Time-Sensitive Networking and Software-Defined Networking: an experimental setup for realistic performances

Claudio Zunino*, Manuel Cheminod*, Stefano Vitturi*, Alberto Morato*, Elena Ferrari†,
Dave Cavalcanti‡, Susruth Sudhakaran‡, and Federico Tramarin§
*National Research Council of Italy, CNR-IEIIT
Email: {alberto.morato, claudio.zunino, manuel.cheminod, stefano.vitturi}@cnr.it,
† Dipartimento di Ingegneria dell'informazione, Università di Padova
Email: elena.ferrari.7@phd.unipd.it,
‡ Intel Labs, Intel Corporation, Hillsboro, Oregon, USA
Email: {dave.cavalcanti, susruth.sudhakaran}@intel.com,
§Department of Engineering "Enzo Ferrari", University of Modena and Reggio Emilia, Modena, Italy
Email: federico.tramarin@unimore.it

***Abstract*—The ever more used and widespread Time-Sensitive Networking (TSN) has changed real-time networks, enabling reliable communication for time-critical applications. At the same time Software-Defined Networking (SDN) has emerged as a solution to ensure proper quality of service for managing dynamic network configurations even in evolving topologies. This paper investigates the integration between TSN and SDN to enable dynamic network re-configuration and improve performance for time-critical applications. We present an experimental setup designed to evaluate this approach. The setup focuses on scenarios where an SDN controller can dynamically reroute critical data flows across different TSN network devices to avoid interference and maintain consistent Quality-of-Service (QoS).**

***Index Terms*—Time-Sensitive Networking (TSN), Software-Defined Networking (SDN), Seamless routing, Traffic shaping and scheduling**

I. Introduction

The introduction of IEEE 802.1 Time-Sensitive Networking (TSN) [1] is revolutionizing the design and implementation of industrial networks and applications. Comprising over 20 standards, TSN addresses synchronization (802.1AS-2020) [2], traffic shaping and scheduling (802.1Qbv) [3], network management (802.1Qcc) [4] and other issues.

Originally conceived to introduce real time performance on Ethernet networks, TSN is nowadays being extended to wireless networks such as WiFi and 5G [5], [6]. Noticeably, TSN revealed promising not only for the aforementioned applications, but also in the context of reconfigurable network architectures in conjunction with Software-Defined Networking (SDN), a paradigm originating from Information Technology. SDN offers programmable network architectures through centralized control entities, enhancing the capabilities of TSN and enabling dynamic network reconfiguration. This synergy promises improved performance, synchronization, and easier integration of time-sensitive applications.

Recent research explores the integration of SDN and TSN to tackle challenges in industrial automation, mixed-criticality control, dynamic path reconfiguration, and failure handling. However, existing studies primarily focus on legacy TSN applications within Ethernet networks.

For example in [7], the authors introduce the concept of a unified control-plane that bridges the gap between Time-Sensitive Networking (TSN) and Software-Defined Networking (SDN). Depending on specific connection requirements, the proposed solution enables the establishment of either deterministic or non-deterministic paths. The approach has been validated using a testbed. Another paper on this topic is [8] where a solution that combines TSN, SDN, and OPC-UA is presented: the proposed architecture demonstrates the feasibility of configuring time-sensitive traffic streams by harnessing TSN methods for synchronization and Time Aware Shaping, alongside forwarding techniques from SDN. The OPC UA acts as the link for industrial applications, facilitating interaction with network management and efficient transmission of their service demands.

In [9] a SDN-TSN approach is proposed to enforce temporal isolation for flows that were provided during network design. By doing so, flow delays remain unaffected even when new flows are introduced dynamically at runtime. The solution has been tested on a preliminary experimental assessment based on commercial-off-the-shelf TSN-compliant switches.

An emulation based approach to achieve improved network resource management, while meeting high-reliability Quality of Service (QoS) requirements is presented in [10]. The performance analysis carried out by the authors demonstrates that the proposed solution accurately schedules critical time traffic, resulting in bounded end-to-end latency with minimal jitter.

This work has been partially funded by the EU Horizon Europe SNS JU PREDICT-6G (GA 101095890) Project.

A SDN-based Dynamic path reconfiguration approach is presented in [11], where an algorithm is proposed and formulated as an optimization problem. By leveraging the control plane's global view, the authors evaluate, by simulation, various dynamic path configuration strategies under deterministic communication requirements.

