Time-Sensitive Networking Experimentation on Open Testbeds

Gilson Miranda Jr.*†, Esteban Municio*, Jetmir Haxhibeqiri‡,
Daniel F. Macedo†, Jeroen Hoebeke‡, Ingrid Moerman‡, Johann M. Marquez-Barja*
*IDLab - imec, University of Antwerp, Belgium
‡IDLab - imec, Ghent University, Belgium
†Universidade Federal de Minas Gerais - Computer Science Department, Brazil
{gilson.miranda,esteban.municio,johann.marquez-barja}@uantwerpen.be
{jetmir.haxhibeqiri,jeroen.hoebeke,ingrid.moerman}@ugent.be
damacedo@dcc.ufmg.br

***Abstract*—Time-Sensitive Networking (TSN) is vital to enable time-critical deterministic communication, especially for applications with industrial-grade requirements. IEEE TSN standards are key enablers to provide deterministic and reliable operation of Ethernet networks. However, much of the research is still done in simulated environments or using commercial TSN switches lacking flexibility in terms of hardware and software support. In this work, we evaluate two different Cloud testbeds for TSN experimentation, analyzing their hardware features, the influence of the testbed management infrastructure, and the data plane performance. Furthermore, we present a prototype of a modular Software-Defined Networking (SDN) controller that facilitates the deployment of Linux-based TSN networks. We identify and discuss the controller modules and evaluate its feasibility by using it to deploy TSN networks on different testbeds. Finally, we provide insights for researchers interested in experimenting with TSN features on open Cloud testbeds and discuss the features and limitations that we found during our experiments.**

***Index Terms*—TSN, Experimentation, Network Programmability, SDN**

I. Introduction

In recent years, the IEEE Time-Sensitive Networking (TSN) Task Group has been developing a set of standards to enable reliable and deterministic communication on top of IEEE 802.1 networks [1], [2]. IEEE TSN development evolved from the Audio-Video Bridging (AVB), widening the scope to support the broad requirements of automotive and industrial networks. These standards enable Ethernet-based networks to support the coexistence of time-critical and best-effort flows, providing isolation and preventing best-effort flows from affecting the performance of the time-critical ones. This way, an Ethernet-based TSN network can carry critical traffic, e.g., from control applications in an Industry 4.0 context, while also carrying best-effort traffic from administrative sectors of the industry, reducing costs and deployment complexity [3].

To further evolve TSN standards and effectively implement the proposals on real networks, it is crucial that researchers in industry and academia have access to resources for development and experimentation with TSN features. Although plenty of research advances can be achieved through simulation [4], [5], the implementation and validation using real hardware in realistic conditions is paramount to validate simulation results and push further the technology development.

Fortunately, testbed facilities support researchers by providing very flexible and reconfigurable environments. There are currently several testbeds on different domains, such as Cloud [6], Internet of Things [7], and Smart Highways [8]. These testbeds are excellent means to validate and reproduce the performance of new technologies in realistic conditions. However, for experiments encompassing TSN features, some requirements in terms of hardware support by Network Elements (NEs) and testbed infrastructure are necessary. For example, the network adapter must support hardware timestamping of packets for higher synchronization accuracy with Precision Time Protocol (PTP). Another case is the Time-Aware Scheduling (TAS), which requires network adapters with multiple queues, as well as driver support.

In this paper, we evaluate two Cloud testbeds and their suitability for TSN experimentation. We focus on a basic set of features that offer support for time synchronization, traffic scheduling, and traffic filtering. For resource management and network configuration, we present and develop a prototype of an Software Defined Networking (SDN) controller to perform TSN Centralized Network Configuration (CNC). We identify, describe, and implement the main modules that facilitate the deployment and management of a TSN network. We deploy TSN networks with similar topologies on different testbeds and evaluate their performance in terms of time synchronization accuracy and data plane performance. We also compare the features supported by the testbed nodes and discuss their advantages and limitations for TSN experimentation.

The paper is organized as follows: Section II gives a brief overview of TSN standards. Section III describes the architecture of our TSN controller. Section IV describes the deployment of TSN networks on two different testbeds, detailing the features and issues faced in each case. Section V presents results achieved on each testbed. Section VI brings the main takeaways and lessons learned during the execution of this work, which can help other researchers conducting experimental TSN research using open testbeds. Finally, Section VII concludes this work and presents future directions.

