Received 1 May 2024, accepted 3 June 2024, date of publication 5 June 2024, date of current version 12 June 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3410109

METHODS

Improving TSN Simulation Accuracy in OMNeT++: A Hardware-Aligned Approach

HOW-HANG LIU[1], STEFAN SENK[1], (Graduate Student Member, IEEE), MARIAN ULBRICHT[1], HOSEIN K. NAZARI[1], TOBIAS SCHEINERT[1], (Graduate Student Member, IEEE), MARTIN REISSLEIN[2], (Fellow, IEEE), GIANG T. NGUYEN[3,4], AND FRANK H. P. FITZEK[1,4], (Senior Member, IEEE)

[1]Deutsche Telekom Chair of Communication Networks, Technische Universität Dresden, 01062 Dresden, Germany
[2]School of Electrical, Computer, and Energy Engineering, Arizona State University, Tempe, AZ 85278, USA
[3]Chair of Haptic Communication Systems, Technische Universität Dresden, 01062 Dresden, Germany
[4]Centre for Tactile Internet with Human-in-the-Loop (CeTI), Technische Universität Dresden, 01062 Dresden, Germany

Corresponding author: Martin Reisslein (reisslein@asu.edu)

This work was supported in part by German Federal Ministry for Economic Affairs and Climate Action (BMWK) through the Project "TICCTEC" under Grant 01MC22007A, through the Project "5G-OPERA" under Grant 01MJ22008A, and through the Project "stic5G" under Grant 01MJ22018C; and in part by German Research Foundation (DFG—Deutsche Forschungsgemeinschaft) as part of Germany's Excellence Strategy—Cluster of Excellence "Centre for Tactile Internet with Human-in-the-Loop"(CeTI), Technische Universität Dresden, under Grant EXC 2050/1—Project ID 390696704.

ABSTRACT Time-Sensitive Networking (TSN) encompasses standards to facilitate near-deterministic performance in wired networks and the integration of TSN with modern wireless systems, such as Fifth Generation (5G) systems, is intensely pursued. Consequently, TSN is examined by numerous research groups. Given the limited and costly access to TSN hardware, a reliable simulator becomes crucial. OMNeT++ is a modular simulator that can be expanded with network simulation models, such as INET, which can simulate TSN functionalities. We evaluate the accuracy of the OMNeT++ INET simulator in mirroring the physical (hardware-based) reality by comparing OMNeT++ INET simulations with measurements of two commercial TSN switches. Our evaluations encompass the generalized Precision Time Protocol (gPTP) accuracy, the Store-and-Forward (SF) and Cut-Through (CT) switching forwarding latencies, the Time Aware Shaper (TAS), and Frame Preemption (FP), including combinations of CT, TAS, and FP. We find that compared to hardware measurements, INET simulations exhibit different clock synchronization dynamics, underestimate the switching latencies, and do not support CT with FP. We enhance the alignment of INET simulations with authentic hardware behaviors by modifying INET modules. We modify the INET gPTP synchronization model and the INET simulation models for the SF and CT forwarding latencies. Also, we modify the INET simulation modules to support the combined operation of CT and FP. We demonstrate how our INET modifications, which we make publicly available, accurately simulate the behaviors of real TSN hardware.

INDEX TERMS Frame preemption (FP), hardware testbed, measurement, OMNeT++, open-source software, simulation, time-aware shaper (TAS), time-sensitive networking (TSN), time synchronization.

The associate editor coordinating the review of this manuscript and approving it for publication was Jingxian Wu.

I. INTRODUCTION

Emerging industrial applications exhibit diverse requirements, with some mandating high availability, reliability, and guaranteed performance in terms of both jitter and latency [1], [2], [3], [4], [5]. For instance, bounded jitter is crucial in machine-to-machine communication, where an uptick in jitter may be treated as packet loss, triggering a robot's transition to a safe mode and causing interruptions in the production line. The benefits of near-deterministic communication extend beyond industry, encompassing, for instance, healthcare [6], Vehicle to Everything (V2X) [7], [8], and tactile Internet communication [9], [10], [11]. Generally, various applications, with their unique requirements, share the same network infrastructure. Effectively managing the coexistence of critical high-priority traffic and regular low-priority traffic within the communication bandwidth poses a notable challenge [12]. Time-Sensitive Networking (TSN) incorporates over 20 standards to enable near-deterministic communication over wired links [13], [14], [15]. The ongoing integration of TSN with 5G aims to expand the application scope of TSN [16], [17], [18], [19], [20], [21], [22] and has drawn significant attention from both academia and industry in recent years.

TSN [23], [24] utilizes diverse standards for effective flow management, whereby some standards rely on the time synchronization facilitated by synchronization protocols [25], such as the generalized Precision Time Protocol (gPTP) for timely transmission. Shaping mechanisms, such as the Time-Aware Shaper (TAS), enable the isolation of high-priority traffic from low-priority traffic, fostering near-deterministic communication [26], [27]. Integrating these standardized mechanisms with others, such as Cut-Through (CT) switching and Frame Preemption (FP) with the Hold-Release mechanism, elevates the achievable level of determinism. Several studies, e.g., [12], [28], and [29], underscored the significant impact of configurations and tools on TSN performance. Consequently, it becomes imperative to thoroughly evaluate these mechanisms across various scenarios and combinations.

