V-TSN: A Software-Defined TSN Overlay for General-Purpose Networks

Mohammadparsa Karimi[1], Majid Nabi[1], Ahmed Khalaf[2], Andrew Nelson[1], Kees Goossens[1], Twan Basten[1]
[1]Department of Electrical Engineering, Eindhoven University of Technology, The Netherlands
Emails: {m.karimi, m.nabi, a.t.nelson, k.g.w.goossens, a.a.basten}@tue.nl
[2]Architecture and Network Solutions, AUMOVIO, Germany
Email: ahmed.khalaf@aumovio.com

***Abstract*—Time-Sensitive Networking (TSN) extends Ethernet with deterministic communication for time-critical applications such as industrial automation, in-vehicle networks, and cyber-physical systems. However, realizing TSN behavior without dedicated hardware is difficult. During design and validation, offline simulation cannot run application software at real-time speed when costly specialized TSN hardware is not (yet) available. At deployment time, many systems run on general-purpose and cloud networks with no native TSN support, where provisioning full TSN hardware is unnecessary or impractical for applications that tolerate relaxed timing. In this paper, we introduce Virtual Time-Sensitive Networking (V-TSN), a software-defined overlay that realizes gPTP-based synchronization and TSN traffic shaping over general-purpose, non-deterministic networks without specialized hardware. V-TSN runs in real time alongside the unmodified application stack, serving both as a development-time emulation tool and as a cost-efficient deployment option where relaxed timing is acceptable. In a cloud-based deployment, V-TSN achieves an average clock offset below 200 microseconds, it isolates time-critical traffic through a virtual Time-Aware Shaper (TAS), and it enforces per-class bandwidth reservations through a virtual Credit-Based Shaper (CBS).**

***Index Terms*—Time-Sensitive Networking, Virtual TSN, Network Function Virtualization, Software-Defined Networking**

I. Introduction

Time-Sensitive Networking (TSN) [1] is a set of IEEE 802.1 standards that extend Ethernet with deterministic communication, enabling time synchronization, bounded latency, low jitter, and high reliability for time-critical distributed systems. It integrates functionalities such as precise time synchronization [2], Time-Aware shaper (TAS) [3], Credit-Based Shaper (CBS) [4], and frame preemption [5] to enable predictable data exchange over standard Ethernet. TSN is a key enabling technology across domains such as industrial, in-vehicle, aerospace, and robotic, where distributed components must operate in tight coordination based on a shared notion of time to ensure correctness, safety, and reliability [6].

Modern distributed systems are increasingly deployed on commodity and virtualized platforms, where applications run as virtual machines or containers on nodes ranging from edge platforms to on-chip systems. These nodes are typically interconnected through general-purpose networks such as the Internet or cloud infrastructures, which provide neither deterministic communication nor native TSN capabilities [7]. In such environments, clocks are at best loosely synchronized through software protocols such as NTP, or not synchronized at all, and there is no synchronized TSN protocol stack, no deterministic forwarding, and no time-aware traffic shaping. Achieving genuine TSN determinism and TSN-grade synchronization on these platforms requires specialized hardware, such as TSN-capable network interfaces and switches with hardware timestamping, which significantly increases system cost and deployment effort [8]. This level of precision, however, is not always necessary; in many practical scenarios, moderate timing accuracy is sufficient for correct system behavior [9].

Beyond cost, faithfully reproducing TSN behavior during development and validation is itself challenging, particularly because in early design stages the target TSN hardware is often not yet available. Thus, validation must be carried out on virtual platforms rather than on the final infrastructure. This is especially evident in domains such as automotive [10] and aerospace [11], where extensive validation is performed in virtual environments before integration into physical systems and where timing behavior is critical for correct and safe operation. Simulation-based tools [12], [19] support offline design and analysis of TSN systems, but they run in simulated time, far slower than real time, and cannot execute the actual application software. They therefore cannot reproduce the runtime behavior of a complete system over real distributed infrastructure, which is what is needed to validate timing interactions and functional correctness prior to deployment.