II. Background

The TSN standards can be categorized into four main pillars [9]: time synchronization; bounded low latency; reliability; and resource management. Time synchronization is specified by IEEE 802.1AS standard [10], and can be achieved through PTP. TSN nodes usually perform actions based on strict deadlines or time intervals, and PTP allows all nodes to have a common sense of time. IEEE 802.1Qbv [11] and IEEE 802.1Qbu [12] are the main standards related to bounded low-latency communication. The former leverages network synchronization to coordinate scheduled communications using TAS. The latter defines frame preemption, allowing frames of lower priority flows to be interrupted during transmission in favor of frames of higher priority flows.

For improving reliability the IEEE 802.1Qci standard [13] provides enhancements for flow filtering and policing, allowing traffic to be directed to specific queues on each bridge. This enables precise coordination and isolation of traffic flows. The IEEE 802.1CB [14] also improves reliability through frame replication and elimination, i.e., using redundant transmission of frames through separate paths between source and destination. Finally, for resource management, we highlight IEEE 802.1Qcc [15], which defines protocols for the configuration of flow reservation and their requirements. Three configuration models are defined on 802.1Qcc:

- **Fully distributed:** in this model the flow specifications from the end stations are propagated to the NEs using a distributed protocol. Bridges in the path between sender and receiver are configured based only on their local knowledge.
- **Centralized Network/Distributed User:** this configuration model has a CNC entity, with global knowledge of the network topology and configuration of devices. Bridges on the edges relay flow specifications/requirements received from the end stations to the CNC.
- **Fully centralized:** this model includes a Centralized User Configuration (CUC) entity, aimed at cases in which significant configuration of the end stations is required. Flow characteristics and requirements are specified through the CUC directly to the CNC.

In this work, we will focus on the **fully centralized** model, which gives us more control over all the NEs. Flow characteristics and requirements are provided to the CNC through a CUC API. Synchronization, schedules, and other configurations are specified through the CUC API using auxiliary tools and applied to NEs by the CNC. In the next section, we detail the architecture of the CNC and the agent module.

III. CNC Architecture for TSN

To set up and manage a TSN network, many software components must be carefully configured and coordinated across all NEs. Among them, we highlight the PTP synchronization service, the scheduling configuration, and traffic filtering and policing. However, other functionalities such as real-time data plane and control plane monitoring are also useful. Therefore, we designed a CNC architecture with a minimal set of modules to assist the deployment of TSN networks on Linux-based devices. Our architecture is composed of a central node assuming the roles of CNC and CUC. We simplify the description focusing on the CNC architecture and its counterpart, the TSN Agent (TSNA), that runs in the NEs.

