On Hard Realtime Traffic in Converged Time-Sensitive Networks

Willi Brekenfelder, Helge Parzyjegla, Peter Danielis, Gero Mühl
Institute of Computer Science
University of Rostock
18051 Rostock, Germany
firstname.lastname@uni-rostock.de

Fabian Kummer, Eike Schweissguth, Frank Golatowski
Institute of Applied Microelectronics and Computer Engineering
University of Rostock
18051 Rostock, Germany
firstname.lastname@uni-rostock.de

***Abstract*—Time-Sensitive Networking (TSN) extends IEEE Ethernet with realtime capabilities providing standardized support for multiple traffic classes within a shared network. While TSN-capable devices are becoming increasingly available, practical experience with managing traffic of varying realtime requirements in a converged infrastructure remains limited. In this paper, we present insights into the hardware and software configuration of a TSN testbed designed to support data streams of mixed criticality. In an evaluation, we apply different methods for isolating hard realtime streams from other traffic and analyze their impact on latency and jitter.**

I. Introduction

The convergence of operational technology (OT) and information technology (IT) in Industry 4.0 demands a unified, fast, and reliable communication infrastructure. *Time-Sensitive Networking* (TSN) enables the shift from fragmented and specialized systems (e. g., traditional Ethernet and fieldbuses) to a converged infrastructure where industrial and office devices communicate seamlessly with guaranteed performance and Quality of Service (QoS). To facilitate this, the IEEE 802.1Q standard [1] extends Ethernet to accommodate multiple traffic classes including time-triggered (TT), rate-constrained (RC) and best-effort (BE) traffic within a shared network. TT hard realtime communication transmits messages periodically with fixed latencies and minimal jitter. Exclusive communication slots are preplanned and reserved along the network path from talker to listener. For RC data streams, bandwidth is allocated along the path, and traffic shaping mitigates burstiness while limiting latency and jitter. BE traffic utilizes remaining resources but must not be unduly disadvantaged.

As TSN standards mature, a growing number of compatible devices become available that also drive the development of test methodologies to assess their compliance [2]. However, experience with installations involving multiple switches, talkers, and listeners with diverse realtime requirements remains scarce. This is due to the complexity of coordinating various aspects including switch and network card capabilities, realtime task design, network planning and configuration, device synchronization, and time measurement for evaluation. In this paper, we present initial measurement results for TT realtime streams in a converged network infrastructure and share lessons learned from setting up the testbed.

The remainder of the paper is structured as follows: Section II describes the testbed's hardware and software that is used in the case study, whereas Sect. III discusses the stream setup, the TSN configuration as well as the measurement results and their analysis. Finally, Sect. IV summarizes and concludes the paper.

II. Hardware and Software Setup

The hardware setup includes six Kontron KBox A-230-LS switches, each with four TSN-capable Ethernet ports, and four Supermicro SYS-1019C-FHTN8 servers, each equipped with eight Intel I210 Ethernet ports that support hardware timestamping. All components are mounted in a mobile 19-inch rack as shown in Fig. 1. Each TSN switch connects to a central configuration server (yellow cables) and is interconnected with other switches in a mesh structure (red cables). Two measurement servers handle traffic generation: one for TT hard realtime traffic (cyan cables) and the other for RC data flows or BE traffic (blue cables). A fourth server stores measurement results. White and black cables provide network access to the servers and their baseboard management controllers (BMCs) for remote administration. Two smart power distribution units (PDUs) with eight sockets each are installed on the rack's backside and allow remote power cycling of individual servers and switches in case of misconfigurations.

We implemented a custom TSN network controller that fulfills the roles of the *Central User Configuration* (CUC) and *Central Network Configuration* (CNC) as defined in IEEE 802.1Q [1]. The controller offers a web interface based on a REST API for remote administration and simplifies the deployment of inherently complex TSN configurations for TT streams by automating rollout and installation. Automation was necessary because the TSN switches lack standard configuration interfaces such as NETCONF. Instead, they run Yocto Linux for embedded devices and require manual configuration via secure shell (SSH). A dedicated command-line tool is used to configure the TSN ASIC including Gate Control Lists (GCLs) for Time-Aware Shaping (TAS), VLAN IDs for stream identification, and prepopulated Forwarding Database (FDB) entries for preventing forwarding loops. The latter is essential since the switches do not support separate spanning trees

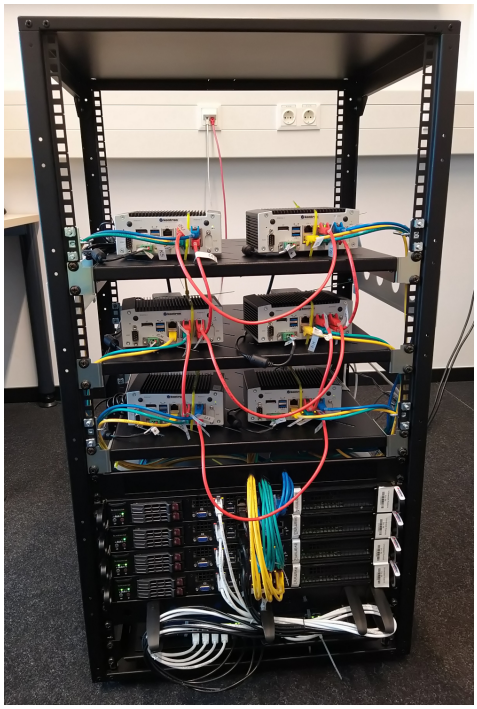


Figure 1: TSN testbed with six switches and four servers.

for different VLANs which limits the benefits of the mesh topology (in particular w.r.t. fault-tolerance).