In [12] an architecture is presented that integrates wireless TSN with Software-Defined Networking (SDN), aiming to facilitate dynamic network configuration and enhance the performance of time-sensitive applications. A practical test environment is introduced, based on a hybrid network configuration that includes both wireless and wired TSN links.

Moving from the work described so far, in this paper we address the use of SDN to dynamically reconfigure TSN-based communication paths. In detail, we present an experimental setup which employs TSN switches and [13] nodes to achieve

- Evaluation of end-to-end latency to identify actual communication delays.
- Dynamic routing of time-sensitive traffic across the network based on latency measurements for reliable, low-latency communication.

In TSN applications, where guaranteed low latency and reliability are crucial, the ability to seamlessly update data flow paths is critical. Assessing this capability is a key objective of our paper. We present a test environment utilizing a network configuration managed by an SDN controller. Through practical setups, low cost devices and experimental sessions, we investigate the seamless transition of traffic flows between different links while maintaining performance levels, particularly in scenarios where communication issues arise on some paths.

The paper is structured as follow: in Section II our proposal based on TSN and SDN is introduced, while in Section III the setup we used for experimental evaluation is described. Section IV reports on the results we obtained, and Section V draws some conclusions.

II. Dynamic path re-configuration using SDN

As will be detailed in the next Sections, this paper presents a technique to dynamically and seamlessly reconfigure the paths between network nodes when the traffic on the current path becomes critically delayed. The overall system is based on the measurement tool described in [14] that relies on the synchronization of two nodes that act, respectively, as sender and receiver. This approach is based on how TSN's time synchronization acts as the groundwork for the proposed measurement method. Leveraging the ETF (Earliest TxTime First) mechanism, we optimize frame transmission precision and timeliness at the network's link layer, ensuring frames are sent with nanosecond-level accuracy. Simultaneously, at the receiving end, we utilize the SO_TIMESTAMPING API from the Linux kernel to access hardware timestamps. By integrating TSN features with this API, it has been possible to obtain a dependable and precise assessment of latency, focusing solely on network communication delays regardless of device setup and performance.

Once the latency measurement system is installed and operational, it exploits the measured values to determine, in the event of latency increases, any path changes necessary to bring the latency values back below a predefined threshold. In other words, after setting up and activating the measurement system, it continuously monitors latency levels. If it detects an increase in latency beyond a given threshold, the system automatically adjusts the network path to restore latency values to an acceptable level.

This operation is performed by the SDN controller that monitors the messages received by nodes and decides whether or not paths have to be dynamically changed. Path reconfiguration, if necessary, is triggered by messages from the controller to the Open vSwitch node.

To evaluate the network devices under different stress conditions we increase the network load along the path between *sender* and *receiver* by directly injecting interfering traffic that flows in the same direction of the main traffic.

To stress the network setup under different conditions and to evaluate different aspects of this TSN topology, four experiments were performed

1) interfering traffic with no reconfiguration
2) interfering traffic with reconfiguration on the OSW switch, to change the path used by traffic with high priority
3) interfering traffic with reconfiguration on the TSW_1 switch, to isolate the different types of traffic
4) interfering traffic with priority conflicting with the main traffic

III. The experimental setup

The experimental setup is shown in Fig. 1. It comprises two main nodes, referred to as *sender* and *receiver*. Both of them are NUCs Intel NUC11TNKI5, Ram 16GB DDR4-3200, SSD M.2 250GB PCIe. They are both equipped with an Ethernet card with chipset Intel I225, able to supply TSN capabilities, including hardware timestamping, hardware management of ETF and the PTP protocol. The *sender* has been programmed to generate the probe traffic, based on UDP datagrams with a payload of 256 bytes, that includes the *TX_TIME* value used as launch time for each packet by the ETF scheduler. This scheduler leverages the *offload* capability of the network card, effectively delegating the packet sending operations to the network card hardware. Moreover, to further limit any possible delay due to the software layer, a set of datagram are pre-loaded and sent to the Ethernet card until the buffer of the interface is full. As soon as a number of frames are sent, others are pre-loaded, in a continuous loop. In this way, the software avoids any problems that can arise when a precise periodic operation is required at application level. Two additional identical NUCs (int_1 and int_2 in Fig. 1) are used to generate the interfering traffic. The software they use is *iperf3*. All the NUCs run an Ubuntu 22.04.3 LTS Linux distribution. The PC indicated as OSW in Fig. 1 runs Open vSwitch. It is a Dell OptiPlex 5000, with an Intel i7-12700 CPU, Ram 16GB DDR4-3200, HDD 4TB. It is equipped with a 4-ports network

