Evaluating Time-Sensitive Networking Features 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 on top of Ethernet networks. 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 demonstration, we use an open Cloud testbed for TSN experimentation, leveraging the hardware features that support precise time synchronization, and fine-grained scheduling according to TSN standards. We demonstrate the setup and operation of a Linux-based TSN network in the testbed using our modular Centralized Network Configuration (CNC) controller prototype. With our CNC we are able to quickly initialize the TSN bridges and end nodes, as well as manage their configurations, modify schedules, and visualize overall network operation in real-time. The results show how the TSN features can be effectively used for traffic management and resource isolation.**

***Index Terms*—TSN, Experimentation, 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]. These standards enable Ethernet-based networks to provide flow isolation, allowing the co-existence of time-critical and best-effort flows, but avoiding that the best-effort traffic interfere on the QoS of the time-critical traffic. Until now plenty of TSN research is done in simulated environments. However, for further development in this field, it is of utmost importance that researchers can test their solutions in real environments using open testbeds.

In this demo we show how open Cloud testbeds can be used for TSN experimentation by utilizing different hardware features providing end-to-end time synchronization through Precision Time Protocol (PTP), traffic classification, and scheduling. In addition to TSN hardware features, means for network management and control need to be provided as well. We demonstrate that our Centralized Network Configuration (CNC) controller prototype is capable of setting up a TSN network in the testbed, managing the fundamental TSN functions for Network Elements (NEs) initialization, time synchronization, end-to-end scheduling, and performance monitoring.

II. TSN and CNC Architecture

Besides time synchronization, traffic scheduling and policing and reliability features, IEEE 802.1 TSN standards cover the resource management as well [3]. Network resource management can be done in fully distributed model, centralized model, or the centralized network/distributed user model. Our CNC controller allows us to manage TSN features on Linux-based NEs. Figure 1a gives an overview of the CNC modules, while Figure 1b shows the architecture of the agent deployed on NEs. Figure 1c shows the message flow diagram between controller and agents.

The main component of the CNC is the TSN Controller (TSNC), which communicates with the TSN Agents (TSNAs) through the Southbound Interface. The TSNC provides an Internal Interface, which allows users (through the Centralized User Configuration (CUC) API) and other modules (e.g. Control Loop) to send commands and receive information from NEs. The TSNC also receives telemetry data in a publish/subscriber mode, and stores long-term telemetry in a database. Data visualization is done with a Dashboard module.

In the NEs, the TSNA interacts with the CNC through the *Southbound Interface*. A *Telemetry Manager* controls which telemetry statistics are reported to the CNC. State of interfaces and important processes (e.g. *linuxptp*) are monitored by the *Resource Monitor*, and relevant changes are notified to the CNC. The *PTP Manager* controls the synchronization service, while the *Schedule Manager* applies Time-Aware Scheduling (TAS) rules and traffic filtering policies. Each NE is identified by an Unique Identifier (UID), defined on CNC configuration, and announced during TSNA connection.

III. Demonstration Scenario

Figure 2 shows the topology used in the demonstration, as well as the traffic flows generated. The CNC is logically connected to all nodes. PC1 transmits two UDP flows (UDP Apps 1 and 2, at 1000 packets/s each) to PC2, and PC3 generates TCP traffic to PC4 using *iperf*. The three schedules