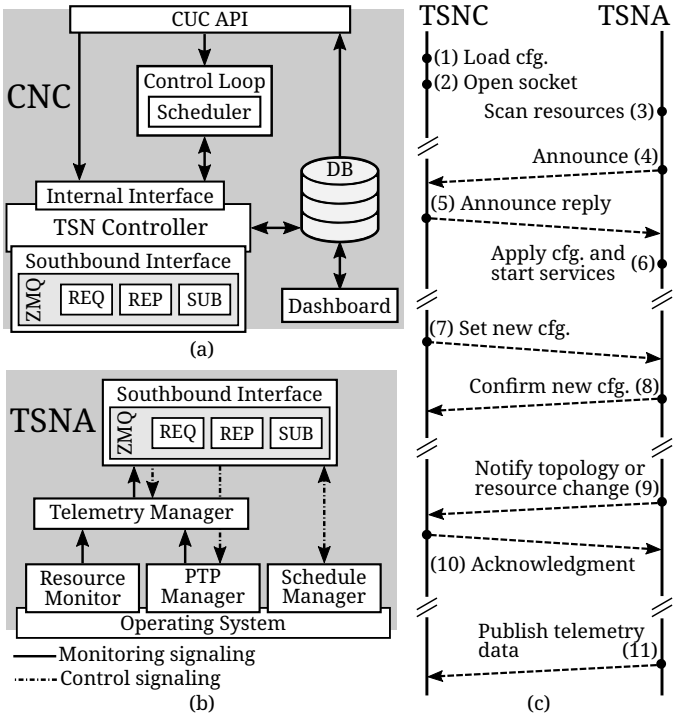

Figure 1: CNC architecture and communications diagram

Figures 1a and 1b show the internal modules of the CNC, and the TSNA, respectively. The main functions of the CNC are carried out by the TSN Controller (TSNC). Figure 1c shows the messages exchanged between the TSNC and TSNA during operation. The TSNC is initialized loading a configuration file with the basic network configuration (1), such as IP addresses, initial schedules, PTP roles and interfaces. Each NE is identified by an Unique Identifier (UID). The TSNC opens a socket (2) and listens for incoming connections of TSNAs started at the NEs. A TSNA initializes scanning the node resources (3), e.g., interfaces, timestamping support, number of queues, and sends an Announce message (4) to the TSNC informing the node UID, node type (end-node or bridge), and its resources. The TSNC replies to the announce (5) by sending the node's configuration specified in the local configuration file, and the TSNA applies the received configuration to the node and starts TSN services such as PTP (6). The TSNC can send messages with new configurations to the TSNA (7), which replies with a confirmation of whether the configuration was applied or an error occurred (8). Similarly, the TSNA can notify topology or resource changes (9), to which the TSNC replies with an acknowledgment (10).

Communication between TSNC and TSNA occurs through the Southbound Interface. For simplicity and faster deployment, we used ZeroMQ (ZMQ) sockets for communication, with messages encoded in JavaScript Object Notation (JSON).

However, the ZMQ sockets can be replaced by NETCONF implementations in future versions, following the current standardization trend for TSN configuration [15]. The Southbound Interface contains three types of ZMQ sockets. A REQ socket is opened for each TSNA that connects to the TSNC, and allows the TSNC to make *requests* to the TSNA, for example, to set a new configuration (messages (7) and (8) of Figure 1c). A global REP socket is used to *reply* to requests coming from TSNAs (messages (9) and (10)). Finally, a SUB socket works as a *subscriber* for telemetry data (11). Telemetry data (e.g., synchronization offset) are transmitted using publish/subscribe mode to the TSNC, and stored in the database.

The Southbound Interface of the TSNA has the counterparts of the three sockets. A REQ socket is used to initiate transmissions to the TSNC (messages (4) and (9) of Figure 1c). The REP socket listens for transmissions coming from the TSNC (message (7)), and the PUB socket for telemetry. We use pub/sub to transmit information that is not crucial for network operation, such as long-term monitoring of synchronization offsets, while topology, link speeds, or address changes are transmitted immediately using the REQ socket.

In the CNC we define the *Control Loop* module, coupled with a *Scheduler*. The control loop verifies if the achieved performance of the flows corresponds to the specifications received from the CUC API. When a new flow is registered through the CUC API, the control loop runs a scheduling algorithm to support the new flow and applies the new network configuration through the TSNC. The information for schedule generation (e.g., network topology, link speeds) is obtained through the TSNC's Internal Interface. Schedules and filtering rules are applied by the control loop on NEs via commands to the Internal Interface of the TSNC.

We define a *Dashboard* module for network monitoring and alarms. Telemetry data published by TSNAs are stored in the database, and the Grafana[1]-based Dashboard module allows the operator to configure screens for statistics visualization and set up alarms to notify about network issues. For example, an operator may set an alarm if PTP reports a synchronization offset higher than a threshold, allowing troubleshooting actions to be taken to avoid further issues.

For network operation, the CNC offers the *CUC API* as a centralized management interface. NE configurations can be set based on their UIDs. Flow characteristics and performance requirements are also specified using this API, realizing the fully centralized configuration model. Access permissions are stored in the database and the CUC API enforces them when a request is issued either to the control loop or directly to the TSNC (via Internal Interface).