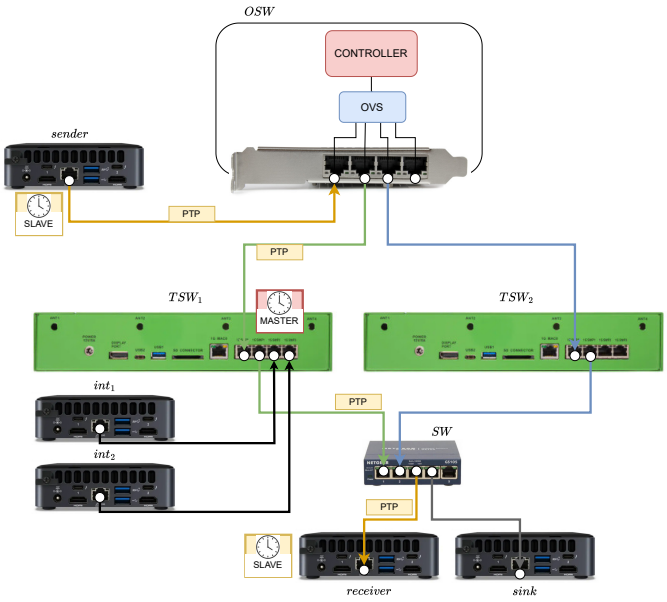

Fig. 1. Topology.

card with four interfaces based on the Intel chipset I225. This card is used to deploy the virtual switch used to create both paths from the *sender* to the *receiver*, exploiting either the switch TSW_1 or TSW_2. The Open vSwitch used is version 3.3.90. After some measurements, where we noticed a latency introduced by the OVS with values around $80\,\mu$s, we decided to leverage the Data Plane Development Kit (DPDK, version 23.11.0) to optimize the usage of the 4 network cards as a switch, with a remarkable improvement in terms of latency.

Both TSW_1 and TSW_2 switches are based on the NXP LS1028ARDB model. These switches are powered by an ARM Cortex-A72 dual-core processor clocked at 1.3 GHz and have 4 GB of RAM. Each switch features a 1Gbps Ethernet port utilized for control and monitoring operations. Then, the model supplies 4 1Gbps ports that permit switch functionality through Quad Serial Gigabit Media Independent Interface (QSGMII). This enables seamless connectivity to multiple network devices concurrently, all supporting Time-Sensitive Networking (TSN) and IEEE 1588 protocols.

An additional NUC was introduced in the configuration, namely the sink to draw the interfering traffic. To facilitate the convergence of the two paths leading to, respectively, the *receiver* and the sink, a further switch indicated as *SW* Fig. 1, was deployed that connects both such devices. *SW* is a 5-port Netgear GS105 Prosafe switch.

A picture of the real physical setup is shown in Fig. 2.

On the OSW node, a Ryu controller is running, awaiting REST commands dispatched by the receiver whenever the latency threshold is exceeded. This threshold is assessed every second. Considering a periodic traffic interval of 1 ms, the value is computed based on a moving average derived from a window of 1000 samples.

Both the *sender* and the *receiver* are synchronized using PTP with TSW_1 as the clock master, whereas the interfering nodes int_1 and int_2 are not synchronized at all.

These nodes are controlled via a simple command messages sent through the network at the beginning of each experiment. This command schedules a predefined waiting time on both interfering nodes, after which they both launch a *iperf3* instance to generate traffic with specific characteristics.

In each experiment the initial configuration allows the traffic to flow between the *sender* and *receiver* through the TSW_1 switch. To ensure this specific path, the Open vSwitch (OVS) was configured with the following static rules

- Traffic from *sender* to *receiver*: all incoming traffic from the sender is directed to switch TSW_1.
- Traffic from *receiver* to *sender*: similarly, all traffic from the receiver back to the sender is directed to TSW_1.
- REST Commands: an additional rule intercepts REST commands originating from the receiver and directs them to the Ryu controller.

.

TABLE I
HARDWARE DEVICES

Device	Latency (ns)				
	Avg	Min	p_{99}	Max	Std
TSW_1 and TSW_2	2711.2	2600	2788	3278	38.6
SW	4951.3	4827	5052	5086	56.29

IV. RESULTS

A. Setup evaluation

Before performing the experiments, some preliminary assessments have been carried out, to characterize the devices included in the paths between *sender* and *receiver*.