A. EXISTING TSN EVALUATION METHODOLOGIES

Hardware deployments provide insights into real-world networks and their practical capabilities, but the cost and limited accessibility to the required hardware pose significant challenges [29]. Moreover, not all hardware includes the desired features. Alternatively, simulations, with a level of simplification, prioritize rapid iterations, enabling the testing of diverse ideas across various topologies and scales. Diverse simulators are utilized to simulate TSN networks. Recent research [29] reveals that about 53% of TSN simulations utilize the OMNeT++ framework [30], while 14% use the Riverbed Modeler (formerly OPNET Modeler Suite), and 33% use various other simulators. Due to the dominant role of the OMNeT++ framework in TSN simulations, we focus on OMNeT++ in this study. However, similar to our study methodology, the Riverbed Modeler and the other simulators should be validated with hardware measurements in future research. OMNeT++ lacks networking models in its simulation framework and works with external networking libraries, such as the INET framework. INET encompasses a diverse set of simulation models, including TSN features. Approximately 80% of TSN simulations based on OMNeT++ leverage the INET framework for performance evaluation [31].

Ensuring that simulation results closely align with hardware outcomes is crucial for accurately reflecting real-world scenarios. However, most validation studies for OMNeT++ results, such as [31] and [32], have not considered TSN features. One notable exception is a study based on the Engine platform [33], [34]. The Engine platform, initially designed for reproducible TSN network emulation, is expanded in [35] to include configuration capabilities for OMNeT++ and INET. The findings in [35] indicate that simulations generally exhibit lower delays and jitters compared to hardware. While the results in [35] shed light on traffic-shaper behaviors in simulation, the study refrains from modifying the simulation framework to enhance the simulation modeling for better alignment with hardware characteristics.

B. SUMMARY OF ORIGINAL CONTRIBUTIONS

We rigorously evaluate the simulation accuracy by comparing INET results with our TSN-FlexTest testbed [28], which was developed for measurements of real-world TSN hardware as a validation reference. Specifically, we utilized the Kontron D10 MMT Series and FibroLAN Falcon-RX/G TSN switches in our TSN-FlexTest testbed. To ensure the timely transmission of high-priority traffic, the corresponding transmitters were equipped with Intel I210 interface cards. Initially, we investigate clock deviations across different gPTP configurations, identifying discrepancies in the INET results. We modify INET and demonstrate through hardware measurement results that the modified INET version improves alignment with real-world scenarios across diverse gPTP settings.

We examine the switch forwarding latency, finding that simulated Store-and-Forward (SF) switching exhibits lower latencies than the hardware measurements. Furthermore, CT in the original INET simulation model maintains a stable latency for all packet sizes, while hardware measurements show that the latency increases linearly for packet sizes below a certain threshold. We modify the INET framework to better reflect the forwarding latency of the real hardware testbed. Simulations with the modified INET demonstrate that our modifications produce more accurate outcomes. We publicly provide the source code for the modified INET [36].

We also investigate the accuracy of TAS simulations, revealing that our modified INET version aligns the simulations closely with hardware measurements. Moreover, we find that when FP is combined with CT, then the original INET simulator exhibits higher latency than hardware measurements due to the conflict of enabling CT and FP. We modify INET to resolve this conflict and demonstrate that the modified INET achieves sub-microsecond accuracy for the combined TAS, CT, and FP features. In general, we align the INET simulator more closely with the latency characteristics of real TSN hardware, providing researchers with an improved simulation tool for evaluating TSN networks.

The paper is organized with the following section structure. A comprehensive definition of latency is presented in Section II, which also introduces the fundamental features of TSN. Section III presents a summary of relevant literature on software simulations and hardware testbed validations of TSN features, highlighting the distinguishing original aspects of our study. In Section IV, we model the processing delay within switches and compare the CT mode with the SF mode. Section V describes the testbed and simulation setups employed for our comparative validation experiments, while the comparison results are presented in Section VI. Finally, we summarize key insights from our study in Section VII.

II. BACKGROUND

This section defines the latency metrics and introduces four core TSN features, namely gPTP, CT, TAS, and FP. These four core TSN features are fundamental pillars employed in various scenarios and can support a broad range of industrial use cases [2], [3], [8], [12], [14], [26], [29]. Importantly, INET has adopted these core features for implementation, thereby underscoring their significance.

A. DEFINING LATENCY

Four different delay components in a network node are identified in [37, Section 1.4.1]: The processing delay is the time interval that it takes to examine the packet header and to identify the correct output port. During the queuing delay, the packet waits for the previously arrived packets to be transmitted. The transmission delay is the time interval needed to push all of the packet's bits into the link. The propagation delay is the time interval that it takes for the bits to propagate from the beginning of the link to the end, which depends on the physical medium and the length of the link.

Latency is commonly measured in two ways: the end-to-end delay or the first-bit to first-bit delay. The end-to-end delay encompasses the transmission delay of the sender (source node), the propagation delay from the source node to the first intermediate network node (switch), and all four delay components in all intermediate switching nodes between the sender and the receiver (destination node). That is, the end-to-end delay is the time interval from the time instant when the sender starts the transmission of the first bit of the packet to the time instant when the last bit of the packet is received by the destination node.

On the other hand, the first-bit to first-bit delay is the time interval between the time instant when the sender starts transmitting the first bit of the packet to the time instant when the receiver begins receiving the first bit of the packet.

Collectively, we refer to the two latency metrics (end-to-end and first-bit to first-bit) as representing the "forwarding latency" in discussions that address the general latency characteristics of a network. Throughout this study, we examine the one-way forwarding latency from a sender to a receiver (and do not consider the round-trip latency).

