Design and Deployment of a Flexible COTS-based TSN Testbed

Fabian Kummer*, Michael Nast*, Helge Parzyjegla†, Peter Danielis†, Christian Haubelt*, Frank Golatowski*
*Institute of Applied Microelectronics and Computer Engineering, †Institute of Computer Science
University of Rostock, 18051 Rostock, Germany
firstname.lastname@uni-rostock.de

***Abstract*—Time-Sensitive Networking (TSN) extends Ethernet with deterministic communication and is therefore well suited to latency-sensitive industrial applications. For time-triggered traffic with hard real-time requirements, transmissions must be scheduled along defined paths at precise times to achieve low end-to-end latency and bounded jitter. Although many scheduling methods have been proposed, their practical applicability, particularly in dynamic scenarios, requires experimental validation. This paper presents a testbed with commercial off-the-shelf hardware for evaluating static and dynamic TSN schedules in real systems. The testbed supports automated experiments and runtime reconfiguration, enabling the investigation of dynamic TSN scheduling methods. We present the testbed architecture, the time-synchronization configuration, the controller software, and initial measurements of real-time performance in a static scenario under load.**

***Index Terms*—Ethernet, Testbed, Time-Sensitive Networking**

I. Introduction

Time-Sensitive Networking (TSN) [1] extends Ethernet to support deterministic communication. It enables the transmission of hard real-time traffic with low end-to-end latency and bounded jitter and allows critical and non-critical traffic to coexist within the same local network. These capabilities make TSN a promising technology for realizing real-time subnetworks in future network-of-networks architectures, as envisioned for 6G systems. Representative application scenarios include smart factories, where multiple robots must coordinate their actions under strict timing constraints. Such scenarios require real-time-capable networks that support deterministic communication while remaining adaptable to changing requirements. However, online scheduling under hard real-time constraints requires further investigation, especially regarding experimental validation on physical hardware.

To investigate these challenges, this paper presents a TSN testbed for evaluating online scheduling algorithms for hard real-time traffic using commercial off-the-shelf (COTS) hardware. We describe the testbed architecture, including its hardware and software components, the time synchronization configuration, and the controller software that provides a fully automated platform for deploying and evaluating TSN schedules. In addition, we compare Time-Aware Shaper (TAS)-based and strict-priority (SP) scheduling under different static configurations and load conditions.

II. Related Work

In [2], a TSN testbed was developed to evaluate TSN schedules in a practical system. Our work builds on this approach and extends the system to enable the evaluation of dynamic scheduling methods while also integrating a fully automated test environment. A TSN testbed for evaluating switch characteristics that also includes an automated test environment is presented in [3]. The latencies of generated streams were investigated for different frame sizes, forwarding settings, traffic shapers, and interfering traffic. Our approach follows a similar design. However, beyond switch evaluation, it focuses on dynamic scheduling under hard real-time constraints in a testbed with runtime-adaptable stream requirements, topology, and configuration. In [4], a TSN testbed for avionics networks with a REST-based central controller is presented. The controller generates YANG-based configurations and deploys them automatically to the network components via a Python-based NETCONF client.

Figure 1. Photograph of the TSN testbed showing the three endpoints up-1 to up-3, the two switches sw-1 and sw-2, and the controller ctrl.

III. Testbed Architecture

A. Testbed Setup

Fig. 1 shows our testbed, which comprises two switches, three endpoints, and one controller. The endpoints up-1 to up-3 are located in the foreground, the controller ctrl is on the left, and switches sw-1 and sw-2 are on the right. The endpoints are three UP SQUARED PRO 7000 EDGE systems, each equipped with an Intel Atom x7425E processor and two Intel i226 network interfaces supporting up to 2.5 Gbit/s. The controller is a PicoSYS 2683 embedded PC equipped with an Intel Core i7-8565U processor and eight Intel i210 interfaces supporting up to 1 Gbit/s. Both the Intel i210 and i226 support hardware timestamping, enabling generalized Precision Time Protocol (gPTP)-based time synchronization in accordance

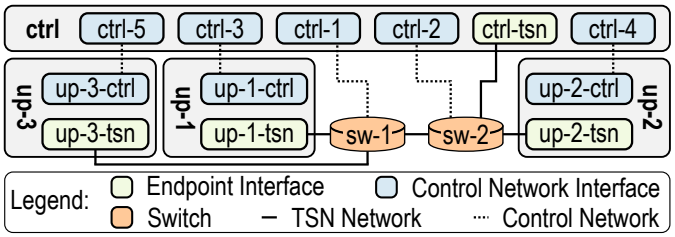

Figure 2. Testbed topology with separate TSN data and control network.