The controller's operations are defined in an OpenAPI specification from which an interactive web-based user interface is generated using Swagger UI. Its logic is implemented in Python with the Flask web framework. The controller supports uploading a topology file that describes network nodes, links, and characteristics (e. g., bandwidth, propagation delay, and processing latency) and a traffic file specifying realtime stream parameters (e. g., source, destination, period, maximum latency, and maximum frame size). Based on this data, an incremental scheduling heuristic adapted from Dijkstra's algorithm selects the earliest available time slots along the links on the communication path from talker to listener. Since streams are scheduled sequentially, additional streams can be interactively added or removed later. The result is a communication schedule with derived GCLs for each TSN switch port that the controller deploys to the switches. Alternatively, users can upload precomputed schedules from more advanced schedulers such as [3]. Furthermore, the controller offers convenience functions to start and stop measurements and retrieve results from end stations.

Besides running the TSN controller, the configuration server acts as the reference clock for the TSN switches and other servers. Time synchronization is performed using the *generalized Precision Time Protocol* (gPTP) as specified in IEEE 802.1AS [4]. To simplify the setup, we synchronize the time out-of-band via the switches' management ports. Although this approach introduces greater inaccuracies, it eliminates the need to consider synchronization traffic when planning TT streams easing experimentation.

All servers run Ubuntu 22.04 LTS. The measurement servers act as end stations and can start arbitrary talker and listener processes. For each stream, a dedicated VLAN with a virtual interface is created. Talkers use the *Earliest TxTime First* (ETF) network scheduler [5] leveraging hardware-based transmission arbitration via Launch Time support of the Intel network interface controller (NIC). Additionally, we record hardware timestamps when packets are actually transmitted. Each packet's payload includes its stream ID and sequence number. By combining this information with the hardware timestamp from the listener, we determine a packet's latency.

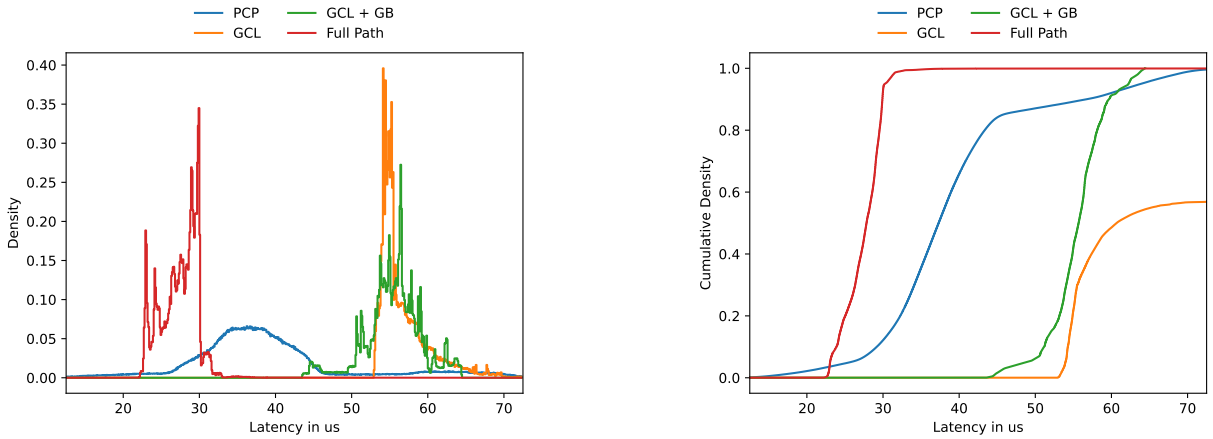
III. Case Study

To demonstrate the testbed, we evaluate different methods to isolate hard realtime streams from other traffic in a converged network infrastructure. We set up a TT stream that transmits an Ethernet frame of 1500 bytes every 500 µs from one measurement server to the two TSN switches on the rack's middle shelf (see Fig. 1) over their connecting link and back to the server. This ensures a common time base for the talker and listener. Please note that the server's clock and the Intel NIC hardware clocks are separate but locally synchronized via phc2sys. Furthermore, we use iperf to generate two 500 Mbit/s BE streams from the second measurement server. Each stream additionally passes two of the four remaining TSN switches and is steered to also cross the central link on the middle shelf alongside with the TT traffic. Together, these BE streams fully saturate the central link.

To protect the TT stream, we assign priorities, reserve exclusive communication slots along the path from talker to listener, and schedule guard bands as protection windows. In our first experiment, we assign the TT traffic a higher *Priority Code Point* (PCP) than the BE traffic. This makes sure that the TSN switches queue TT and BE packets separately with TT traffic prioritized for forwarding and thus allowing it to overtake BE traffic.