B. SELECTED TIME-SENSITIVE NETWORKING FEATURES

1) TIME SYNCHRONIZATION WITH PTP

The Precision Time Protocol (PTP) is a clock synchronization protocol commonly used in networks. The PTP can achieve sub-microsecond accuracy and is primarily designed for mission-critical applications. The PTP employs a master-slave architecture, where the master clock transmits the synchronization message to the slave clock along with a timestamp. The slave clock records the received timestamp and responds to the master clock with the delay-request message containing the send-out time. The master clock includes the timestamp when it received the request message and sends a delay-response message back to the slave clocks. Through this process, slave clocks can adjust their time to synchronize with the master clock.

The generalized PTP (gPTP) protocol, defined by IEEE 802.1AS [38], is an adaptation of the PTP to the requirements of TSN applications. There are three key distinctions between gPTP and PTP: First, gPTP operates exclusively at the data link layer (layer two) for all communication, whereas PTP offers support for higher protocol layers. Second, in gPTP, devices capable of gPTP directly exchange information with other gPTP devices, while the PTP protocol accommodates non-PTP-capable devices between two PTP devices. Third, gPTP makes it easier to classify time-aware systems by reducing the types of PTP Instances [38, Section 7.5] to only two: time-aware stations (PTP End Instances) and time-aware bridges (PTP Relay Instances), in contrast to the four clock types (ordinary clocks, boundary clocks, end-to-end transparent clocks, and P2P transparent clocks) in PTP. Consequently, gPTP presents a simplified profile of PTP, reducing configuration complexity and enhancing performance in terms of minimized jitter and accelerated synchronization.

2) FORWARDING WITH CUT-THROUGH (CT) SWITCHING

Packet switching networks commonly use SF switching. CT switching is an ongoing project involved in the TSN standard [39]. With CT switching, the frame forwarding begins as soon as the destination address is identified. CT reduces the contribution of the frame transmission delay component in the intermediate network nodes to the end-to-end delay and the first-bit to first-bit delay compared to SF switching, which requires complete frame reception before forwarding the frame. However, this latency reduction comes at the expense of not checking the integrity of frames at the intermediate switches, which can lead to error propagation. For instance, if all layer-two intermediate nodes work in CT switching mode, only the receiver can check the frame integrity.

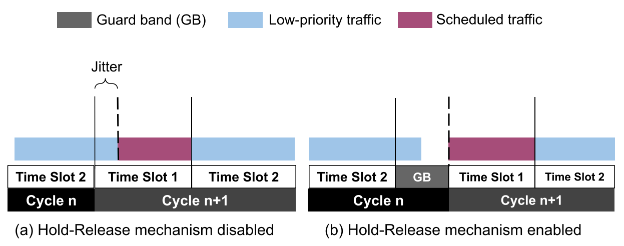


FIGURE 1. Illustration of effects of Hold-Release mechanisms on response times and jitter for low-priority traffic (blue rectangle) and scheduled traffic (purple rectangle).

3) SCHEDULING AND SHAPING WITH TIME-AWARE SHAPER (TAS)

TAS was introduced in [40] to provide bounded latency and jitter for high-priority Scheduled Traffic (ST) streams by isolating them from background traffic streams [41]. This isolation is achieved by allotting transmission times for different frame classes. In TAS, traffic streams are enqueued into a certain number of queues, each representing a Quality-of-Service (QoS) level. By opening and closing the gates of each queue, TAS offers various QoS levels. This queue management complies with a Gate Control List (GCL) that repeats cyclically. The GCL specifies which queues are permitted to transmit when and for how long.

For isolating high-priority ST streams from background traffic streams, each GCL cycle is commonly composed of two distinct time slots: During time slot one, only high-priority traffic is permitted for transmission, while time slot two allows for the transmission of lower-priority traffic. This concept is founded on a time-division multiple access scheme, which effectively allocates time intervals for different traffic priorities. This scheme requires that all network nodes are synchronized [42], which ensures that the sender is well-informed about the designated time slot (slot one) for transmitting ST streams, thereby preventing collisions with other traffic streams. With the implementation of this mechanism, the network can effectively prioritize and manage different traffic types, enabling hard real-time traffic, soft real-time traffic, and background traffic to coexist non-reactively on the same Ethernet infrastructure.

4) INTERRUPTING TRANSMISSIONS WITH FRAME PREEMPTION (FP)

FP was introduced in [43] to reduce the worst-case response times and to ensure that time-critical frames receive priority over lower-priority or non-time-critical frames. The FP method is often beneficial in the presence of large non-time-critical frames. To reduce the latency of time-critical traffic, the switch suspends the lower-priority traffic by preempting the ongoing transmission of a lower-priority frame and transmitting time-critical traffic instead. After full transmission of the time-critical frame, the remaining data in the previously preempted frame can be transmitted. FP is a MAC layer extension; whereby, both the sending and receiving node of a given link must support FP in order to employ FP on the communication link connecting the two nodes [43], [44].

Annex S of the IEEE 802.1Q-2018 standard specifies two approaches for implementing TAS with FP: with or without the Hold-Release mechanism. The choice of this mode influences the response time and jitter [45]. Figure 1 contrasts the system behavior with an example when the mechanism is disabled (subfigure a) and enabled (subfigure b). In the depicted example, the gate opens for high-priority ST in the first time slot and for low-priority traffic in the second time slot.

In the absence of the Hold-Release mechanism [Fig. 1(a)], the low-priority preemptable traffic can transmit up to 123 bytes (B) uninterrupted, even when the ST gate is open because frames of 123 B or smaller cannot be preempted [46]. In this example, the low-priority traffic continues transmission even after time slot 2 in cycle n finishes. Consequently, the transmission of ST starts in cycle $n + 1$ only after the ongoing transmission of low-priority traffic is completed, leading to jitter for ST.