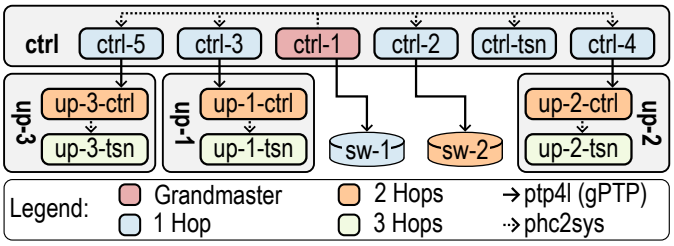

Figure 3. Time synchronization structure with `ptp4l` and `phc2sys`.

with IEEE 802.1AS [5]. The controller and endpoints run Ubuntu 24.04 with kernel version 6.8.1-1042-realtime. The switching infrastructure consists of two KSWITCH D10 MMT 8G devices, each providing six 1 Gbit/s and two 2.5 Gbit/s interfaces and supporting the TSN functions relevant to this work, including gPTP and TAS.

Fig. 2 shows the testbed topology, comprising two isolated networks: a TSN network for application traffic and a control network for management traffic. In the control network, all endpoints and switches are connected directly to the controller. The control network is used for SSH-based configuration, gPTP-based time synchronization, and collection of measurement data at the controller. The switches do not support NETCONF natively, so SSH-based configuration serves as a substitute.

The TSN network is used exclusively for communication between endpoints. Within this network, sw-1 and sw-2 are interconnected by a 1 Gbit/s link. Endpoint up-1 is connected to sw-1 via 1 Gbit/s, up-3 to sw-1 via 2.5 Gbit/s, and up-2 to sw-2 via 1 Gbit/s. The isolated controller interface ctrl-tsn acts as an additional TSN endpoint connected to sw-2 via 1 Gbit/s. All endpoints support the Earliest TxTime First (ETF) qdisc [6], which allows time-triggered (TT) frames to be scheduled for transmission at precisely defined times. For experiments requiring synthetic traffic, we developed a traffic generator based on UDP sockets and hardware timestamping. Configuring the TAS on the switches enables deterministic forwarding of TT streams at scheduled times. This topology, with two endpoints per switch and a shared inter-switch link, is a minimal yet representative setup for investigating contention on shared links in a potential TSN backbone network.

B. Time Synchronization

The endpoints and switches use gPTP for time synchronization over the control network. This separates gPTP traffic from the TSN network under investigation. Synchronization over the TSN network is also possible, enabling measurements under the influence of gPTP traffic. LinuxPTP [7] is used on the controller and endpoints for network-based synchronization with `ptp4l` and for local clock synchronization with `phc2sys`. The switches use their integrated gPTP functionality. For `ptp4l` and `phc2sys`, the configurations are adapted to the gPTP profile [5]. Both run on isolated CPU cores to minimize runtime variations and their impact on time synchronization. To increase the resilience of time synchronization against load-dependent runtime variations that cause significant frequency adjustments and oscillatory servo behavior, `phc2sys` was modified with a sliding median-based outlier filter to remove high-delay outliers during time queries. Fig. 3 shows the synchronization structure of the testbed. Each synchronization step, whether over the network or between clocks on one device, is defined as one hop. The clock of interface ctrl-1 serves as the grandmaster. The clocks of all other interfaces, including the system clock, are synchronized to this clock using `phc2sys`. Switch sw-1 is synchronized directly to ctrl-1 via `ptp4l`. Endpoints up-1, up-2, up-3, and switch sw-2 are synchronized with `ptp4l` via their control network interfaces. In the final step, the TSN network interfaces and the system clock of the endpoints are synchronized internally on each endpoint using `phc2sys`.