Table I summarizes the latency measurements for the key network components, namely, TSW_1, TSW_2, *SW*, (the columns refer to, respectively, Average, Minimum, 99th percentile, Maximum and Standard Deviation values).

As expected, the hardware switches (TSW_1, TSW_2, and *SW*) exhibit consistent (deterministic) latency values. These measurements have been carried out sending 1000000 samples, with a period of $1\,\mathrm{ms}$.

We have also evaluated the software switch measuring the total latency between the *sender* and the *receiver*. Because all the other component are deterministic, the greater variability in the measured latency is due to the software switch OSW. This evaluation has been carried out in this way, including two hardware switches because TSW_1 was used as the PTP master clock. It is worth highlighting that the performance of OVS configured with DPDK remains significantly better than a standard OVS installation on the same PC. In fact, the standard installation averages around 80 μs of latency.

While the latency statistics for the default path using the TSW_1 switch were previously presented in Table II, we include a corresponding plot here to highlight some key observations. As can be seen in Figure 3 there is an interesting anomaly in latency between samples 33,200 and 34,200. This

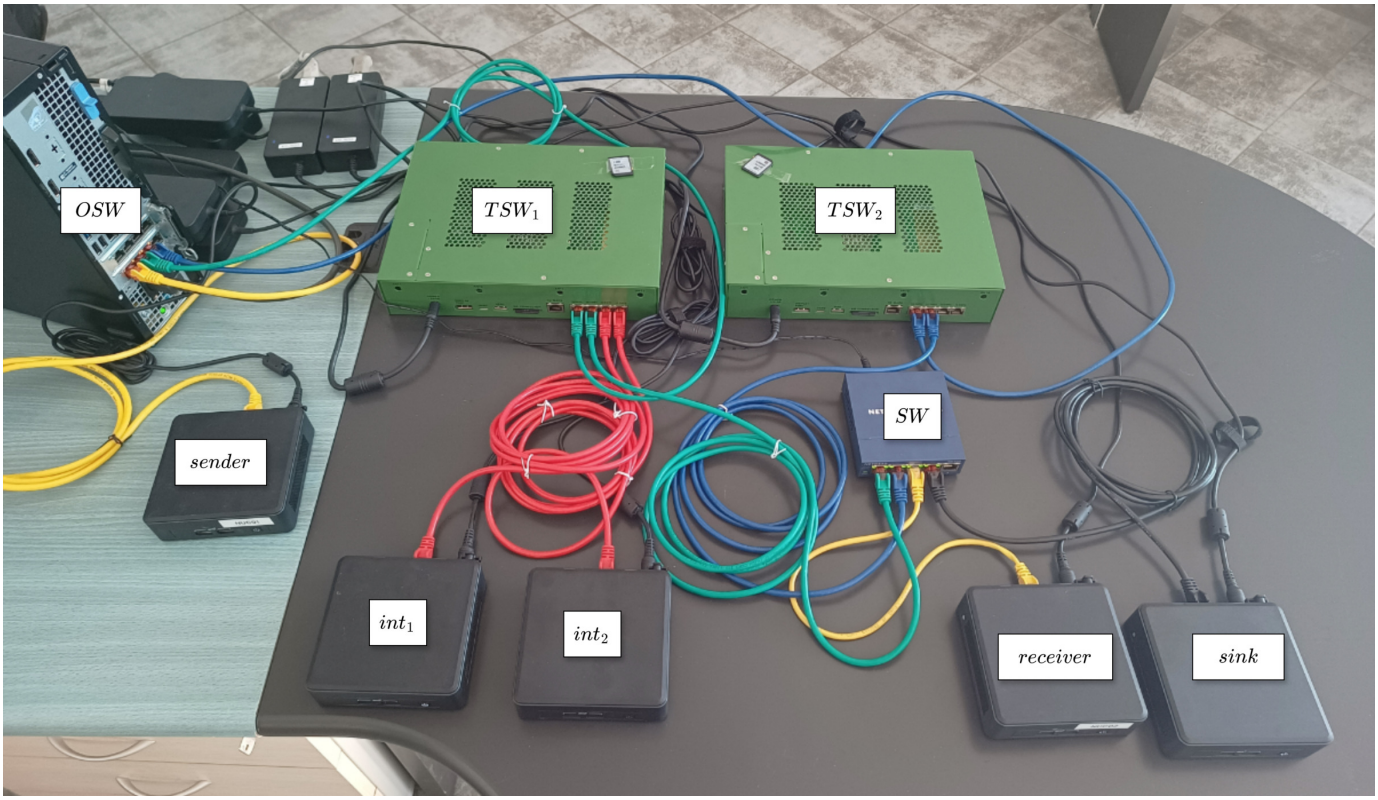

Fig. 2. Picture of the experimental setup.