In our second experiment, we additionally reserve an exclusive communication slot of 12,160 ns per link and cycle for the TT frame. During this slot, the gates of all other queues remain closed except for the TT queue with the highest priority. However, our TSN switches do not fully comply with the standard's frame transmission rules for scheduled traffic (cf. [1, Annex Q]) allowing a BE frame to extend into a reserved TT slot if its transmission is not yet complete. To prevent this, our third experiment introduces a *Guard Band* (GB) equal to the size of a maximum Ethernet frame before the TT slot. In this protection window, the gates of all switch queues are closed making sure that an ongoing transmission is finished until the TT slot starts.

Finally, in our fourth experiment, we use GBs and reserve a single large TT slot of 66,480 ns across the entire path from talker to listener applying the same GCL entry (i. e., open and close times for the TT queue) for all switches along the route. Thus, the path is fully reserved for the TT frame allowing for an uninterrupted transmission. Please note the length of the TT slot accounts not only for the frame's aggregated transmission



(a) Latency distribution.

(b) Cumulative latency distribution.

Figure 2: Measured latencies of TT packets.

time but also for propagation and processing delays as well as an estimated safety margin to compensate inaccuracies in time synchronization between host and switches.

In all experiments, we measure the latency of transmitted TT packets. The results are shown in Fig. 2. When assigning priorities only (blue curve, PCP), the packet latency seems to be normally distributed following a bell curve in the beginning, but also has an additional tail (Fig. 2a). The jitter is substantial leading to a slowly increasing *Cumulative Density Function* (CDF) over a long range (Fig. 2b). This is because TT frames have to wait despite prioritization for ongoing transmissions of BE frames to complete. In contrast, reserving exclusive time slots for TT traffic significantly reduces jitter indicated by faster increasing CDFs. But without GBs (orange curve, GCL), only less than 60% of TT packets arrive within 70 µs while the rest miss their deadline and are delayed to the next period. This happens when TT packets are initially blocked by BE traffic causing them to miss their scheduled slot at the next switch. In this case, they are buffered until the missed slot repeats in the next cycle. Introducing GBs (green curve, GCL+GB) eliminates this issue making sure that all packets arrive in the same cycle they are sent. Finally, reserving a single TT slot across the entire path (red curve, Full Path) significantly improves latency (Fig. 2) because packets are never buffered at intermediate switches. However, this approach requires the most resources blocking the entire path for other traffic. When scheduling individual TT time slots per link, blocking is shorter, but additional latency is introduced due to conservative estimations to guarantee that frames arrive at the next switch before their reserved slot begins.

Please note that, in addition to the conducted measurements above, an experiment with *Frame Preemption* (FP) (cf. [1, Annex S]) would have been valuable. FP allows a higher-priority Ethernet frame to interrupt the transmission of a lower-priority frame at boundaries of 64 bytes in order to reduce the forwarding delay for critical data. The transmission of the preempted frame is resumed once the higher-priority frame has been sent. Although our TSN switch advertised FP as a supported feature, we were not able to make it work reliably. Furthermore, time synchronization was a major challenge during the case study. This not only involved synchronizing different devices (i.e., servers and switches) but also ensuring alignment between multiple hardware clocks within the servers (i.e., the host system and NICs). Repeating the experiments with time synchronization performed in-band over the mesh network of the TSN switches may yield further insights.

IV. Conclusions

In this paper, we presented a converged time-sensitive network infrastructure designed for experimenting with data streams of mixed criticality. We discussed the hardware and software setup and shared key lessons learned. In a case study, we evaluated different methods for traffic isolation and analyzed their impact on latency and jitter. For future work, we aim to improve the testbed's time synchronization and want to leverage the gained practical insights to refine our scheduling methodologies in order to produce more robust and reliable communication schedules.

References

[1] IEEE Standards Association, "IEEE standard for local and metropolitan area networks – bridges and bridged networks," *IEEE Std 802.1Q-2022*, pp. 1–2163, Dec. 2022, DOI: 10.1109/IEEESTD.2022.10004498.

[2] 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, Apr. 2024, DOI: 10.1109/TNSM.2023.3327108.

[3] E. Schweissguth, P. Danielis, D. Timmermann, H. Parzyjegla, and G. Mühl, "ILP-based joint routing and scheduling for time-triggered networks," in *25th International Conference on Real-Time Networks and Systems (RTNS 2017)*. Grenoble, France: ACM, 2017, pp. 8–17, DOI: 10.1145/3139258.3139289.

[4] IEEE Standards Association, "IEEE standard for local and metropolitan area networks – timing and synchronization for time-sensitive applications," *IEEE Std 802.1AS-2020*, pp. 1–421, Jun. 2020, DOI: 10.1109/IEEESTD.2020.9121845.

[5] J. Sanchez-Palencia and V. C. Gomes, *tc-etf(8) – Linux manual page*, Jul. 2018, URL: https://man7.org/linux/man-pages/man8/tc-etf.8.html.