To address the cost and flexibility limitations of hardware-based network infrastructures, the networking community is increasingly adopting paradigms such as Network Function Virtualization (NFV) and Software-Defined Networking (SDN) [13]. NFV enables the implementation of network functions as software-based Virtual Network Functions (VNFs), allowing them to run on commodity computing platforms instead of dedicated hardware appliances. Similarly, SDN decouples the control and data planes, enabling centralized and programmable network management. These approaches improve flexibility, scalability, and resource utilization, while also reducing deployment and operational costs [14]. A wide range of network functions, including routing [15], firewalling [16], load balancing, and traffic monitoring [17], have been successfully virtualized using these paradigms [18]. However, only a limited number of studies, reviewed in Section II, have investigated the virtualization of TSN functionalities.

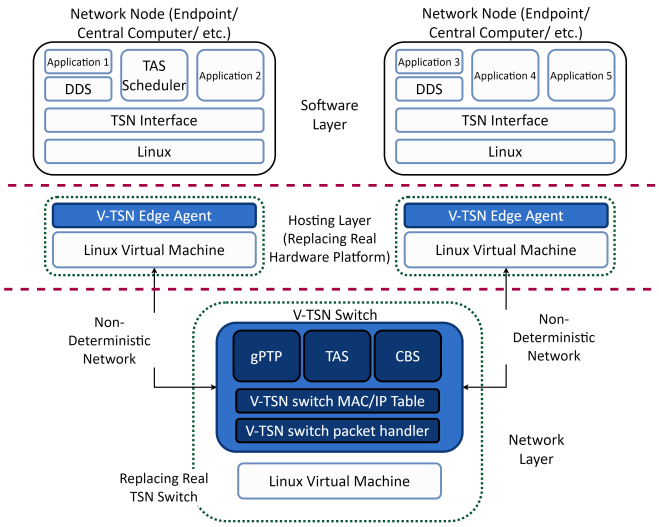


Fig. 1: The V-TSN overlay. Each endpoint and the switch run as software components on Linux virtual machines, interconnected over a general-purpose, non-deterministic network. The V-TSN switch replaces a physical TSN switch and implements gPTP synchronization, TAS, and CBS in software.

This paper introduces V-TSN, a software-defined overlay that realizes core TSN functionalities, namely gPTP-based synchronization, TAS, and CBS, over general-purpose, non-deterministic networks without requiring specialized hardware. Unlike simulation, V-TSN runs in real time together with the unmodified application stack, and, unlike hardware TSN, it targets relaxed sub-millisecond timing precision rather than nanosecond-level guarantees. V-TSN supports two use cases:

- **UC1, development-time emulation:** V-TSN reproduces TSN behavior so that a system can be validated before integration on physical TSN infrastructure, where we assume that the virtualized system runs at *implementation speed*, so that its timing reflects the eventual hardware deployment.
- **UC2, cost-efficient deployment:** Where full TSN hardware is unnecessary and the application tolerates relaxed timing, V-TSN provides TSN-related timing behavior directly over general-purpose networks, for example in distributed sensor data collection, multimedia streaming, and coordination between edge and cloud.

Fig. 1 shows the V-TSN overlay, in which endpoints and the switch run as standard computing nodes, such as cloud-based virtual machines, interconnected over a general-purpose network. We demonstrate the feasibility of the proposed approach through a real-world cloud-based deployment, evaluating synchronization accuracy, traffic isolation through virtual TAS, and bandwidth regulation through virtual CBS. The software described in this paper is available via the TU/e ES GitHub repository (https://github.com/TUE-EE-ES/V-TSN).

The remainder of this paper is organized as follows. Section II reviews related work. Section III presents the design of the proposed V-TSN platform. Section IV evaluates the proposed approach. Finally, Section V concludes.

II. Related Work

Realizing TSN behavior over general-purpose networks requires both time synchronization and traffic shaping, since time-aware shaping is meaningful only when nodes are synchronized. Prior work addresses parts of this problem through TSN simulation, hardware testbeds, endpoint synchronization, and, more recently, the virtualization of TSN over commodity platforms, which we review and contrast with V-TSN.

TSN simulation frameworks such as OMNeT++ [19] and INSIM [12] enable offline evaluation of scheduling strategies and protocol behavior. However, they run in simulated time, far slower than real time, and cannot connect real applications, so they are unsuitable for development workflows that require interaction with actual software.