On the TSNA side, the *Schedule Manager* applies the schedules received from the TSNC, and it also filters policies to direct traffic to the correct queues. This is performed using the Traffic Control (TC) tool for Linux, specifically the tc-taprio[2] module that implements the Time Aware Priority Shaper (TAPRIO). The basic specification of a schedule contains a Gate Control List (GCL), a base-time that indicates when the schedule starts, and a set of schedule entries that indicate which gates (or queues) will be active (or transmitting) at a given moment of a cycle. The example below shows a configuration with a base-time and three schedule entries. The first entry allows traffic mapped to queue 1 to be transmitted in the first $200\mu s$ of the cycle, the second entry gives $100\mu s$ for traffic in queue 3, and the third entry gives $200\mu s$ for traffic in queues 1 and 2 (gate mask 0x03).

```
base-time 1528743495910289987
sched-entry S 0x01 200000
sched-entry S 0x04 100000
sched-entry S 0x03 200002
```

The *PTP Manager* controls the synchronization service, based on *linuxptp*[3], according to the configuration parameters received from the TSNC. A synchronization offset between the NE and the PTP GrandMaster (GM) is reported by *linuxptp* and collected by the PTP Manager. This information is delivered to the *Telemetry Manager*, which may publish to the TSNC if configured to do so. The *Telemetry Manager* can be configured to publish aggregated statistics of this offset (average over several minutes) or only when the offset is above a threshold, to reduce monitoring traffic. Lastly, the *Resource Monitor* constantly checks the state of interfaces, their speed, and the status of *linuxptp* processes, immediately informing the TSNC if any changes occur on these elements. The objective is to quickly inform the TSNC about topology changes, or failure of crucial processes, so recovery measures can be taken.

IV. Deployment of TSN Networks

TSN networks require specialized hardware and software support for some functionalities. For the scope of the experiments in this paper, the main requirements are hardware-level packet timestamping for PTP synchronization and multiple transmission queues for TAPRIO. It is possible to configure nodes without hardware timestamping, however, PTP synchronization will use software timestamping, taking longer to converge and usually presenting higher synchronization errors. The multi-queue and timestamping support are required not only in the hardware level but also in software.

We set up two distinct experiments on two testbeds: Virtual Wall 2 [16], and CloudLab Utah [17]; using topologies as similar as possible, in order to compare the characteristics of both testbeds. We configured all nodes on both experiments with Ubuntu 20.04.3 LTS, Linux kernel v5.4.0-91, iproute2 v5.5.0, and linuxptp v3.1.1. On Virtual Wall we used the *pcgen03-5p* nodes equipped with Intel(R) Xeon(R) CPU E5645 with 24 threads and 24GB of RAM. On CloudLab we used the *d6515* nodes equipped with AMD EPYC 7452 CPU with 64 threads and 128GB of RAM. The nodes had different configurations of network adapters, with CloudLab nodes having three different models on each selected node.

[1]https://grafana.com/

[2]https://man7.org/linux/man-pages/man8/tc-taprio.8.html

[3]http://linuxptp.sourceforge.net/

Table I lists the network adapters on each testbed. The first column shows interface names returned by the Operating System (OS), the second column indicates the support for hardware timestamping. The third and fourth columns show the number of transmission (TX) and reception (RX) queues, reported by the *ethtool* application. The fifth column shows the maximum link speeds of each card, and the last column shows the controller model.

Table I: Network adapters on testbed nodes selected

Interface	HW TS	TXQ	RXQ	Speed	Controller
Virtual Wall					
eth*	yes	8	8	1Gb/s	i82576
Cloudlab					
ens1f*np*	no[1]	74	74	25Gb/s	BCM57414
eno*	yes	4	4	1Gb/s	BCM95720
ens3f*	yes	126[2]	63[2]	100Gb/s	MT28800

[1]Limited support due to driver version.
[2]Queue numbers depend on the number of CPU logical cores.

