queue that holds the highest priority packets is emptied first. Figure 7(b) shows this SPQ approach with stream set Θ without and with added cross traffic. We observe from Figure 7(b), that for the less challenging scenario without cross traffic, the distribution of the one-way latencies tends to align with the priority levels, i.e., a higher priority stream has a slightly shorter median latency and 75% percentile of the latency. Thus, we conclude that SPQ achieves some rudimentary priority isolation between the three streams.

However, Figure 7(b) shows that when cross traffic is added, the overall average one-way latency still increases from 12.0 μs, 12.4 μs, and 14.2 μs to 16.7 μs, 20.8 μs, and 17.8 μs for stream Θ_{high}, Θ_{med}, and Θ_{low}, respectively. This is an increase by 44% on average for all three streams. Therefore, we characterize the SPQ approach as soft QoS, as SPQ does not enforce strict guarantees. This can be explained by the fact that lower priority frames can be scheduled and selected for transmission just prior to the arrival time instant of a higher priority frame at the DuT. In this case, the lower priority frame is not interrupted, and the higher priority frame must wait until the lower priority frame is transmitted, which for full regular Ethernet frames with 1522 B takes around 12.3 μs in the worst case. This effect can be observed in Figure 7(b): without cross traffic, the highest priority stream Θ_{high} is delayed from 4.894 μs (minimum value) to at most 18.540 μs, a growth by about one full Ethernet frame transmission.

**E. Hard Quality-of-Service With Time-Aware Traffic Shaping**

1) **Overview:** In a second step towards providing QoS, we measured the one-way packet latency for TAS. Figures 8 and Figure 9 show measurements for the GCLs in Table IV. We strive to gain insights for the different TAS parameters settings. We systematically created three different GCL configurations and, where appropriate, we added a guard band to protect the highest priority stream, see Table II. We added cross traffic in all nine measurements. Furthermore, we varied the slot size (the time a gate of the respective queue is open) from around 10 μs (minimum value) to at most 540 μs, 20 μs, and 17 μs.

2) **GCL Configuration 1:** We observe in Figure 8 the fundamental effects of the GCL settings: GCL 1 has a round-robin-like configuration in which every stream priority gets the same amount of transmission time. The goal of this configuration is to allocate equal bandwidth to all streams, without accounting for the different stream characteristics. However, we observe that the Θ_{high} stream experiences an increased tail-latency in the A_60 configuration, as indicated by the numerous outliers in the 90–600 μs range. This is due to the higher number of frames transmitted in the Θ_{high} stream compared to the Θ_{med} and Θ_{low} streams (see Table II). If the GCL is composed of only short slots for each single stream, then streams with higher frame frequency are more affected by blocked time slots than other streams—a frame from the previous GCL cycle which was buffered at the switch
The measurement configuration A leads to more outliers and a higher median latency for the medium-priority audio streams, while configurations B and D have fewer outliers, particularly for the high-priority streams. Increasing the slot size to 45 $\mu$s in measurement configuration A has an effect on the high-priority stream $\Psi_{high}$.

4) GCL Configuration 3: GCL 3 gives more transmission opportunities to higher priority streams, striving to isolate the stream priorities. We observe from Figure 9 that this prioritization works for stream sets $\Theta$ and $\Omega$ in the expected manner. In stream set $\Psi$, the bursty high-bitrate low-priority video VBR stream $\Psi_{low}$ has an effect on the high-priority stream $\Psi_{high}$. Specifically, for the short 15 $\mu$s time slots, the stream $\Psi_{low}$ can block the high-priority stream $\Psi_{high}$ from being transmitted, leading to numerous latency outliers for the high-priority stream $\Psi_{high}$ in measurement configuration $\Lambda_{\Psi}$. Increasing the slot size to 45 $\mu$s ($B_{\Psi}$), eliminates the $\Psi_{high}$ outliers at the expense of slightly increasing the median latency for $\Psi_{high}$ and quite substantially increasing the latencies for the lower priority streams $\Psi_{med}$ and $\Psi_{low}$.