C. Controller Software

We further developed the controller software from [2] to operate the testbed. The controller provides a Flask-based web server and a FastAPI-based REST API for managing devices, streams, the topology, and stream requirements. Fig. 4 shows the web interface for creating TT streams by specifying frame size, period, source, and destination. Users can choose whether a stream uses real data or is generated by the UDP traffic generator. The controller integrates the scheduler from [8], which computes a schedule for all TT streams from the topology, stream requirements, and configuration. Based on this schedule, the controller configures the filtering database, routing-table entries, and TAS configuration on the switches, as well as VLAN and priority settings on the endpoints. In addition to TT streams, it can create best-effort (BE) streams, for example to generate interfering traffic for schedule evaluation.

The controller can also execute automated measurements for predefined or runtime-created schedules, as shown in Fig. 4. The user specifies the number and duration of measurements for a schedule, which the controller then deploys. After the measurement period, all timestamps of TT and BE streams, together with the `ptp4l` and `phc2sys` data collected during the measurement, are automatically aggregated and made available for download. Besides static configuration with predefined stream requirements, the controller also supports dynamic scheduling, allowing streams to be created, deleted, and modified at runtime.

IV. Measurement Results

To demonstrate the system functionality, we evaluate a TT stream scenario from up-1 to up-2 with a frame size of 1 500 B and a period of 1 ms. Measurements are conducted with and

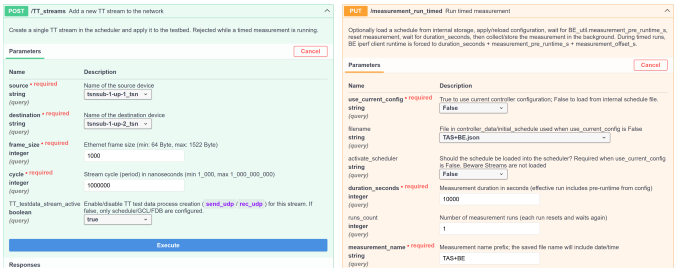

Figure 4. Web interface for creating TT streams (left) and performing automated measurements (right).

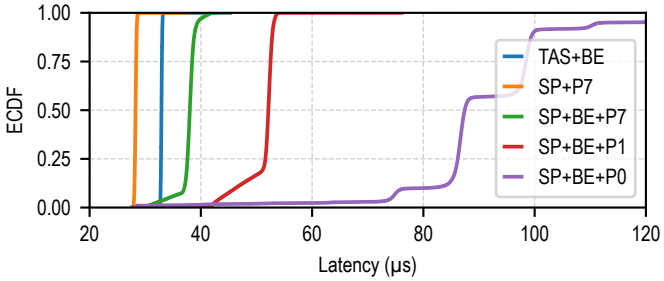

Figure 5. ECDF of the end-to-end latency for TAS and different strict-priority (SP) configurations over 10 000 s. The measurements consider a TT stream with a frame size of 1 500 B and a period of 1 ms from up-1 to up-2, with an optional BE stream from up-3 to up-2 at 900 Mbps.

without TAS, using only Ethernet SP. To expose resource contention at the switch egress interfaces, an additional BE stream with priority 1 from up-3 to up-2 is generated at 900 Mbps. Fig. 5 shows the empirical cumulative distribution function (ECDF) of the measured latencies. The ECDF directly represents the observed latency samples without assuming an underlying theoretical distribution. `TAS+BE` denotes the case with TAS enabled, the TT stream assigned to priority 7, and an active BE stream. In this configuration, guard bands are inserted before the transmission slots, isolating the scheduled time window and eliminating SP-level scheduling conflicts. `SP+P7` denotes the best case in which the TT stream is transmitted in isolation without BE traffic and TAS. In the `SP+BE` configurations, both the TT stream and a BE stream are active without TAS. Accordingly, the TT stream has higher priority in `SP+BE+P7`, equal priority in `SP+BE+P1`, and lower priority in `SP+BE+P0`.