All Virtual Wall nodes that we selected have two Intel 82576 network adapters, one with two ports and the other with four ports. One port is allocated for testbed management plane, leaving the other 5 available for connection between nodes. The interfaces have hardware timestamp support, 8 TX and RX queues, and 1Gbps line speed. The nodes selected on CloudLab have three different adapters, each with two ports. The interfaces listed as *ens1f0np0* and *ens1f1np1* use the Broadcom BCM57414, which offer line speeds of 25Gbps, and report 74 TX and RX queues. Although the network adapter documentation indicates support for hardware timestamping and IEEE 802.1AS synchronization, *ethtool* and *linuxptp* were not able to identify such support, limiting the capabilities of those interfaces for PTP synchronization. The other two interfaces, with naming pattern *eno**, used Broadcom BCM95720 controllers. These interfaces had a maximum speed of 1Gbps, and hardware timestamping support. The controller documentation indicates support for 17 RX queues and 16 TX queues, but from the OS, we were able to enable at most 4 RX and TX queues. The last row of the table details the *ens3f0* and *ens3f1* interfaces, which use the Mellanox MT28800 controller. These interfaces have speeds of 100Gbps, hardware timestamping support, and a number of queues according to the number of logical CPU cores. As we used nodes with 64 cores, the OS reported 63 TX and 126 RX queues for those interfaces.

The topology used for the tests is shown in Figure 2. In CloudLab, despite having 6 network ports (1 reserved for testbed infrastructure), we were unable to create more than 3 links per node. Therefore, for the CloudLab setup we used a virtual shared link to connect PC1, PC3, and SW1, and another virtual shared link to connect PC2, PC4, and SW3. In those cases, the link of a single port is split into more links to the connected nodes. In Virtual Wall we were able to use all the network ports of the nodes. Figure 2 also indicates the traffic generated during the scheduling tests. We generate UDP traffic with two test applications from PC1 to PC2 (UDP App 1 and UDP App 2) and use *iperf3*[4] to generate TCP traffic from PC3 to PC4. In both experiments, we configured SW1 as PTP GM, SW2 and SW3 as PTP Boundary Clock (BC), and the PCs as PTP slaves. The CNC was deployed in a separate node, with a logical control link to all other nodes.

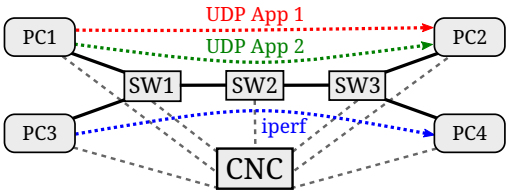

Figure 2: Topology used for the experiments

V. Experimental Results

We deployed the two experiments on Virtual Wall and CloudLab testbeds, seeking to achieve topology and node configurations as similar as possible. We evaluated the functionalities of precise time synchronization and traffic scheduling running similar tests in both experiments.

A. PTP Synchronization Accuracy

Our first analysis regards the PTP synchronization accuracy over 30 minutes of execution. We disregard the first two minutes after initializing PTP to allow all nodes to reach a steady-state of synchronization. Table II shows the absolute PTP synchronization offset (error) between each node and the GM, in each testbed. For each case, we show the median offset, the 90th, and 99.9th percentiles. We observed better synchronization accuracy on CloudLab nodes, with median offset up to 26 ns, while on Virtual Wall nodes the median offset was up to 416 ns. The testbed infrastructure contains switches or routing equipment to interconnect nodes. Such equipment may introduce delays and jitter in the traffic between the nodes of the experiment, affecting the PTP operation. Nevertheless, both testbeds offer sub-microsecond synchronization accuracy most of the time, allowing reliable experimentation with schedules having slots in the range of tens of microseconds.

Table II: Absolute PTP synchronization offset (in ns)

	Median	90th	99.9th	Median	90th	99.9th
Node	**Virtual Wall**			**CloudLab**		
SW2	218	572	1061	12	30	604
SW3	305	738	1535	16	39	935
PC1	192	478	1020	16	44	547
PC2	394	984	1795	26	68	1352
PC3	220	520	1069	12	30	562
PC4	416	998	5624	23	58	1398

B. Traffic Scheduling

The scheduling experiments demonstrate the use of multi-queues and traffic policing to perform fine-grained control over traffic behavior. We defined the three schedules shown

[4]https://iperf.fr/

in Figure 3 for these experiments. Each square represents a slot of 250μs. The number in each square indicates the active queue. We also defined slots with all queues inactive (crossed squares), which can represent slots allocated to other traffic classes. Best-effort traffic is directed to queue 0, including communication between TSNC and TSNAs, and PTP packets. Iperf traffic was allocated to queue 1, while UDP Application 1 was allocated to queue 2, and UDP Application 2 allocated to queue 3. The schedules were applied through commands to the CUC API on egress ports of nodes PC1, SW1, SW2, and SW3, considering the direction of traffic shown in Figure 2.