5) Comparison of GCL Configurations: Overall, we observe from Figures 8 and 9 that GCLs 2 and 3 have the fewest outliers and the clearest differentiation between the stream priorities. It is important to note that the y-axis uses logarithmic scaling. Therefore, a larger box in the box plots indicates a vastly increased latency variation. Furthermore, the logarithmically scaled boxplot reveals outliers more clearly. Especially in GCL 1, both without and with guard band, there are (numerous) outliers, including several outliers for the high-priority streams.

6) Slot Size: With increased slot size, see measurement configurations B and D versus measurement configurations A and C in Figures 8 and 9, the overall delay generally increases. When gates are opened and closed for a longer period, frames from another queue have to wait longer for the next transmission window, i.e., there is a general latency-increasing effect due to the longer GCL cycle durations (that correspond to the longer slots). On the other hand, the tail-latency (maximum delay whiskers and maximum delay outlier values) can be slightly decreased due to the more relaxed time window when the gate is open and traffic bursts can be accommodated, e.g., for GCLs 1 and 2, see the reductions for the maximum latency outliers and whiskers for the bursty low-priority streams $\Psi_{low}$ and $\Omega_{low}$ for the configurations B and D vs. A and C, respectively.

7) Guard Band: Theoretically, a guard band should reduce the effect, that the tail of long frames blocks the next time slot. Measurement configurations C and D in Figure 8 investigate the effects of an added guard band. The guard
band should protect the highest-priority stream $\Theta_{\text{high}}$ from interference from the lower-priority streams $\Theta_{\text{med}}$ and $\Theta_{\text{low}}$ and is configured to always have a slot size of 15 $\mu$s, regardless of the slot length for the data, since the guard band should only cover the worst case: A full lower-priority Ethernet frame is enqueued for transmission right before the gate closes. If this occurs at the end of a GCL cycle (usually the highest priority gets the first transmission window in a GCL cycle), this could interfere and delay the next high-priority frame. Only the GCL configuration 1 could potentially benefit from guard bands, since the gate for the highest priority is always kept open in the GCLs 2 and 3.

Comparing the measurement configurations A and B without the guard band versus the measurement configurations C and D with the guard band in Figure 8 shows that the guard band has generally only a minor effect. On the positive side, we observe that the guard band reduces the maximum latency outliers for the generic high-priority traffic stream $\Theta_{\text{high}}$ (fewer outliers for C than for A). However, generally, the added guard band increases the latency metrics, mainly due to the reduction of the bandwidth that is available for the data traffic.

8) Traffic Burstiness: GCL 1 without guard band provides an equal channel bandwidth of 250 Mbit s$^{-1}$ for each stream. The bandwidth slightly changes if a guard band is applied: for the small 15 $\mu$s slot size, each stream has still 200 Mbit s$^{-1}$; and for large 45 $\mu$s slot size, there are 230 Mbit s$^{-1}$ available per stream. The streams in stream set $\Theta$ have fixed data rates of 61 Mbit s$^{-1}$, 41 Mbit s$^{-1}$, and 24 Mbit s$^{-1}$, respectively. Thus, without bursts, the data streams are not affected by congestion. In a 15 $\mu$s slot, a burst of two frames will be most likely (and a burst of three or more frames will certainly) not be forwarded immediately.

The influence of traffic burstiness can be observed by comparing the $\Psi_{\text{low}}$ boxes in Figures 8 and Figure 9. We observe that the stream $\Psi_{\text{low}}$ is generally strongly delayed because the GCL configurations do not respect the bursty characteristics of stream $\Psi_{\text{low}}$. However, we observe that increasing the GCL slot size to 45 $\mu$s slightly reduces the latencies for stream $\Psi_{\text{low}}$ (and for stream $\Omega_{\text{low}}$) for GCLs 1 and 2 since the longer slots generally accommodate the bursty streams better. For GCL 3, the increased slot size generally increases the latencies of all streams (except for stream $\Psi_{\text{high}}$ whose latencies become more consistent). However, the GCL 3 triples the transmission slots for the medium-priority streams compared to the GCL 2, while the transmission slots for the low-priority streams are only doubled (see Table II). Accordingly, the latency increases for the medium-priority streams (with the increased slot size) are generally substantially smaller with GCL 3 compared to GCL 2. On the other hand, the low-priority streams in GCL 3 can no longer freely utilize the increased slot size as the two low-priority transmission slots can now also be utilized by medium-priority streams (which was prohibited in GCL 2), leading to the slight latency increases for streams $\Psi_{\text{low}}$ and $\Omega_{\text{low}}$ (configuration B in GCL 3 compared to the latency decreases for GCLs 1 and 2).