Hardware testbeds such as [20] and EnGINE [21] combine Commercial Off-The-Shelf Ethernet devices with orchestration frameworks for high-fidelity measurements. They depend on physical TSN-capable infrastructure and fixed setups, which we avoid in this work.

Protocols such as Linux PTP [22] and PTPd [23] assume hardware support for timestamping and synchronize network endpoints only. In contrast, V-TSN implements gPTP, which is part of the TSN standard, entirely in software. It synchronizes endpoints as well as intermediate virtualized TSN switches. The latter also implement TAS and CBS traffic shaping.

Closest to our work is a recent research line that virtualizes TSN over commodity platforms. Sasiain et al. [24] propose to integrate TSN into the ETSI NFV framework, but their proposal is unimplemented, limited to TAS, and assumes TSN hardware. Zhang et al. [25] treat virtualized TSN as a control-plane problem of joint admission control and VNF embedding, evaluated in simulation rather than as a running overlay. Most similar, KuberneTSN [26] provides a deterministic container overlay with a userspace implementation of TAS and kernel-bypass acceleration for Kubernetes environments. However, KuberneTSN virtualizes packet scheduling at the container host and relies on an existing TSN infrastructure for synchronization and switching. Furthermore, it is evaluated on a controlled local testbed with externally synchronized hosts and a physical TSN-compliant switch. Instead, V-TSN virtualizes the complete TSN communication substrate entirely in software. The proposed platform virtualizes both the endpoints and the switch and it operates over non-deterministic general-purpose networks without TSN hardware. Table I summarizes the differences between existing approaches and V-TSN.

III. V-TSN Platform Design

In V-TSN, both the endpoints and the switches are software components that can run on general-purpose hosts and be placed anywhere in the network. A virtual switch is just another software component, a virtual machine (VM) that may even run on an endpoint. To the underlying network, it carries only ordinary traffic that standard network switches forward, like any other switch. Lightweight software agents run at the endpoints, while the core networking logic resides in the

TABLE I: Comparing TSN virtualization approaches

Feature	Sasiain [24]	Zhang [25]	KuberneTSN [26]	V-TSN (Ours)
Prototype implementation	✗	✗	✓	✓
gPTP virtualization	✗	✗	✗	✓
Virtual TSN switch	✗	✗	✗	✓
TAS	Partial	✗	✓	✓
CBS	✗	✗	✗	✓
No TSN hardware needed	✗	N/A	✗[1]	✓
WAN deployment	✗	✗	✗	✓

[1] KuberneTSN runs on commodity hosts, but its determinism is demonstrated on a testbed with TSN-capable NICs and a physical TSN switch.

virtual V-TSN switch, on top of which the TSN functionalities are layered.

The following subsections describe, first, the virtual packet switching that carries all traffic, then the software realization of gPTP for synchronization, and finally the TAS and CBS traffic shaping implemented within the V-TSN switch.

A. Virtual Packet Switching Mechanism

The basis of the V-TSN platform is a virtual packet switching mechanism that enables communication between distributed endpoints over a general-purpose network. Since the underlying network does not provide native TSN support, the proposed approach introduces an overlay communication layer in which Ethernet frames are encapsulated, transmitted, and processed through software-defined components at the endpoints and the virtual V-TSN switches.

Fig. 2 illustrates this packet processing pipeline. At the talker, a frame encapsulation module wraps each outgoing Ethernet frame in a UDP/IP packet addressed to the V-TSN switch, enabling transport over standard IP networks. At the switch, (i) incoming packets are decapsulated, (ii) a virtual destination resolution module selects the output endpoint from a mapping table of logical endpoints to IP addresses and ports, (iii) the frame is optionally processed by the traffic shaping mechanisms, and (iv) it is then re-encapsulated and forwarded. At the listener, a decapsulation module recovers the original Ethernet frame and an application mapping mechanism delivers it to the correct endpoint component using header fields such as ports. This effectively emulates a layer-2 switch over a layer-3 network while integrating TSN traffic control.

B. Virtual gPTP Synchronization

Time synchronization in TSN is provided by gPTP, which aligns the local clocks of distributed nodes with a grandmaster reference clock. Unlike end-to-end protocols, gPTP is a hop-to-hop protocol: it measures link delay on each hop through a peer-delay exchange (*Pdelay_Req*/*Pdelay_Resp*) and distributes the reference time through *Sync* and *Follow_Up* messages, which yields higher synchronization accuracy.