TABLE II
sender TO *receiver* PATH THROUGH TSW_1

Latency (ns)				
Avg	Min	p_{99}	Max	Std
25530.3	14327	26358	33646	679.4

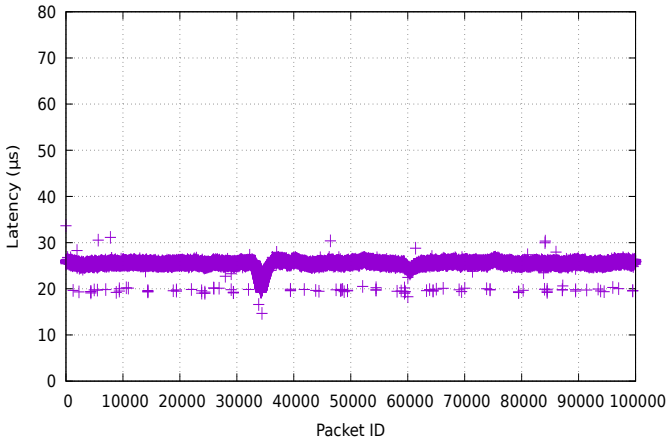

Fig. 3. Latency of the initial configuration, through TSW_1 switch.

appears as a dip followed by a rise, both lasting approximately 1 second. This behaviour is likely caused by an anomalous spike in the delay caused by the OSW node, affecting one PTP packet. This triggered the PTP algorithm to compensate and adjust the clock frequency accordingly on the *sender*, resulting in an offset between *sender* and *receiver* that is reflected in the overall latency measurement. This effect persisted until a new PTP frame arrived providing updated delay estimation, causing a new clock adjustment. Significantly, the duration of both ramps (1 second) aligns with the default PTP synchronization interval. We also observe a similar, albeit less pronounced, effect around sample 60,000. This anomaly exhibits a smaller change in latency compared to the one previously discussed.

After the above described initial characterization of the devices, we conducted a series of experiments to evaluate the proposed architecture. Each scenario was evaluated using 100000 samples generated at a constant rate of 1000 PPS (Packets Per Second).

B. First experiment: interfering traffic with no reconfiguration

The first experiment investigates the impact of interfering traffic on the TSW_1 switch. To achieve a more stable and significant level of interference, we employed two separate nodes, int_1 and int_2, running the *iperf3* tool, generating two different traffic flows. This approach proved more effective in saturating the TSW_1 switch, requiring a combined bandwidth of approximately $800\,\mathrm{Mbps}$ in total to produce a noticeable effect. The *sender* initiates the interfering traffic by sending a command to the two designated nodes, int_1 and int_2. These nodes are programmed to listen for this command. Upon receiving it, they wait for 15 seconds before starting *iperf3*.

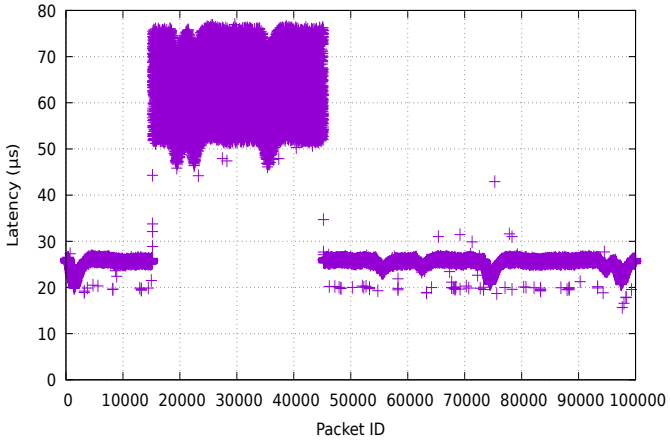

Fig. 4. Experiment 1: interfering traffic.