Figure 4 shows the results obtained on Virtual Wall. The top graph shows the 99th percentile of one-way delay of packets from the UDP Apps. The UDP sender transmits a packet every 1 ms. The bottom graph shows the TCP throughput achieved with iperf3. Schedule 1 allocates one slot for each flow, resulting in 5.75 ms delay for the UDP Apps and 5.5 MB/s throughput for iperf. Schedule 2 allocates another slot for iperf, doubling the throughput to 11 MB/s. Schedule 3 replaces the slot from queue 1 with a slot for queue 2, giving the additional slot to App 1. We can observe the simultaneous change of behavior in both flows, reducing the delay of App 1 to 3.5 ms, and reducing iperf throughput to the initial 5.5 MB/s. Lastly, we apply Schedule 1 again and observe the same behavior from the start of the experiment.

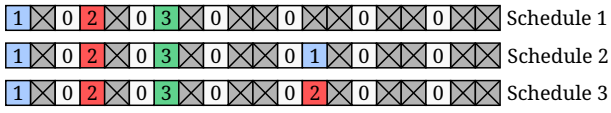

Figure 3: Schedules used for traffic control experiments

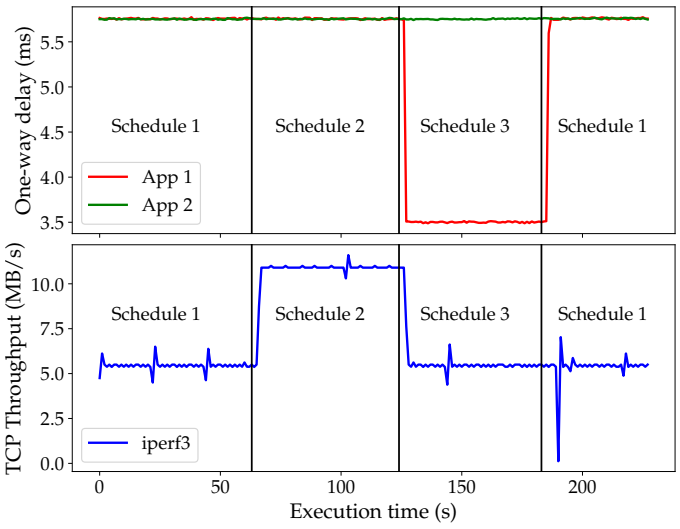

Figure 4: Scheduling on Virtual Wall testbed

Figure 5 shows the results of the same experiment on CloudLab. Due to the higher speed of network adapters, we observed a significantly higher throughput using iperf, starting at around 80 MB/s and reaching up to 175 MB/s when applying Schedule 2. We also see a slight decrease of one-way delay of UDP Apps as the propagation of packets through the links can occur much faster. Despite that, we observed a few peaks of delay in this deployment. We suspected that these peaks were caused by the shared link created by the testbed to interconnect SW3, PC2, and PC4. The egress port of SW3 (on which we apply the schedules) is logically split into two links to connect PC2 and PC4. After the packets egress from SW3 port, packet queuing or aggregation policies are not under our control, and the flows might be affected by the testbed configuration. We performed an additional experiment where flows do not converge to a common link. We used SW3 as end node, taking the roles of PC2 and PC4 on receiving traffic from PC1 and PC3. The result of this test is shown in Figure 6, in which we omit the iperf results due to space limitation. We observed that the delay spikes did not occur and one-way delay was more deterministic.