In the V-TSN platform, gPTP is implemented by handlers that run as ordinary applications on the endpoints and the V-TSN switch, as shown in Fig. 3. One Linux VM acts as the grandmaster node. Because gPTP uses normal Ethernet packets, the exchanges are encapsulated and transported over the virtualized TSN network as described above like regular data traffic, avoiding a separate synchronization channel and preserving the structure of gPTP communication over a non-deterministic network.

C. Virtual Traffic Shaping

Traffic shaping in TSN controls the timing and rate of packet transmissions to achieve predictable communication behavior. In the V-TSN platform, traffic shaping is implemented within the V-TSN switch, providing a software-based realization of TSN scheduling over a non-deterministic network.

Fig. 4 illustrates the architecture of the virtual traffic shaping module. At the egress of the shaping pipeline, Ethernet frames are first classified by their Priority Code Point (PCP) values and mapped to one of eight virtual queues corresponding to the standard priority levels (PCP 0 to PCP 7), representing the logical separation of traffic classes in TSN.

Two shaping mechanisms are applied within the switch: CBS and TAS. CBS is implemented through a set of virtual CBS modules. A CBS controller regulates the transmission rate of each queue by maintaining and updating credit values, ensuring bandwidth sharing and smooth traffic behavior. After CBS, the TAS mechanism is realized using a virtual Gate Control List (GCL), where each queue is associated with a gate that opens or closes according to a predefined schedule, enabling time-based control so that high-priority traffic is transmitted within specific time windows. The GCL gates open and close at specific time instants based on the global time base provided by gPTP. The accuracy of this alignment is limited by the gPTP synchronization error and each switch hop can add up to one GCL cycle of gating delay, both of which contribute to the end-to-end latency of time-critical traffic.

After passing through the CBS and TAS stages, packets are forwarded to a virtual strict-priority selector, which determines the next packet to transmit based on priority. The selected packet is then transmitted through the underlying network using the encapsulation mechanism described earlier.

Although these mechanisms are implemented in software and do not provide the same level of determinism and accuracy as hardware-based TSN, they preserve the logical behavior of TSN traffic shaping. This allows the V-TSN platform to emulate scheduling effects and traffic interactions, enabling meaningful evaluation of time-sensitive applications.

IV. Evaluation and Results

We evaluate the V-TSN platform deployed on a commercial cloud provider over the public Internet, with two objectives. First, we measure the clock synchronization accuracy achieved by the virtual gPTP mechanism, which determines the quality of the shared time base available to applications. It is relevant to both use cases highlighted in the introduction. Second, for the emulation use case, UC1, we validate that the virtual TAS and CBS reproduce the behavior expected of TSN traffic shapers. The aim of this second part is to confirm that the platform exhibits the characteristic TSN shaping behavior.

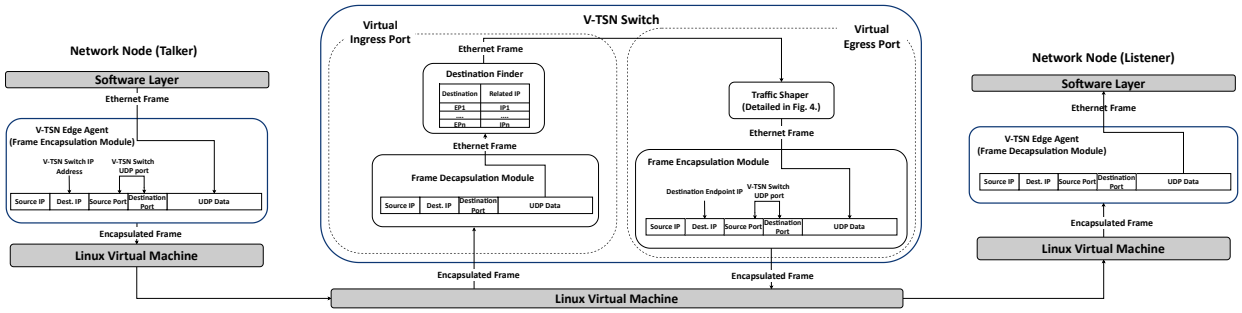


Fig. 2: Virtual packet switching mechanism in the V-TSN platform.