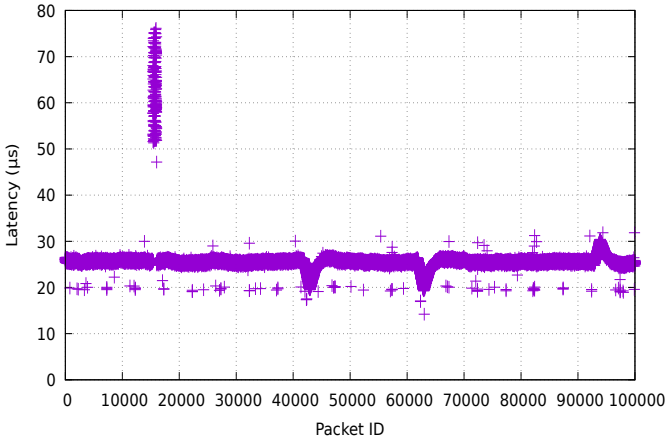

Fig. 5. Experiment 2: dynamic re-configuration.

Each *iperf3* instance is configured to transmit UDP traffic at a rate of $400\,\mathrm{Mbps}$ for a duration of 30 seconds.

Fig. 4 illustrates the impact of interfering traffic on latency. A significant increase in latency exceeding $75\,\mu\mathrm{s}$ is observed around sample 16000, coinciding with the activation of the two generators of interfering traffic using *iperf3*. Thereafter, the latency fluctuates in the range of $50\,\mu\mathrm{s}$ to $75\,\mu\mathrm{s}$. Once the two interference generators int_1 and int_2 cease their activity, the system returns to normal behavior and the latency values are again on average.

C. Second experiment: interfering traffic with reconfiguration on the OSW switch, to change the path used by traffic with high priority

In the second experiment, we introduce the dynamic re-configuration of the network. The *receiver* was configured to evaluate the latency on the fly using a simple moving average (SMA) on a window of 1000 samples. In practice, since the traffic under analysis has a generation period of 1 packet every $1\,\mathrm{ms}$, the average of the latency was evaluated considering all th packets acquired in the last elapsed second.

A threshold of $40\,\mu\mathrm{s}$ was configured, so that if the SMA exceeeds this threshold, the *receiver* sends a REST command to the controller running on OSW. On the OSW node there is a Ryu controller that has been configured, on the startup, to deploy the initial configuration to forward the traffic from the *sender* to the *receiver* using the TSW_1 switch. This controller, as soon as it receives a specific REST command from the *receiver*, triggers a rules change in the Open vSwtich to use the path through the TSW_2 switch. More specifically, to avoid any packet loss, the controller adds two rules that affect only the IP traffic. In this way, the PTP protocol frames can be still exchanged without any interruption between the master clock and the *sender*. It is to worth noting how new rules are added with higher priority while preserving existing rules. This ensures continuity on the connection and prevents packet loss during route changes.

In Fig. 5 the results of the experiment are shown. We can see how the system is working normally until the interfering nodes start to send their packets (always around sample 16000). But at this time, as soon as the calculated value of the mobile average goes over $40\,\mu\mathrm{s}$, the *receiver* reacts and makes the controller to change path, so that the latency is brought back to the former value. It may be concluded that the dynamic reconfiguration technique works appropriately and, consequently, it may applied in practice.

D. Third experiment: interfering traffic with reconfiguration on the TSW_1 switch

Until now we have used only the synchronization mechanism to evaluate latencies in real time, but we haven't yet used any of the features supplied by the TSN paradigm. In this third experiment, we'll demonstrate two key aspects of the use of TSN. First of all, we introduce the Time Aware Shaper (IEEE 802.1Qbv) on the TSW_1 switch to isolate different types of traffic. To reach this scope, we introduce traffic priorities and we assign the *sender* traffic to priority 3 (critical applications), leaving the interfering traffic as Best Effort (priority 0). For this purpose, the C program running on the *sender* uses the combination of *setsockopt* system call and *SO_PRIORITY* parameter. At the same time, the *sender* network card has been configured with a VLAN subinterface that, through the usage of *egress* settings, sets correctly the PCP field in the Ethernet frame reflecting the chosen priority value. We now provide a more complex and general configuration on the TSW_1 switch, defining 3 windows for 3 different types of traffic, where each window has a time duration specified as in the following:

- High priority traffic (priority 3): $300\,\mu\mathrm{s}$
- Intermediate priority traffic (priority 2): $300\,\mu\mathrm{s}$
- Best Effort traffic (all other priorities): $400\,\mu\mathrm{s}$

This configuration is applied on the TSW_1 switch using the Linux *TAPRIO qdisc* implementation. Moreover, the switch can leverage the full-offload feature meaning that the *taprio* configuration will pass the gate control list to the network card which will execute it cyclically in hardware.