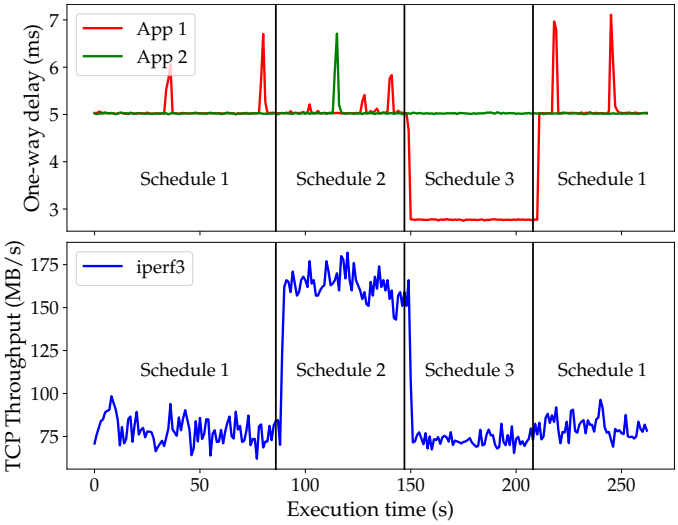

Figure 5: Scheduling on CloudLab testbed

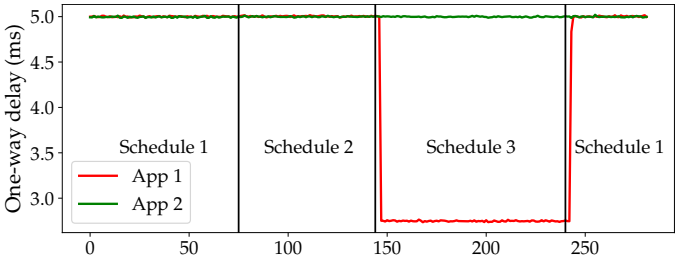

Figure 6: Scheduling on CloudLab testbed ending on SW3

VI. Main Takeaways and Lessons Learned

Experimental research with TSN features requires specialized support in hardware and software. Fortunately, different testbeds offer support for such research, allowing experimental validation of theoretical works. We list below a few takeaways and lessons learned during this work that may help other researchers when setting up their experiments:

- **Preliminary feature test:** before setting up experiments with several nodes, it is important to verify hardware and

software support for the desired features. As we observed, it may occur that a feature is supported by hardware but not fully supported in software. Based on preliminary tests, the user can better plan the experiments or necessary software upgrades before setting up larger deployments.

- **Synchronization:** the supporting infrastructure of the testbeds may affect synchronization accuracy. During initial tests in Virtual Wall, we configured the Controller as PTP GM, and observed higher synchronization offset on some nodes. Changing the GM to SW1 offered better overall synchronization.
- **Effect of shared logical links:** testbeds allow creating logical links that share a single port to overcome the limitation of physical ports and offer more flexibility on network topology. However, the user might have no control over such links (e.g., over frame aggregation or queuing policy). When running experiments with TSN scheduled traffic, the use of such shared links might influence the intended deterministic behavior of flows.

VII. Conclusion

TSN standards are enablers for achieving reliable, deterministic, and bounded low-latency communications over Ethernet. In this paper, we demonstrate the use of Virtual Wall and CloudLab tesbeds for TSN experimentation. We introduce an architecture for a TSN controller composed of a CNC and a TSN Agent that helps with the fast deployment of TSN networks. We present a qualitative analysis of the features in both testbeds for TSN experimentation, and also show qualitative results of basic TSN features – synchronization and traffic scheduling. We show how the TSN standards can be used to perform precise control over traffic behavior, as well as performance isolation between different flows. We summarize the main takeaways and lessons learned during the development of this work, in order to help other users, and conclude that both testbeds offer a valuable set of features and performance for TSN research.

Acknowledgment

This research was funded by the Flemish FWO SBO #S003921N VERI-END.com (Verifiable and elastic end-to-end communication infrastructures for private professional environments) project, the Flemish Government under the "Onderzoeksprogramma Artificiele Intelligentie (AI) Vlaanderen" program, and from the FWO-Flanders (Grant agreement No. G055619N). This research has also been supported by the Horizon 2020 Fed4FIRE+ project (Grant Agreement No. 723638).