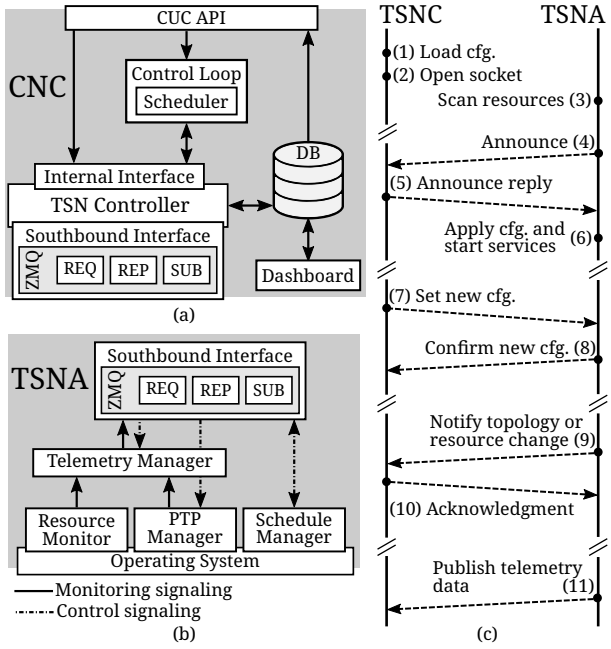

Figure 1: CNC architecture and communications diagram

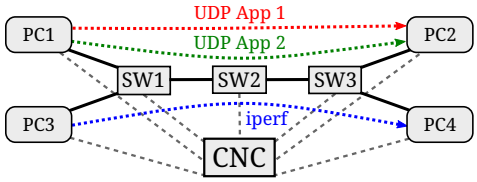

Figure 2: Topology used for the experiments

shown in Figure 3 are applied during the experiment. Each square represent a slot of 250μs. Slots with a number indicate which queue is served at a given moment. We allocate queue 0 for best-effort traffic (e.g. PTP and telemetry traffic), queue 1 for iperf, queue 2 for the UDP App 1, and queue 3 for UDP App 2. Queues crossed represent unallocated slots, and no transmissions take place. The nodes publish statistics of 99th percentile of one-way delay of the UDP Apps, and iperf throughput to the CNC for real-time visualization.

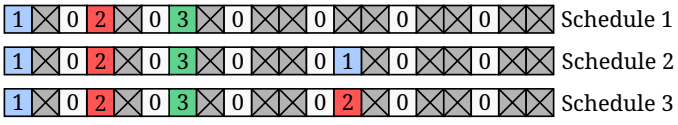

Figure 3: Schedules used for traffic control experiments

IV. Demonstration Output

Table I shows the synchronization accuracy achieved by each node, with SW1 running as PTP GrandMaster (GM), SW2 and SW3 running as PTP Boundary Clocks (BCs), and the PCs running as PTP slaves. During most of the time all nodes achieve sub-microsecond synchronization, a sufficient margin for our schedules using 250μs slots.

Figure 4 shows the scheduling results. Schedule 1 assigns a single slot to each flow. Schedule 2 adds one more slots

Table I: Synchronization offset in nanoseconds from GM

Node	SW2	SW3	PC1	PC2	PC3	PC4
Median	218	305	192	394	220	416
90th	572	738	478	984	520	998
99.9th	1061	1535	1020	1795	1069	5624

to iperf flow, doubling its throughput. This change does not affect the delay of UDP Apps. Schedule 3 assigns the slot to UDP App 1, reducing its one-way delay, and returning iperf throughput to 5.5MB/s. Finally, Schedule 1 is applied again and all flows return to their initial behavior. UDP App 2 is unaffected during the experiment.

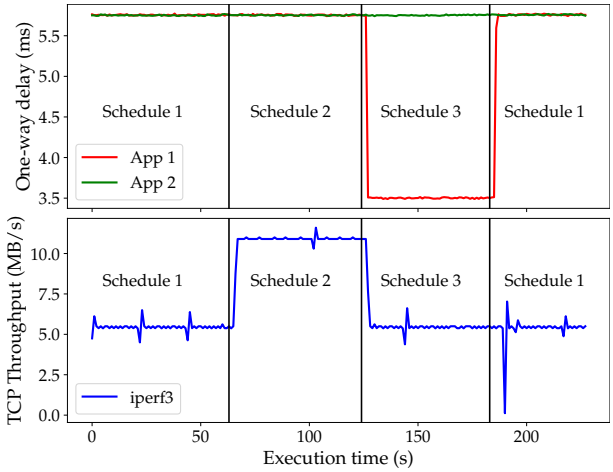

Figure 4: Scheduling on Virtual Wall testbed

V. Conclusion

In this paper we demonstrate the utilization of TSN features on nodes of the Virtual Wall testbed, using our CNC controller prototype. Our results show how fine-grained control and flow isolation can be achieved using the TSN features, and the suitability of exiting open testbeds for TSN experimentation.

Acknowledgment

This research was partially 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] 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