With this configuration, the traffic under analysis and the interfering one will be managed separately, with the former one that will not be affected by the latter. From the point of view of the *receiver*, no effects on the measured latency will be noticed. However, in this scenario it is interesting to evaluate what happens during a configuration change. So, in this experiment, when the latency threshold is reached, we do not change the flow rules on the OSW node but instead we apply the Time Aware Shaper on TSW_1 and we analyse its impact.

A portion of the complete experiment results is shown in Fig. 6, where we reported the latency measurements in a time window of 15 seconds (corresponding to 15 thousand packets). Around the sample 16000 we notice again an increase in the latency, as expected, followed by a small cluster of packets suffering high latency (up to 30 ms), after which the latency returns to usual average values. We used a logarithmic scale on the y-axis to see all the data comprehensively but it is still possible to notice how the experiment initially shows the same behavior noticed in experiment 2. However, as soon as QBV settings are activated, the switch TSW_1 effectively separates the two kinds of traffic, protecting the one with high priority from the influence of the interfering nodes. We also see 28 packets suffering high delays of tens of milliseconds magnitude. These packets are included in the area delimited by green vertical lines in both plots, where the inner one uses a standard y-axis scale. We clearly see a regular decrease in the measured latency for these packets. The explanation is that, while the change in the configuration of the switch does not cause any packet loss, it, however, forces some packets to be enqueued and delayed. As soon as the configuration change ends, these packets are de-queued and are sent as fast as possible to their destination.

This experiment highlights the potential for dynamically adjusting transmitting window sizes based on traffic priority. While feasible, it also reveals two challenges: increased delay caused by network card reconfiguration and the need for a standardized switch reconfiguration protocol.

E. Fourth experiment: interfering traffic with priority conflicting with the main traffic

The last experiment we have carried out starts from the TSW_1 switch already configured with the Time Aware Shaper activated. We have then configured int_1 to use *iperf3* to send the interfering traffic, but we have changed the VLAN settings of the network card to re-map this traffic from Best Effort to the same priority of the one under analysis; in particular, the *egress* settings has been configured to change the priority from 0 to 3. With this scenario configuration, we have again the same situation as before: the interfering traffic impacts on the network performance and causes delays to the traffic under control, as can be seen in Fig. 7. Moreover, this time, the window reserved for the high-priority traffic is smaller (from $1\,\mathrm{ms}$ to $300\,\mu\mathrm{s}$) and a single interfering node, that generates a traffic with a $260\,\mathrm{Mbps}$ bandwidth, is enough to cause a delay that exceeds the threshold and that triggers the path change.

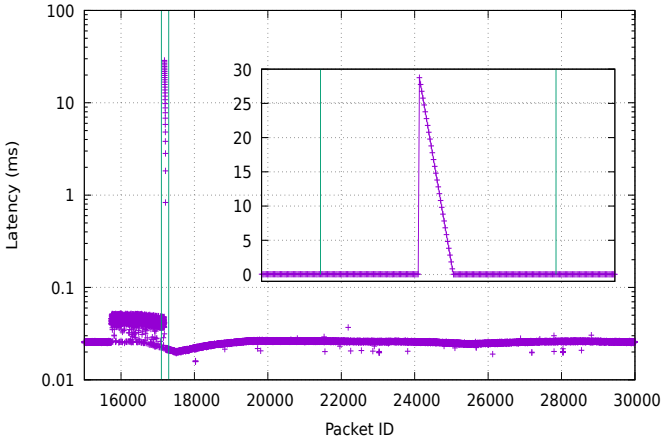

Fig. 6. Experiment 3: Time Aware Shaper on the TSW_1 and zoom for a comparison with other experiments.

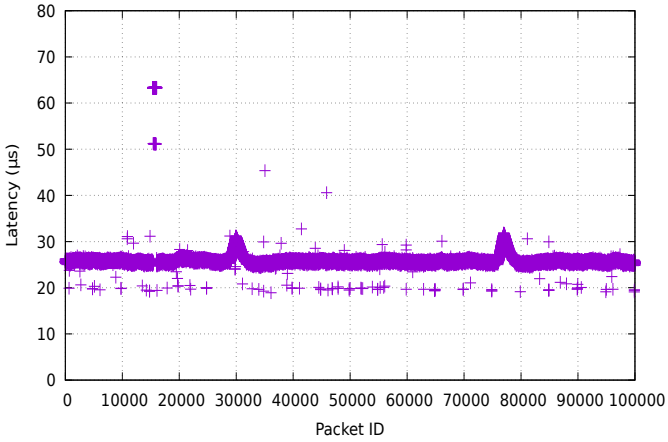

Fig. 7. Experiment 4: interfering traffic mapped to high priority and taprio configured on the TSW_1 switch.