References

[1] J. L. Messenger, "Time-Sensitive Networking: An Introduction," *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 29–33, jun 2018. [Online]. Available: doi.org/10.1109/mcomstd.2018.1700047

[2] "Time-Sensitive Networking: A Technical Introduction," *Cisco Public White Paper*, 2017. [Online]. Available: www.cisco.com/c/dam/en/us/solutions/collateral/industry-solutions/white-paper-c11-738950.pdf

[3] N. Finn, "Introduction to Time-Sensitive Networking," *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 22–28, 2018. [Online]. Available: doi.org/10.1109/MCOMSTD.2018.1700076

[4] A. C. T. d. Santos, B. Schneider, and V. Nigam, "TSNSCHED: Automated Schedule Generation for Time Sensitive Networking," in *2019 Formal Methods in Computer Aided Design (FMCAD)*, 2019, pp. 69–77. [Online]. Available: doi.org/10.23919/FMCAD.2019.8894249

[5] J. Falk, F. Durr, and K. Rothermel, "Exploring Practical Limitations of Joint Routing and Scheduling for TSN with ILP," in *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, aug 2018, pp. 136–146. [Online]. Available: doi.org/10.1109/RTCSA.2018.00025

[6] J. Mambretti, J. Chen, and F. Yeh, "Next generation clouds, the Chameleon cloud testbed, and Software Defined Networking (SDN)," in *International Conference on Cloud Computing Research and Innovation (ICCCRI)*. IEEE, 2015. [Online]. Available: doi.org/10.1109/ICCCRI.2015.10

[7] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele *et al.*, "FIT IoT-LAB: A large scale open experimental IoT testbed," in *2nd World Forum on Internet of Things (WF-IoT)*. IEEE, 2015. [Online]. Available: doi.org/10.1109/WF-IoT.2015.7389098

[8] J. Marquez-Barja, B. Lannoo, D. Naudts, B. Braem, V. Maglogiannis, C. Donato, S. Mercelis, R. Berkvens, P. Hellinckx *et al.*, "Smart Highway: ITS-G5 and C2VX based testbed for vehicular communications in real environments enhanced by edge/cloud technologies," in *EuCNC, European Conference on Networks and Communications*, 2019. [Online]. Available: biblio.ugent.be/publication/8642435/file/8656511

[9] 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. [Online]. Available: doi.org/10.1109/JPROC.2019.2905334

[10] "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. [Online]. Available: doi.org/10.1109/IEEESTD.2020.9121845

[11] "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. [Online]. Available: doi.org/10.1109/IEEESTD.2016.8613095

[12] "IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks – Amendment 26: Frame Preemption," *IEEE Std 802.1Qbu-2016 (Amendment to IEEE Std 802.1Q-2014)*, pp. 1–52, 2016. [Online]. Available: doi.org/10.1109/IEEESTD.2016.7553415

[13] "IEEE Standard for Local and metropolitan area networks–Bridges and Bridged Networks–Amendment 28: Per-Stream Filtering and Policing," *IEEE Std 802.1Qci-2017 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, IEEE Std 802.1Q-2014/Cor 1-2015, IEEE Std 802.1Qbv-2015, IEEE Std 802.1Qbu-2016, and IEEE Std 802.1Qbz-2016)*, pp. 1–65, 2017. [Online]. Available: doi.org/10.1109/IEEESTD.2017.8064221

[14] "IEEE Standard for Local and metropolitan area networks–Frame Replication and Elimination for Reliability," *IEEE Std 802.1CB-2017*, pp. 1–102, 2017. [Online]. Available: doi.org/10.1109/IEEESTD.2017.8091139

[15] "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. [Online]. Available: doi.org/10.1109/IEEESTD.2018.8514112

[16] "Virtual Wall: imec iLab.t documentation." [Online]. Available: https://doc.ilabt.imec.be/ilabt/virtualwall/index.html

[17] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, J. Duerig, E. Eide, L. Stoller, M. Hibler, D. Johnson, K. Webb, A. Akella, K. Wang, G. Ricart, L. Landweber, C. Elliott, M. Zink, E. Cecchet, S. Kar, and P. Mishra, "The Design and Operation of CloudLab," in *Proceedings of the USENIX Annual Technical Conference (ATC)*, jul 2019, pp. 1–14. [Online]. Available: https://www.flux.utah.edu/paper/duplyakin-atc19