9) Summary: Comparing the latencies for the hard QoS service with TAS in this section with the baseline latencies in Section V-B, and specifically in Figure 6(a), reveals that TAS latencies are substantially longer than the baseline latencies. One main explanation for the increased latencies with TAS is that in the conventional TAS operation that is considered in this section, the senders are not synchronized with the gates on the switch. In particular, the measurement runs were started randomly (compared to the GCL cycle of the switch). Also, as we have discussed in Section IV-B, our tools for sending data depend on the host machine’s precision. Therefore, frames arrive at the switch not necessarily when the corresponding gate is open. The frames are then enqueued and must wait until the gate opens again. This queuing is not efficient and can significantly increase the one-way latency.

F. Time-Aware Traffic Shaping With Synchronized Senders

While TAS scheduling can provide stream prioritization, the TAS comes at the expense of reduced bandwidth and increased latencies for all streams. An optimally configured TAS setup should have all components configured so as to avoid collisions. If a data source is sending traffic already periodically (cyclically), the source can be synchronized to the network.

In this section, we examine for the first time a strategy to achieve near minimal latency and PDV in a measurement testbed using COTS hardware. Generally, the Linux kernel provides tools to use a software TAS (whereby the Linux kernel could not support a 15 $\mu$s slot size, therefore a 150 $\mu$s slot size was used in this section). A software TAS can align (time-synchronize) the sender nodes with the switch, since the network testbed nodes are already time-synchronized via PTP with the DuT. The examined synchronized-sendners approach uses buffering on the sender to shape the packet traffic even before it is transmitted by the sender node. Essentially, we distribute the GCL configuration on the switch over the network nodes with some adjustments for path delay and other system inaccuracies [28, Sec. V-D1]. This is realistic as in real systems, such as in robotics systems, the application traffic stream cycle of the robot nodes can be adjusted to be time-synchronized with the TAS network.

The boxplots in Figure 6(b) characterize the one-way packet latencies of the generic traffic streams $\Theta$ for TAS with synchronized senders. With this sender node synchronization, the packet delay caused by queuing in the network can be reduced to the same minimum, as in the baseline measurement for the $\Theta$ traffic flows in Figure 6(a). Thus, we conclude that with the synchronized senders, i.e., with a distributed TAS that extends to the senders, we can achieve essentially the same low one-way latencies as in our baseline measurement (when only a single stream was transmitted in Figure 6(a)), but now with added best-effort cross traffic of 2 Gbit s$^{-1}$ transmitted at the same time in Figure 6(b). In Figure 6(b), there is close to no PDV for the streams from stream set $\Theta$ while the DuT is forced to drop at least half of the incoming data packets since the streams in stream set $\Theta$ have 61, 41, and 24 Mbit s$^{-1}$ and the cross traffic has 2 Gbit s$^{-1}$, while the outgoing link of the DuT provides only 1 Gbit s$^{-1}$. However, despite this overload condition, the one-way latency remains at a minimum, i.e., 4.968 $\mu$s and a standard deviation of 40 ns for the high-priority stream $\Theta_{\text{high}}$ while the one-way latencies of the medium and low-priority streams $\Theta_{\text{med}}$ and $\Theta_{\text{low}}$ are only negligibly longer; thus, only best-effort cross traffic is dropped.
In summary, we conclude that TAS with synchronized senders, achieved as a software TAS with the Linux kernel, allows arbitrary interference, such as high-bandwidth file sharing, on the same physical link while providing consistent low-latency QoS to critical packet traffic flows. We acknowledge two caveats to the TAS with synchronized senders approach: First, the software TAS using the current Linux kernel cannot support the very short time slots on the order of ten microseconds of the TAS on the TSN DuT in Section V-E; instead, time slots need to be lengthened to the order of hundred microseconds for software TAS. Second, the packet latency measurements for the TAS with synchronized senders assume that the packets are ready for transmission just as sender’s transmission slot commences. If the timing characteristics of the application generating the data packets cannot be synchronized to the TAS, then the packets would still incur a latency between their actual generation time instant and the time instant when the sender’s transmission slot commences.