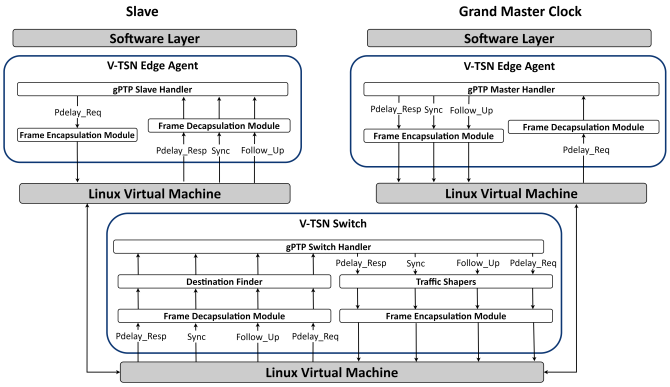


Fig. 3: Virtual gPTP synchronization in the V-TSN platform.

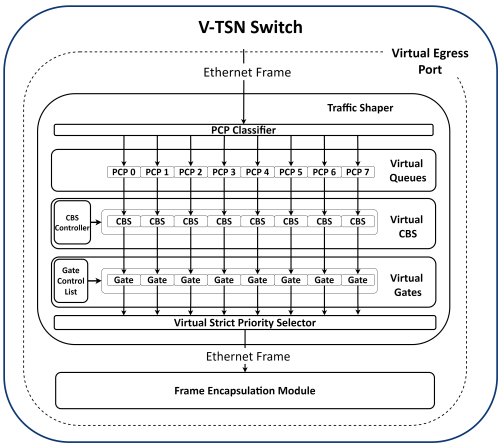


Fig. 4: Virtual traffic shaping architecture inside the V-TSN switch, detailing the traffic shaper of Fig. 2.

A. Experimental Setup

To evaluate V-TSN under realistic conditions, we deploy it on commercial cloud infrastructure rather than a controlled testbed. As summarized in Table II, the experiments use four cloud VMs acting as a grandmaster clock (and also as endpoint EP0), the V-TSN switch, and two endpoints EP1 and EP2. Three nodes are co-located in the same region, while EP2 is in a different country, introducing wide-area variability; the co-located nodes therefore represent a favorable setup and EP2 dominates the wide-area behavior. All machines communicate over UDP across the public Internet, with no TSN-capable hardware and no controlled network conditions. So the overlay runs over a genuinely non-deterministic wide-area network. All endpoint traffic is routed through the V-TSN switch, representative of a real deployment in which distributed nodes are interconnected through general-purpose cloud networks.

TABLE II: Experimental setup summary

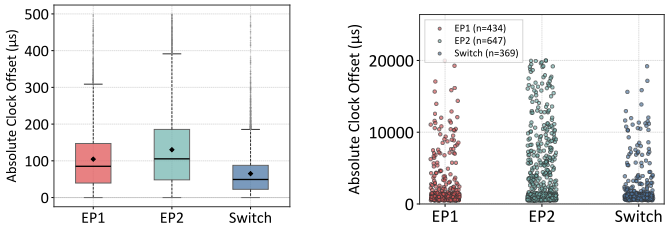
VM	Location	Configuration	Role
VM1	Germany	Hetzner, Linux, 2 vCPU, 4 GB	Grandmaster (EP0)
VM2	Germany	Hetzner, Linux, 2 vCPU, 4 GB	V-TSN Switch
VM3	Germany	Hetzner, Linux, 2 vCPU, 4 GB	Endpoint 1 (EP1)
VM4	Finland	Hetzner, Linux, 2 vCPU, 4 GB	Endpoint 2 (EP2)

B. Time Synchronization Accuracy

We evaluate synchronization accuracy through the clock offset of each node relative to the grandmaster. At each synchronization instant, when a node receives a new reference time from the grandmaster, it computes its offset as the received grandmaster time minus its own local time at that moment. We report the absolute value of this offset. Fig. 5a shows a boxplot of 10,000 such offset samples per node collected over 80 hours, with the diamond marking the mean. For readability, samples exceeding 500 μs are excluded here and shown separately in Fig. 5b.