With the Hold-Release mechanism enabled [Fig. 1(b)], a guard band is enforced prior to the start of time slot 1 for ST traffic in cycle $n + 1$. An ongoing traffic transmission can continue within the guard band. However, the ongoing transmission must either be completed before the onset of the next time slot or must be preempted and resumed when the next slot (time slot 2 in cycle $n+1$) becomes available for low-priority packets. No new frame transmission can commence in the guard band. In essence, enabling the Hold-Release mechanism increases communication delay for low-priority frames, while decreasing the jitter for ST frames.

III. RELATED WORK

A. GENERAL OVERVIEW

This section reviews the related literature on INET verification and TSN feature evaluation. The first release of INET was in 2010, without TSN protocols. Based on pure INET and OMNeT++, CoRE4INET [47] is an open-source project which supports TSN features, including IEEE 802.1Qci and IEEE 802.1Qbv (TAS). NeSTiNg [48] was published in 2019 and implemented TSN functionalities based on pure INET. The NeSTiNg evaluation study [48] compared TAS and FP (independently, not in combination) with the strict priority mechanisms in the NeSTiNg implementation. The TSN framework and capabilities released from INET in 2022 are based on the experience gained from CoRE4INET and NeSTiNg; specifically, the 2022 INET release includes gPTP, CT, TAS, and FP. To clarify, in this study, we focus on verifying the TSN functionalities in the INET release.

Regarding the accuracy of INET simulations in general, to our knowledge, there have been only three validation studies that have used networking hardware to verify simulation results. One validation study focuses on WiFi communication [31], while another validation study investigates the real-time Ethernet protocol [32]. Therefore, these two

TABLE 1. Comparison of related studies on evaluation of the four core TSN features with INET simulations (SW) or measurements in hardware (HW) testbeds.

Type	Ref.	PTP	CT	TAS	FP
SW	Heise et al. [49]	☐	☐	☐	☒
	Ashjaei et al. [45]	☐	☐	☐	☒
	Huang et al. [50]	☐	☐	☒	☐
	Falk et al. [48]	☐	☐	☒	☒
	Arestova et al. [51]	☐	☐	☒	☒
HW	Ma et al. [52]	☐	☐	☐	☒
	Konradi et al. [53]	☒	☐	☒	☐
	Miranda et al. [54]	☒	☐	☒	☐
	Hagargund et al. [55]	☐	☐	☒	☐
	Ulbricht et al. [28]	☒	☒	☒	☐
HW+SW	Bosk et al. [35]	☐	☐	☒	☐
	This study	☒	☒	☒	☒

validation studies are not directly relevant to the validation of the INET TSN capabilities. The third study is relevant as it verified the INET TSN capabilities with measurements of TSN hardware [35], as elaborated in Section III-D.

As TSN features released from INET were introduced in May 2022, there has been very limited research to date on the accuracy of the implementation of the TSN features in INET. However, a number of studies have evaluated the core functions of TSN in various scenarios. As summarized in Table 1, we review studies that focus on evaluating TSN core functions with simulations conducted using OMNeT++ INET as well as with measurements based on hardware testbeds. Overall, we note that there is a pronounced lack of literature on validating the accuracy of TSN simulations in comparison to measurements with real TSN hardware, whereby validation studies of simulations that combine multiple TSN functions are entirely lacking.

B. INET SIMULATIONS OF CORE TSN FEATURES

Several existing studies have evaluated the four core TSN functions with OMNeT++ INET simulations. However, the existing simulation studies typically focused on an individual or a subset of the four core TSN functions. For instance, Heise et al. [49] present an open-source TSN simulation framework based on OMNeT++ and simulated FP. However, in [49], FP is simulated in isolation as the simulated scenario does not account for time synchronization. Similarly, Ashjaei et al. [45] design a novel FP technique and evaluate the performance of the proposed FP technique using OMNeT++ in different scenarios, assuming perfect time synchronization. However, evaluating FP in isolation overlooks the potential performance gains that could be obtained by utilizing TSN features in combination.

The combination of TAS and limited TSN features in INET simulation has been investigated in several prior studies, including [48], [50], [51]. Huang et al. [50] comprehensively simulate Cyclic Queuing and Forwarding (CQF) and develop a novel scheduling algorithm called Time-Aware Cyclic-Queuing (TACQ), which combines TAS with CQF. TACQ achieved zero jitter and lower delay for isochronous flows compared to CQF in the evaluations in [50]; however, the precision of time synchronization was neglected in [50]. Falk et al. [48] and Arestova et al. [51] simulated TAS and FP separately (i.e., without combining them) in the INET simulation framework; in contrast, we evaluate TAS operating jointly with FP in this study.

C. HARDWARE MEASUREMENTS OF CORE TSN FEATURES

Ma et al. [52] implemented latency control label scheduling in an asynchronous bridged network hardware testbed. Ma et al. [52] claim to have included FP in their design; however, no comparison to non-frame preemption is presented in [52]. Konradi et al. [53] implemented a TSN testbed that integrates the Open Platform Communications Unified Architecture (OPC UA) PubSub specification with TSN, allowing researchers to evaluate the interoperability of TSN devices from different vendors.

Miranda et al. [54] built a cloud-based testbed with Software Defined Networking (SDN) controller to enable TSN deployments. However, both studies [53], [54] focus on PTP synchronization with TAS, while their implementations lack consideration of the other core TSN features, such as FP and CT, which we consider this study. Hagargund et al. [55] built an open-source, SDN-based TSN testbed that integrates 802.1Qbv (TAS) and 802.1Qcc (Stream Reservation Protocol). The open-source testbed [55] includes the software, which uses the Linux queuing discipline to implement virtual queues.