VI. CONCLUSION

We introduced TSN-FlexTest, a flexible highly-precise TSN measurement testbed, for evaluating TSN features. Following a comprehensive review of the available hardware and software, we designed TSN-FlexTest with Commercial-off-the-Shelf (COTS) hardware and open-source software to foster further studies, allowing low-cost TSN testbed measurements. The validation of the TSN-FlexTest testbed highlighted the following features: The underlying PTP clock synchronization provides nanosecond precision, which enables our high-precision TSN measurements. The flexible cyclic traffic generator empowers researchers to reproduce various stream configurations and thus aids in finding proper GCL configurations for particular network and traffic scenarios.

We conducted extensive evaluation studies with our TSN-FlexTest testbed with multiple stream sets. We measured KPIs for widely used Quality-of-Service (QoS) configurations, including net-neutral transmissions, Strict Priority Queuing (SPQ), and the usage of a Time-Aware Shaper (TAS). We found that the TAS can provide an upper bounded one-way packet latency, although the Packet Delay Variation (PDV) may vary significantly in certain cases, especially for bursty data streams, e.g., VBR video traffic. A properly configured Gate Control List (GCL) is critical for achieving bounded packet latencies. We found that the GCL configuration is challenging and is highly dependent on the stream traffic pattern (e.g., burstiness, packet sizes). Our TSN-FlexTest testbed enables flexible measurements to evaluate GCL configurations and thus aids in finding proper GCL configurations for particular network and traffic scenarios.

We identified the sender node behavior as the root cause of the high PDV: randomly time-shifted data packet transmissions by sender nodes can introduce high packet latencies due to possibly closed gate states at a switch. We conducted measurements with sending nodes synchronized to the state of the TSN switch. By leveraging solely COTS hardware and open-source software in the TSN-FlexTest testbed, we have thus been able to achieve a level of QoS that is comparable to a dedicated link while accommodating multiple parallel streams, with a more than 200% over-saturation of the link. We make the source code of the TSN-FlexTest testbed, including the software TAS configuration for synchronized senders, publicly available [29]; also, the traffic streams [30] are openly available.

A. Future Work

The proposed TSN-FlexTest testbed expands opportunities for investigating new methods and for evaluating existing methods through measurements on a real hardware testbed. We proceed to summarize some potential future research directions. A first step could be to investigate the effectiveness of other TSN standards, such as Frame Preemption (FP), Frame Replication and Elimination for Reliability (FRER), and the Stream Reservation Protocol (SRP). Future work could also consider a wider set of traffic profiles. Furthermore, the TSN-FlexTest testbed topology could be extended to multiple switches and a larger number of sending and receiving nodes in future work.

Evaluating commercial industry-grade hardware components on the market can reveal additional information about their behaviors and performance levels and may provide insights for academic researchers. For example, using a time-coordinated CPU in the testbed could potentially improve the precision of the TSN network, which could be investigated further. Also, the configuration can have a significant impact on the TSN testbed performance. Consequently, one important future work direction is to design a framework for determining proper pre-configurations for static topologies or for reconfiguring parameters according to changes in traffic profiles and network resources over time.