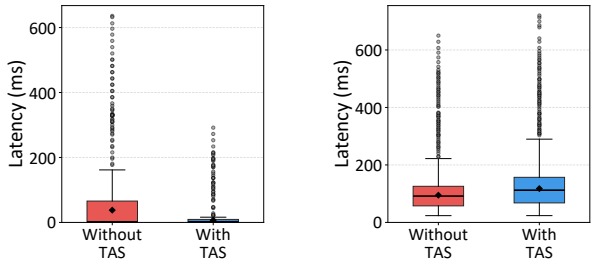
As shown in Fig. 5a, the average clock offset remains below 200 μs for all nodes, with the V-TSN switch the most tightly synchronized, consistent with its co-location with the grandmaster. EP2, which is deployed in a different geographical location, exhibits higher offset values and larger variation compared to the other nodes. Increased geographical distance and the associated network variability have a direct impact on synchronization accuracy.

Fig. 5b presents the offset samples greater than 500 μs. All three nodes exhibit such deviations, with EP1, EP2, and the switch showing 434, 647, and 369 samples above this threshold. These outliers are consistent with transient congestion events on the public Internet, suggesting that WAN latency variability is the dominant factor in worst-case synchronization error. Such deviations represent no more than about 6.5% of each node's samples and the system recovers to normal synchronization accuracy once network conditions stabilize.



(a) Offset within 500 μs. (b) Outliers above 500 μs.

Fig. 5: Clock offset across nodes over approximately 10,000 samples per node: (a) the main distribution and (b) the outliers exceeding 500 μs.



(a) Time-Critical latency with and without TAS. (b) Best-Effort latency with and without TAS.

Fig. 6: Impact of TAS on latency across traffic classes.

Overall, V-TSN achieves sub-millisecond synchronization under typical conditions, with occasional larger deviations caused by the non-deterministic network.

C. Analysis of Virtual TAS Behavior

In this section, we evaluate the behavior of the virtual TAS in terms of traffic isolation and latency. To this end, two traffic streams representing different traffic classes are generated by separate senders and forwarded by the V-TSN switch to a common receiver. Endpoint 1 (EP1) sends a best-effort stream (Traffic Class 0, TC0) and the grandmaster (EP0) sends a time-critical stream (Traffic Class 3, TC3), both arriving independently at the switch, which forwards them to EP2, where the one-way latency of each packet is recorded.

The experiment is designed to create contention at the egress of the V-TSN switch. A bursty Best-Effort (TC0) traffic pattern is introduced to generate congestion on the shared virtual link, potentially blocking Time-Critical (TC3) packets. TC0 traffic is generated as 20-packet bursts every 300 ms, while TC3 traffic follows a steady transmission pattern of 50 packets per second. The egress virtual port is configured with a transmission rate of 1.5 Mb/s. The payload sizes are set to 1200 B for TC0 and 64 B for TC3.

Two configurations are considered. In the first configuration (without TAS), all gates remain open throughout the cycle, resulting in a shared FIFO transmission behavior where TC3 packets must wait behind TC0 traffic. In the second configuration (with TAS), a periodic schedule with a cycle of 14 ms is applied. Within each cycle, TC3 is assigned a dedicated transmission window of 7 ms, while TC0 is assigned the remaining 7 ms. This ensures that TC3 traffic obtains exclusive access to the link during its allocated time window.

The resulting latency distributions are shown in Fig. 6. Fig. 6a shows that TAS sharply reduces the mean and worst-case latency of the time-critical TC3 stream and tightens its distribution. Without TAS, TC3 packets are frequently blocked behind bursty TC0 traffic at the switch egress, giving a mean latency of about 38 ms and a worst case above 600 ms. With TAS, TC3 is served within its dedicated window and shielded from TC0 interference, reducing the mean to about 7 ms and the worst case to about 292 ms. The residual worst case reflects the wide-area path to EP2 rather than switch contention.

Fig. 6b shows that best-effort TC0 latency increases when TAS is enabled, with the mean rising from about 95 ms to 118 ms and the worst case from about 650 ms to 720 ms, as best-effort traffic is held back while the TC3 gate is open.