In our previous research [28], we have created the TSN-FlexTest hardware testbed that is flexible, publicly accessible, and open-source. With the TSN-FlexTest hardware testbed, we measured the end-to-end latency in various scenarios, such as CT and TAS (both in combination with time synchronization); however, we did not measure FP in [28]. To address this omission, we extend our TSN-FlexTest testbed measurements to include FP in the current study; and, we compare all TSN-FlexTest testbed measurements with INET simulations.

D. RESEARCH WITH BOTH HARDWARE AND SIMULATION

Bosk et al. [35] expanded the TSN Engine platform [33], which involves a Commercial-of-the-Shelf (COTS) hardware testbed, to the simulation realm by translating the hardware configuration to the OMNeT++ INET simulator, thus enabling the same setup to run on both hardware and in simulation. The evaluations of Bosk et al. [35] indicate that the simulations typically give lower delays and jitters than the hardware measurements. This discrepancy can be attributed to the idealistic nature of simulations, which do not account for typical system artifacts experienced in hardware setups. However, the validation comparisons in [35] only focus on TAS and the Credit-Based Shaper (CBS) in isolation. The impact of clock drift in the PTP is not considered in [35].

In contrast, our TAS evaluations in this study consider TAS operating in conjunction with PTP.

E. ORIGINAL CONTRIBUTIONS OF THIS ARTICLE

The present study aims to address two gaps in the existing literature, making two main contributions:

1) METHOD CONTRIBUTION

We evaluate the accuracy of the TSN features implemented in INET through hardware testbed measurements, and we modify the INET simulation framework to improve the simulation accuracy. We make the modified INET publicly available [36] so as to advance the openly accessible evaluation methodologies for TSN.

2) RESEARCH CONTRIBUTION

We conduct simulation evaluations with our modified INET and corresponding hardware measurements for scenarios that include PTP, CT, TAS, and FP to demonstrate the benefits of combining TSN features; specifically, we evaluate the combinations PTP+CT+TAS and CT+TAS+FP.

IV. MODIFICATIONS OF INET SIMULATION FRAMEWORK

This section describes how we have enhanced the INET software to mimic the behavior of real TSN switches. In Section IV-A, we provide an overview of the INET TSN software modules. In Section IV-B, we model the switch forwarding delays in the SF and CT modes and introduce new parameters in the INET delay models. In Section IV-C, we explain the gPTP implementation in INET and provide a new approach for modeling the oscillator compensation. In Section IV-D, we describe our INET modification to simulate CT switching in combination with FP and TAS. Table 2 summarizes the parameter notations for the delay and gPTP modeling.

A. OVERVIEW OF INET TSN MODULES

OMNeT++ is a C++ framework that enables the construction of network simulators using a component-based architecture. OMNeT++ comes with the INET library, which provides models for various Internet protocols and components. The INET library includes TSN protocols, making the INET library more comprehensive than other available network simulators. We currently use version 6.0 of OMNeT++ and version 4.5.2 of INET. The TSN features were for the first time included in the INET release 4.4, initially launched in May 2022. As illustrated in Figure 2, the INET TSN encompasses essential features, including time synchronization, per-stream filtering and policing, scheduling and traffic shaping, frame replication and elimination, frame preemption, and CT switching.

This study specifically focuses on the evaluation of gPTP time synchronization, CT switching, TAS scheduling and traffic shaping, and FP within INET TSN. To complement these features, INET includes specialized TSN-specific network devices, such as a TSN clock, TSN device, and TSN switch. A TSN clock emulates a hardware device functioning as a gPTP master node, which is essential for time synchronization. A TSN device models a TSN end device with the capability to run multiple applications. ATSN switch represents an Ethernet switch tailored to support the aforementioned TSN features.

TABLE 2. Summary of notation.

Notation	Definition
Ethernet Frame and Packet Characteristics	
f	Size of Ethernet frame on link layer in bytes
P	Size of Interpacket Gap (IPG) + Preamble + Start Frame Delimiter (SFD) in bytes, $P = 20\,\mathrm{B}$ [56]
$P+f$	Size of Ethernet packet on physical layer in bytes
Network Characteristics	
N	Number of intermediate nodes (switches), $N = 1, 2, \ldots$
L	Link capacity in byte per second
Switch Characteristics	
Δ	Data size in bytes for fwd. decision making in CT
d_l	Lookup delay of switch
d_f^{frame}	Fabric transit delay for a whole frame
d_f^{bit}	Fabric transit delay for one bit
d_b	Internal buffering delay of switch
F_T	Threshold of frame size in bytes
Delay Model	
d	Delay component
d_t	Packet transmission delay, $d_t = (P+f)/L$
d_p	Link propagation delay
d_{sf}	Switch delay for SF mode
d_{ct}	Switch delay for CT mode
α, β	Slope, intercept for modified d_{sf} and d_{ct} models
D	End-to-end delay
D_{sf}	End-to-end delay for SF switch mode
D_{ct}	End-to-end delay for CT switch mode
gPTP Model	
η	Neighbor rate ratio
T_s	Original time of slave clock when receiving `Sync_Follow_Up` message
T_s'	New time of slave clock after calculation when receiving `Sync_Follow_Up` message
T_m	Timestamp when the master clock sends `Sync` message
c	Correction field
r	Residence time
F_n^s	New frequency compensation of slave clock when receiving nth `Sync_Follow_Up` message, $n = 1, 2, \ldots$
γ, δ	Weighting parameters for calculating F_n^s
Γ	Threshold to enable setting the clock time