`TAS+BE` yields a median latency of 32.92 μs with a standard deviation of 0.11 μs, whereas `SP+P7` achieves 28.26 μs and 0.16 μs, respectively. `TAS+BE` has a slightly higher latency due to the additional queuing delay required to protect the scheduled transmission windows, but it still maintains jitter comparable to the isolated-stream reference case `SP+P7` despite competing BE traffic. In contrast, the `SP+BE` configurations show that using SP without TAS is insufficient for deterministic timing. Even `SP+BE+P7`, where the TT stream has higher priority than the BE stream, exhibits a substantially increased median latency of 38.02 μs and a standard deviation of 1.66 μs. With equal priority in `SP+BE+P1`, the median latency and standard deviation further increase to 52.13 μs and 3.44 μs, respectively. With lower priority in `SP+BE+P0`, both latency and jitter become highly unpredictable. Even in this simple scenario, the median latency rises to 87.4 μs with a standard deviation of 1 067.38 μs. In more complex scenarios with multiple competing streams of equal or higher priority, larger delays and jitter are expected as resource conflicts become more frequent. Overall, this example indicates that TAS-based transmission scheduling substantially reduces latency variation for TT streams under competing BE traffic.

In all configurations except `SP+BE+P0`, where congestion occurs, the median offset between the planned and measured transmission time at the endpoint is 14 ns, with a standard deviation of 25 ns. The median synchronization offset is 33 ns for one hop, 45 ns for two hops, and 60 ns for three hops, with corresponding standard deviations of 37 ns, 24 ns, and 131 ns.

V. Conclusions

In this paper, we present a testbed for configuring and evaluating TSN networks in static and dynamic scenarios with real and generated time-triggered traffic under load. We describe the testbed architecture, its hardware and software components, the time synchronization configuration, and the controller software for automated experiments. The measurements indicate that the Time-Aware Shaper enables highly precise time-triggered transmission with very low latency and jitter on COTS hardware, even under load. Future work will investigate the impact of exclusive Time-Aware Shaper slots on gPTP synchronization and dynamic reconfiguration under hard real-time requirements. The testbed will also be integrated into a 6G network-of-networks prototype to evaluate distributed controller software, inter-subnetwork time synchronization, dynamic reconfiguration, and flexible deployment concepts.

Acknowledgment

This work was conducted in the German 6G-Health project and funded by the German Federal Ministry of Research, Technology and Space under reference number 16KISK227.

References

[1] "IEEE standard for local and metropolitan area networks – bridges and bridged networks," *IEEE Std 802.1Q-2022 (Revision of IEEE Std 802.1Q-2018)*, 2022, doi: 10.1109/IEEESTD.2022.10004498.

[2] W. Brekenfelder, H. Parzyjegla, P. Danielis, G. Mühl, F. Kummer, E. Schweissguth, and F. Golatowski, "Testing time-sensitive network schedules for practical applicability," in *IEEE WFCS 2025*, 2025, pp. 1–2, doi: 10.1109/WFCS63373.2025.11077615.

[3] M. Ulbricht, S. Senk, H. K. Nazari, H.-H. Liu, M. Reisslein, G. T. Nguyen, and F. H. P. Fitzek, "Tsn-flextest: Flexible tsn measurement testbed," *IEEE Transactions on Network and Service Management*, vol. 21, no. 2, pp. 1387–1402, 2024, doi: 10.1109/TNSM.2023.3327108.

[4] L. Castro-Lara, P. Vera-Soto, S. Fortes, V. Escaño, R. Ortiz, and R. Barco, "Deployment of a testbed for validation of tsn networks in avionics," *Aerospace*, vol. 12, no. 3, 2025, doi: 10.3390/aerospace12030186.

[5] "Ieee standard for local and metropolitan area networks–timing and synchronization for time-sensitive applications," *IEEE Std 802.1AS-2025 (Revision of IEEE Std 802.1AS-2020)*, pp. 1–491, 2025, doi: 10.1109/IEEESTD.2025.11302966.

[6] Linux manual page, *tc-etf(8)*, 2018. [Online]. Available: https://man7.org/linux/man-pages/man8/tc-etf.8.html

[7] R. Cochran. (2026) The linux ptp project. [Online]. Available: https://linuxptp.sourceforge.net/

[8] F. Kummer, F. Golatowski, W. Brekenfelder, H. Parzyjegla, P. Danielis, and G. Mühl, "An online scheduler for reconfigurable time-sensitive networks," in *IEEE ETFA 2024*, Sep. 2024, pp. 01–08, doi: 10.1109/ETFA61755.2024.10711137.