Overall, the virtual TAS reproduces the key behavior of hardware TAS, providing temporal isolation and prioritized access for time-critical traffic over a non-deterministic network. Note that the underlying Internet adds delay and variability that must be accounted for when designing the GCL schedule.

D. Analysis of Virtual CBS Behavior

In this part, we evaluate the behavior of the virtual CBS and its ability to enforce bandwidth reservations across traffic classes. Similar to the TAS experiment, two traffic streams corresponding to different traffic classes are generated. In this case, two independent senders are used, where the grandmaster node (EP0) is additionally utilized as a traffic source to generate one of the streams.

The egress virtual port is configured with a transmission rate of 10 Mb/s. CBS parameters are configured such that TC0 (best effort) is assigned an idleSlope corresponding to 7 Mb/s, while TC3 (high priority) is assigned 2 Mb/s, leaving remaining bandwidth as headroom. Both traffic streams are generated with time-varying sending rates to create dynamic load conditions that periodically exceed the reserved bandwidth, allowing the shaping behavior to be clearly observed.

Fig. 7 shows the throughput over time. The unshaped traffic follows the input rate and exceeds the desired bandwidth limits during peaks. In contrast, the CBS-shaped traffic is effectively constrained by the configured reservations. In particular, when the input rate exceeds the assigned bandwidth, the throughput is clipped at the configured limit, while for lower input rates, the traffic passes through without modification.

The shaping occurs for both traffic classes simultaneously, confirming that the virtual CBS enforces per-class bandwidth guarantees despite a shared output port, reproducing the rate limiting and fair allocation of hardware CBS under contention. Network-induced delay has less impact on CBS than on TAS, since CBS regulates input rate rather than relying on strict time slots, though software processing overhead may cause slight deviations from ideal CBS.

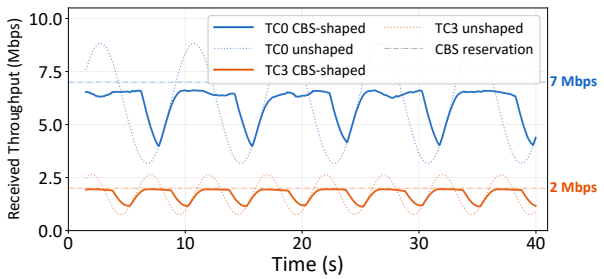


Fig. 7: Effect of CBS on throughput for different traffic classes.

V. Conclusion and Future Work

In this paper, we introduced V-TSN, a software-defined overlay that realizes core TSN functionalities over general-purpose, non-deterministic networks without specialized hardware. The same platform serves two use cases: as a development-time emulation tool for validating TSN-based systems before the target hardware is available, and as a cost-efficient deployment option for applications that tolerate relaxed timing over general-purpose or cloud networks.

A real-world cloud deployment confirms that V-TSN reproduces the essential behavior of hardware TSN mechanisms in software: the virtual gPTP keeps the average clock offset below 200 μs, the virtual TAS isolates time-critical traffic and reduces its mean and worst-case latency under contention, and the virtual CBS enforces per-class bandwidth reservations on a shared output port.

As future work, we plan to combine V-TSN with platforms that virtualize the computation side of distributed systems, such as virtual ECU and application environments. Pairing a virtualized TSN network with virtualized compute would allow an entire time-sensitive system, both its applications and its communication, to be developed and validated end-to-end before any physical hardware is available. An example platform is the AUMOVIO [27] in-vehicle virtualization platform, for which V-TSN could provide the networking layer in a fully virtual end-to-end validation flow for in-vehicle applications.

Acknowledgment

This work has received funding from the European Chips Joint Undertaking under Framework Partnership Agreement No 101139789 (HAL4SDV).

References

[1] T. Stüber *et al.*, "A survey of scheduling algorithms for the time-aware shaper in time-sensitive networking (TSN)," *IEEE Access*, vol. 11, pp. 61192–61233, 2023.

[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, Jun. 19, 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, "IEEE standard for local and metropolitan area networks—virtual bridged local area networks amendment 12: Forwarding and queuing enhancements for time-sensitive streams," *IEEE 802.1Qav-2009 Standard*, pp. 1–72, Jan. 2010.

[5] IEEE Standards Association, "IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks—Amendment 26: Frame Preemption," *IEEE Standard 802*, pp. 10016–5997, 2016.