We modified the code of the TSN switch module. In order to explain the context for these modifications, we describe the software components of the TSN switch in detail. On the right side of Figure 2, the bottom component is the Ethernet interface module which enables Ethernet connectivity. This module acts as a physical port to the hardware. The Ethernet layer module is situated above the interface and has the capability to support the 802.1Q protocol. Above the Ethernet layer are the bridging layer and the logical link control modules. These modules offer packet forwarding, interface

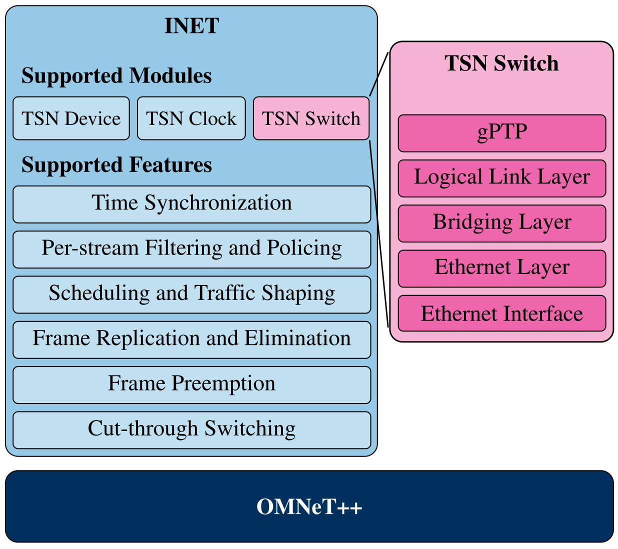


FIGURE 2. **Illustration of the TSN features and modules that INET supports, as well as the illustration of the software architecture within the TSN switch module.**

selection, virtual LAN handling, and logical link control functionalities. Lastly, at the topmost level is the gPTP module which measures link delays to neighboring nodes and provides time synchronization.

B. CUT-THROUGH SWITCHING

We initially model in Section IV-B1 the end-to-end delay in the SF mode, to then arrive at an end-to-end delay model for the CT mode. In Section IV-B2, we describe our modifications of the INET switch simulation models to enhance the accuracy of the INET SF and CT delay models.

1) SWITCH DELAY MODELING

Figure 3 compares the SF end-to-end delay (D_{sf}) and the CT end-to-end delay (D_{ct}) for a network with one intermediate switch. Generally, we denote N for the number of intermediate nodes, i.e., switches. As summarized in Table 2, we denote P [in bytes] for the mandatory Interpacket Gap, Preamble, and Start Frame Delimiter that precede the transmission of the first byte of the link-layer Ethernet frame of size f [bytes]. Thus, the size of a physical-layer Ethernet packet is $P+f$ [bytes], resulting in the packet transmission delay $d_t = (P+f)/L$. We denote d_p for the propagation delay of a link. The variable d_l refers to the time required by a switch for looking up and processing information in order to determine the destination for forwarding.

a: STORE-AND-FORWARD (SF) END-TO-END DELAY

For SF, d_f^{frame} is the time for the fabric transit for the whole (link layer) frame. Each switch incurs a processing delay $d_l + d_f^{\text{frame}}$, which is the time interval from the time instant when the switch has received the last bit of a given packet from the ingress port to the time instant when the switch commences transmitting the first bit of the packet from the egress port. The processing delay $d_l + d_f^{\text{frame}}$ can be shorter than 32 ns [57].

We assume that the delay components d_t, d_p, d_l, and d_f^{frame} are the same for all N switches and links; also, we assume that the sender incurs the packet transmission delay d_t and that the link from the sender to the first switch has propagation delay d_p. Summing the processing delays of the N switches as well as the transmission delays of the sender and the N switches and the propagation delays on the $N+1$ links that interconnect the sender via the N switches with the receiver gives the SF end-to-end delay:

$$D_{\text{sf}} = (d_l + d_f^{\text{frame}}) \times N + (d_p + d_t) \times (N+1). \quad (1)$$

For an SF switch, we define the store-and-forward switch (SF switch) delay d_{sf} as the time interval from the time instant when the considered switch begins to receive the first bit of a given packet (from the preceding node) to the time instant when the switch commences the transmission of the first bit of the packet (to the next node). As illustrated in Figure 3(a), the SF switch delay d_{sf} consists of the packet transmission delay d_t and the processing delay $d_l + d_f^{\text{frame}}$, i.e.,

$$d_{\text{sf}} = d_t + d_l + d_f^{\text{frame}}. \quad (2)$$

We emphasize the contributions of the N switches to the end-to-end latency D_{sf} by grouping the N related delay components together:

$$D_{\text{sf}} = (d_{\text{sf}} + d_p) \times N + d_p + d_t. \quad (3)$$

Thereby, as illustrated in Figure 3(a), the delay $(d_{\text{sf}} + d_p) \times N$ is due to the switches, while the remaining d_p and d_t can be interpreted to account for the transmission delay of the sender and the propagation delay from the sender to the first switch. Commonly, the transmission delay component d_t (as the main contributor to the SF switch delay d_{sf}) dominates the end-to-end delay D_{sf}; for example, transmitting a 1500 B frame via a 1 Gbit s^{-1} link takes 12 µs and propagating through a 2 m cable requires 6.67 ns.

b: CUT-THROUGH (CT) END-TO-END DELAY

We define the cut-through switch (CT switch) delay d_{ct} as the time interval from the time instant when the considered switch begins to receive the first bit of a given packet (from the preceding node) to the time instant when the switch commences the transmission of the first bit of the packet (to the next node), as illustrated in Figure 3(b). With CT switching, the transmission of a given packet can commence before the packet has been fully received, thus reducing the end-to-end delay, as observed by comparing Figure 3(b) with Figure 3(a). Formally,

$$D_{\text{ct}} = \left(d_{\text{ct}} + d_p\right) \times N + d_p + d_t, \quad (4)$$

whereby the CT switch delay d_{ct} is typically substantially shorter than the SF switch delay d_{sf}.