Finally, integrating TSN with wireless technologies can significantly improve flexibility and mobility, which can be advantageous for a wide range of new use cases [4], [82]. Integrating the 5G System (5GS) with a TSN network is considered in recent 3GPP standards [83] in an architecture where the 5GS acts as a virtual TSN bridge. The TSN-FlexTest testbed can be employed to develop a 5GS in accordance with recent 3GPP standards to enable measurement-based performance evaluations for a 5G-TSN network.

REFERENCES

[1] 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 Proc. IEEE 5th Int. Conf. Netw. Softwarizat. (NetSoft), 2022, pp. 402–410.

[2] J. Falk, H. Geppert, F. Dürr, S. Bhowmik, and K. Rothermel, “Dynamic QoS-aware traffic planning for time-triggered flows in the real-time data plane,” IEEE Trans. Netw. Service Manag., vol. 19, no. 2, pp. 1807–1825, Jun. 2022.

[3] V. Gavriliuţ, A. Pruski, and M. S. Berger, “Constructive or optimized: An overview of strategies to design networks for time-critical applications,” ACM Comput. Surveys, vol. 55, no. 3, p. 62, 2022.

[4] T. Hoeschele, C. Dietzel, D. Kopp, F. H. Fitzek, and M. Reisslein, “Importance of Internet exchange point (IXP) infrastructure for 5G: Estimating the impact of 5G use cases,” Telecommun. Policy, vol. 45, no. 3, 2021, Art. no. 102091.

[5] Y. Seol, D. Hyeon, J. Min, M. Kim, and J. Paek, “Timely survey of time-sensitive networking: Past and future directions,” IEEE Access, vol. 9, pp. 142506–142527, 2021.

[6] Y. Huang, S. Wang, T. Huang, and Y. Liu, “Cycle-based time-sensitive and deterministic networks: Architecture, challenges, and open issues,” IEEE Commun. Mag., vol. 60, no. 6, pp. 81–87, Jun. 2022.
W. Quan, W. Fu, J. Yan, and Z. Sun, “OpenTSN: An open-source project,” in Proc. IEEE LCN, 2021, pp. 241–248.

“OMNeT++ Discrete Event Simulator.” Jul. 2022. [Online]. Available: https://omnetpp.org/

I.-H. Liu, L. Y. Chang, J. S. Li, and C. G. Liu, “Industrial control system cybersecurity testbed with TSN feature,” in Proc. Int. Conf. Artif. Life Robot. (ICAROB), 2022, pp. 291–294.

M. M. Tajiki, S. H. G. Petroudi, S. Salsano, S. Uhlig, and I. Castro, “Optimal estimation of link delays based on end-to-end active measurements,” IEEE Trans. Netw. Service Manag., vol. 18, no. 4, pp. 4730–4741, Dec. 2021.

W. Quan, W. Fu, J. Yan, and Z. Sun, “OpenTSN: An open-source project for time-sensitive networking system development,” CCF Trans. Netw., vol. 3, no. 1, pp. 51–65, 2020.

V. D. Naudts, S. Hadiwardoyo, D. van den Akker, J. Marquez-Barja, and I. Moerman, “Experimental V2X2 evaluation for C-V2X and ITS-G5 technologies in a real-life highway environment,” IEEE Trans. Netw. Service Manag., vol. 19, no. 2, pp. 1521–1538, Jun. 2022.

N. Makris, C. Zarafetas, A. Valantasis, and T. Korakis, “Service orchestration over wireless network slices: Testbed setup and integration,” IEEE Trans. Netw. Service Manag., vol. 18, no. 1, pp. 482–497, Mar. 2021.

K. Polachan, J. Pa, C. Singh, and T. V. Prabhakar, “Assessing quality of service orchestration over wireless network slices: Testbed setup and integration,” IEEE Trans. Netw. Service Manag., vol. 19, no. 4, pp. 5348–5365, Dec. 2022.

M. R. Fida, F. Ocampo, and A. Elmokashfi, “Measuring and localising congestion in mobile broadband networks,” IEEE Trans. Netw. Service Manag., vol. 19, no. 1, pp. 366–380, Mar. 2022.