[6] N. Finn, "Introduction to time-sensitive networking," *IEEE Communications Standards Magazine*, vol. 6, no. 4, pp. 8–13, 2023.

[7] V. Struhár *et al.*, "Real-time containers: A survey," in *2nd Workshop on Fog Computing and the IoT (Fog-IoT 2020)*, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2020.

[8] D. Bezerra *et al.*, "Comparative analysis of TSN simulation and real-world setups: End-to-end latency and forwarding performance," *Computer Communications*, 2026, Art. no. 108472.

[9] A. Nasrallah *et al.*, "Ultra-low latency (ULL) networks: The IEEE TSN and IETF DetNet standards and related 5G ULL research," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 88–145, 2018.

[10] Y. Peng *et al.*, "A Survey on In-Vehicle Time-Sensitive Networking," *IEEE IoT Journal*, vol. 10, no. 16, pp. 14375–14396, 2023.

[11] E. Lopez-Ruiz, C. Arias-Martínez, and J. Sanchez-Garrido, "TSN development platform for aerospace: a versatile TSN tool for the streamlined configuration of aerospace missions in combination with RTEMS," in *EDHPC*, 2025.

[12] M. Karimi *et al.*, "INSIM: A Modular Simulation Platform for TSN-based In-Vehicle Networks," in *2025 VTC-Fall*, IEEE, 2025.

[13] R. Mijumbi *et al.*, "Network Function Virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, 2015.

[14] ETSI GS NFV, "Network functions virtualisation (NFV): Architectural framework," *ETSI GS NFV 002 V1.1.1*, 2013.

[15] R. Chaudhary *et al.*, "A comprehensive survey on software-defined networking for smart communities," *International Journal of Communication Systems*, vol. 38, no. 1, Art. no. e5296, 2025.

[16] A. A. A. Saif and A. H. Shamsan, "Virtualized firewalls: Design, implementation, and security challenges in modern network infrastructures," in *2025 5th International Conference on Emerging Smart Technologies and Applications (eSmarTA)*, IEEE, 2025.

[17] R. Farahi, "A comprehensive overview of load balancing methods in software-defined networks," *Discover Internet of Things*, vol. 5, no. 1, Art. no. 6, 2025.

[18] B. Han *et al.*, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, 2015.

[19] A. Varga, "OMNeT++," *Modeling and Tools for Network Simulation*, pp. 35–59, Springer Berlin Heidelberg, 2010.

[20] D. Andronovici *et al.*, "Cross-validating open source in-vehicle TSN simulation models with a COTS hardware testbed," in *IEEE VNC*, pp. 172–179, IEEE, 2024.

[21] F. Rezabek *et al.*, "Engine: Flexible research infrastructure for reliable and scalable time sensitive networks," *Journal of Network and Systems Management*, vol. 30, no. 4, 2022.

[22] J. O'Raw, D. Laverty, and D. McFadden, "Precision time protocol (PTP) using linuxptp on single board computers," in *2023 58th International Universities Power Engineering Conference (UPEC)*, IEEE, 2023.

[23] K. Correll, N. Barendt, and M. Branicky, "Design considerations for software only implementations of the IEEE 1588 precision time protocol," in *Proc. IEEE 1588 Conference*, 2005.

[24] J. Sasiain, A. Atutxa, D. Franco, J. Astorga, and E. Jacob, "Preliminary approaches towards the integration of TSN communications into the NFV architectural framework," in *XVI Jornadas de Ingeniería Telemática (JITEL)*, 2023.

[25] Y. Zhang, C. Chen, M. Li, C. You, X. Guan, and T. Q. S. Quek, "Online flow scheduling in virtualized time-sensitive networks: A joint admission control and VNF embedding approach," *IEEE Transactions on Mobile Computing*, vol. 24, no. 6, pp. 4651–4667, 2025.

[26] A. Garbugli, L. Rosa, A. Bujari, and L. Foschini, "KuberneTSN: A deterministic overlay network for time-sensitive containerized environments," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2023, pp. 1494–1499.

[27] AUMOVIO SE, "Virtualization," Accessed: Jun. 11, 2026. [Online]. Available: https://www.aumovio.com/en/solutions/services/virtualization.html