V. Conclusions

This paper evaluated the integration of a TSN network with an SDN controller for dynamic reconfiguration. Our goal was to maintain Quality-of-Service (QoS) for time-critical data flows even in the presence of interfering traffic. We tested various scenarios in a real experimental setup, demonstrating the effectiveness of this approach and the pros and cons of the different used functionalities. While using an Open vSwitch introduced some non-determinism into the network, it still facilitated reliable and fast path switching without packet loss. On the other hand, another experiment yielded valuable insights into the potential for dynamically configuring Time-Aware Shapers on TSN switches that is effective but with some costs to take into account. Overall, the different experiments we conducted, demonstrated the necessity of having a flexible topology able to make use of both SDN and TSN capabilities, to adapt to different network load scenarios.

REFERENCES

[1] L. Lo Bello and W. Steiner, "A perspective on ieee time-sensitive networking for industrial communication and automation systems," *Proceedings of the IEEE*, vol. 107, no. 6, pp. 1094–1120, 2019.

[2] "Ieee standard for local and metropolitan area networks–timing and synchronization for time-sensitive applications," *IEEE Std 802.1AS-2020 (Revision of IEEE Std 802.1AS-2011)*, pp. 1–421, 2020.

[3] "Ieee standard for local and metropolitan area networks – bridges and bridged networks - amendment 25: Enhancements for scheduled traffic," *IEEE Std 802.1Qbv-2015 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, and IEEE Std 802.1Q-2014/Cor 1-2015)*, pp. 1–57, 2016.

[4] "Ieee standard for local and metropolitan area networks–bridges and bridged networks – amendment 31: Stream reservation protocol (srp) enhancements and performance improvements," *IEEE Std 802.1Qcc-2018 (Amendment to IEEE Std 802.1Q-2018 as amended by IEEE Std 802.1Qcp-2018)*, pp. 1–208, 2018.

[5] D. Cavalcanti, J. Perez-Ramirez, M. M. Rashid, J. Fang, M. Galeev, and K. B. Stanton, "Extending accurate time distribution and timeliness capabilities over the air to enable future wireless industrial automation systems," *Proceedings of the IEEE*, vol. 107, no. 6, pp. 1132–1152, 2019.

[6] A. Morato, S. Vitturi, F. Tramarin, C. Zunino, and M. Cheminod, "Time-sensitive networking to improve the performance of distributed functional safety systems implemented over wi-fi," *SENSORS*, vol. 23, no. 18, SEP 2023.

[7] M. Boehm, J. Ohms, M. Kumar, O. Gebauer, and D. Wermser, "Time-sensitive software-defined networking: A unified control- plane for tsn and sdn," in *Mobile Communication - Technologies and Applications; 24. ITG-Symposium*, 2019, pp. 1–6.

[8] T. Kobzan, I. Blöcher, M. Hendel, S. Althoff, A. Gerhard, S. Schriegel, and J. Jasperneite, "Configuration solution for tsn-based industrial networks utilizing sdn and opc ua," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2020, pp. 1629–1636.

[9] L. Leonardi, L. L. Bello, and G. Patti, "Exploiting software-defined networking to improve runtime reconfigurability of tsn-based networks," in *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2022, pp. 1–4.

[10] M. Seliem and D. Pesch, "Software-defined time sensitive networks (sd-tsn) for industrial automation," in *2022 14th International Conference on Computational Intelligence and Communication Networks (CICN)*, 2022, pp. 1–7.

[11] N. S. Bülbül, D. Ergenç, and M. Fischer, "Towards sdn-based dynamic path reconfiguration for time sensitive networking," in *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, 2022, pp. 1–9.

[12] A. Morato, C. Zunino, M. Cheminod, S. Vitturi, D. Cavalcanti, S. Sudhakaran, and F. Tramarin, "Evaluating the integration of wireless time-sensitive networking with software-defined networking for dynamic network configuration," in *2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2023, pp. 1–4.

[13] "Open vSwitch," https://www.openvswitch.org, accessed: 2024-04-12.

[14] A. Morato, C. Zunino, M. Cheminod, S. Vitturi, and F. Tramarin, "A TSN-based Technique for Real-Time Latency Evaluation in Communication Networks," in *2024 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, To be publisehd in 2024.