O. Michel, J. Sonchack, G. Cusack, M. Nazari, E. Keller, and J. M. Smith, “Software packet-level network analytics at cloud scale,” IEEE Trans. Netw. Service Manag., vol. 18, no. 1, pp. 597–610, Mar. 2021.

P. Emmerich, S. Gallenmiller, D. Rauner, F. Wohlhart, and G. Carle, “MoonGen: A scriptable high-speed packet generator,” in Proc. ACM Internet Meas. Conf., 2015, pp. 275–287.

R. Kuldell, F. Siegmund, J. Blendin, A. Rizk, and B. Koldehoe, “PASTA: High performance packet timestamping with programmable packet processors,” in Proc. IEEE/IFIP NOMS 2020, 2020, p. 1–9.

M. Kassouf, L. Dupont, J. Béland, and A. Fadlallah, “Performance of the precision time protocol for clock synchronisation in smart grid applications,” Trans. Emerg. Telecommun. Technol., vol. 24, no. 5, pp. 476–485, 2013.

E. Kyriakakis, J. Sparso, and M. Schoebeli, “Hardware assisted clock synchronisation with the IEEE 1588–2008 precision time protocol,” in Proc. ACM RTNS, 2018, pp. 51–60.

S. Sudhakaran, K. Montgomery, M. Kashif, D. Cavalcanti, and R. Candell, “Wireless time sensitive networking for industrial collaborative robotic workcells,” in Proc. 17th IEEE Int. Conf. Fact. Commun. Syst. (WFCS), 2021, pp. 91–94.

P. Kell et al., “Prototype of 5G integrated with TSN for edge-controlled mobile robotics,” Electronics, vol. 11, no. 11, p. 1666, 2022.

T. Agarwal, P. Niknejad, F. Rahmani, M. Barzegaran, and L. Vanfretti, “A time-sensitive networking-enabled synchronized three-phase and phasor measurement-based monitoring system for microgrids,” IET Cyber-Phys. Syst. Theory Appl., vol. 6, no. 1, pp. 1–11, 2021.

W. J. P. Xiao, Y. Huang, and D. Wernzer, “Multi-domain time-sensitive networks—An east-westbound protocol for dynamic TSN-stream configuration across domains,” in Proc. 24th IEEE Int. Conf. Emerg. Technol. Fact. Autom. (ETFA), 2019, pp. 1363–1366.

M. Böhm, J. Ohms, and D. Wernzer, “Multi-domain time-sensitive networks—An east-westbound protocol for dynamic TSN-stream configuration across domains,” in Proc. 24th IEEE Int. Conf. Emerg. Technol. Fact. Autom. (ETFA), 2019, pp. 1363–1366.

M. Bosk et al., “Methodology and infrastructure for TSN-based reproducible network experiments,” IEEE Access, vol. 10, pp. 109203–109239, 2022.

“TSN-FlexTest Repository.” 2023. [Online]. Available: https://github.com/5GCampus/tsn-testbed

“IEEE DataPort: TSN-FlexTest Traffic Streams for Spot Robot, Tactile Internet, and Generic Data.” 2023. [Online]. Available: https://dx.doi.org/10.21227/4eyw-n176

L. Maile, K.-S. Hielscher, and R. German, “Network calculus results for TSN: An introduction,” in Proc. IEEE IECIT, 2020, pp. 131–140.

L. Thomas, A. Miflœlon, and J.-Y. L. Boudouc, “Worst-case delay bounds in time-sensitive networks with packet replication and elimination,” IEEE/ACM Trans. Netw., vol. 30, no. 6, pp. 2701–2715, Dec. 2022.

L. Zhao, P. Pop, and S. Steinhorst, “Quantitative performance comparison of various traffic shapers in time-sensitive networking,” IEEE Trans. Netw. Service Manag., vol. 19, no. 3, pp. 2899–2928, Sep. 2022.

M. H. Farzaneh and A. Knoll, “Time-sensitive networking (TSN): An experimental setup,” in Proc. IEEE VNC, 2017, pp. 23–26.

“INET Framework.” Jul. 2022. [Online]. Available: https://inet.omnetpp.org/