According to [56] and [58], the CT switch delay d_{ct} can be modeled based on the size P of the frame preamble

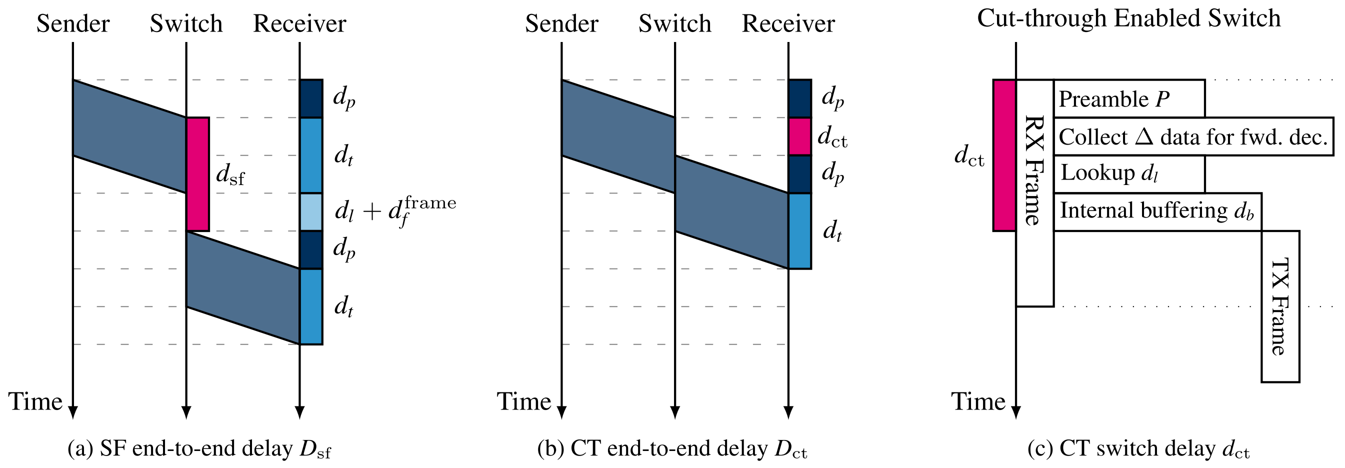

Sender
Switch
Receiver
d_p
d_t
d_{sf}
$d_l + d_f^{\text{frame}}$
d_p
d_t
Time
(a) SF end-to-end delay D_{sf}
Sender
Switch
Receiver
d_p
d_{ct}
d_p
d_t
Time
(b) CT end-to-end delay D_{ct}
Cut-through Enabled Switch
d_{ct}
RX Frame
Preamble P
Collect Δ data for fwd. dec.
Lookup d_l
Internal buffering d_b
TX Frame
Time
(c) CT switch delay d_{ct}

FIGURE 3. Illustration of end-to-end delay D with one intermediate switch between the sender and the receiver. The SF switch delay d_{sf} consists of the packet transmission delay d_t and the switch processing delay $d_l + d_f^{\text{frame}}$; whereas, the CT switch delay d_{ct} consists of the components illustrated in part (c).

[IPG, Preamble, and SFD in byte], the amount Δ of frame data [in byte] required for the forwarding decision, the link capacity L [in byte/s], the lookup delay d_l, the internal switch buffering delay d_b, and the fabric transit delay for one bit d_f^{bit}:

$$d_{\text{ct}} = \frac{P + \Delta}{L} + d_l + d_b + d_f^{\text{bit}}. \tag{5}$$

The CT switch delay d_{ct} is illustrated in Figure 3(c): a switch has to collect sufficient data ($P + \Delta$ bytes) to decide on the forwarding, and then incurs the lookup delay and the internal buffering delay [d_f^{bit} is neglected in Figure 3(c)]. This CT switch delay equals the time interval from the time instant when the first bit of the frame preamble (i.e., the first bit of the packet) begins to be received (RX) in the CT switch to the time instant when the switch commences the transmission (TX) of the first bit of the packet.

c: FIRST-BIT TO FIRST-BIT DELAY

The first-bit to first-bit delay signifies the time taken by a frame to traverse the intermediate nodes (i.e., the switches). According to Figure 3, for the case of one intermediate node, the first-bit to first-bit delay of SF is $2d_p + d_{\text{sf}}$; while CT incurs the first-bit to first-bit delay $2d_p + d_{\text{ct}}$. For the general case, with N switches, the SF first-bit to first-bit delay is $2d_p + (N \times d_{\text{sf}})$, while the CT first-bit to first-bit delay is $2d_p + (N \times d_{\text{ct}})$. Clearly, the shorter CT first-bit to first-bit delay is due to the shorter CT switch delay d_{ct} (compared to the longer SF switch delay d_{sf}).

2) MODIFICATIONS OF INET SWITCH DELAY MODEL

a: STORE-AND-FORWARD (SF)

The default INET d_{sf} model only considers the packet transmission delay d_t; and does not consider the lookup delay d_l, nor the fabric transit delay d_f^{frame}. Since we do not have the ability to measure the d_l and d_f^{frame} separately from our commercial TSN switches, we define a parameter β_{sf} as a constant to model the combined lookup and fabric transit delay, i.e.,

$$\beta_{\text{sf}} = d_l + d_f^{\text{frame}}. \tag{6}$$

With the delay constant β_{sf}, we modify the INET SF forwarding latency model to:

$$d_{\text{sf}} = d_t + \beta_{\text{sf}}. \tag{7}$$

The parameter β_{sf} of the modified INET SF delay model is a constant for a given hardware switch and can be readily measured, e.g., with the TSN-FlexTest testbed [28]. We provide the β_{sf} values for two common TSN switches in Table 4. In terms of computational complexity, our modified INET SF delay model only requires the addition of the constant β_{sf} to the default INET SF delay model ($d_{\text{sf}} = d_t$), which is a negligible added computational effort.

b: CUT-THROUGH (CT)

We modify (5) to mimic the behavior of a real CT switch. Real CT switches typically exhibit linearly increasing switch delay with increasing Ethernet frame size f, up to a prescribed frame size threshold F_T [in byte] at which the CT switch delay plateaus [28], [59].

We specify a linearly increasing delay using two parameters: the slope α_{ct} and the intercept β_{ct}. Specifically, α_{ct} represents the rate (gradient) at which the delay increases linearly with the frame size f. Conversely, β_{ct} signifies a delay component that models the hardware capability. More specifically, we consider d_b in (5) as a parameter related to the frame size f, which produces the linearly increasing portion of the CT forwarding delay d_{ct}, proportional to the frame size. The remaining parameters, i.e., P, Δ, L, d_f^{bit}, only depend on the switch hardware capability.

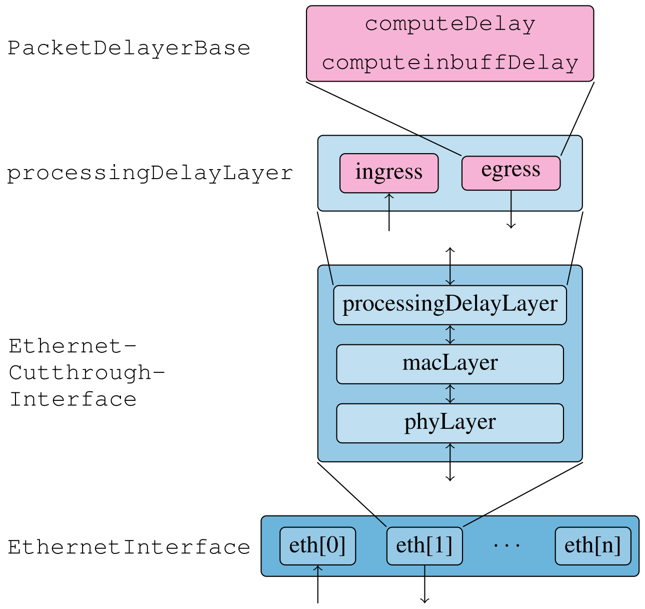


FIGURE 4. **Detailed illustration of the sub-module structure of the INET Ethernet Interface in the TSN Switch in Fig. 2: We modified the** `PacketDelayerBase` **module at the top by adding a configurable processing delay computed by the new** `computeinbuffDelay` **function.**

For our INET modification, we model the CT forwarding delay d_{ct} as a linear function of the frame size f:

$$d_{\text{ct}} = \begin{cases} \alpha_{\text{ct}} f + \beta_{\text{ct}} & \text{if } f \le F_T \\ \alpha_{\text{ct}} F_T + \beta_{\text{ct}} & \text{if } f > F_T. \end{cases} \tag{8}$$

We implemented this CT switch delay d_{ct} model in all versions of our modified INET simulation framework [36]. The slope α_{ct} and intercept β_{ct} are configurable parameters in [36]. For a given hardware switch, α_{ct} and β_{ct} are constants that can be readily obtained from measurements, e.g., with the TSN-FlexTest testbed [28]. We provide the α_{ct} and β_{ct} values for two common commercial TSN switches in Table 4. From a computational complexity perspective, the d_{ct} model in Eqn. (8) requires one if-then condition, one multiplication, and one addition; whereas, the default INET CT model neglects d_{ct} entirely, thus avoiding computational cost but incurring errors (see Fig. 8).

c: MODIFIED DELAY MODEL IMPLEMENTATION IN INET

For SF, there is an existing INET module, namely `PacketDelayerBase`, allowing the addition of the constant delay β_{sf}.

For CT, we changed the switching behavior in the `EthernetInterface` module as visualized in Figure 4. To enable the CT mode, we need to configure the egress port of the TSN switch, which is defined by the `EthernetCutthroughInterface` module. By default, this module only contains the `macLayer` and `phyLayer` submodules. Therefore, we manually include the `processingDelayLayer` (with the two submodules `ingress` and `egress`) in the `EthernetCutthrough Interface`. To implement our CT delay model, we only add latency to the `egress` module. We configure the `egress` module as a `PacketDelayerBase` module and implement a new function called `computeinbuffDelay`. This function accepts a packet object as input and calculates the CT switch delay d_{ct} based on (8).

C. TIME SYNCHRONIZATION IN INET FRAMEWORK

We focus on the gPTP clock synchronization mechanism implemented in INET, which is aligned with the TSN standard [38]. Our investigation focuses on a network architecture consisting of a master clock (node) and a slave clock (node), as depicted in Figure 5. Time synchronization in INET is achieved through two fundamental mechanisms. The first mechanism entails the measurement of propagation delays through peer-delay messages. The second mechanism revolves around the distribution of the master clock time via sync messages.

1) gPTP CLOCK SYNCHRONIZATION PROCEDURE

In the current INET release (4.5.2), the slave clock initiates the propagation delay measurement process. The slave clock accomplishes this by transmitting the `Pdelay_Req` message to the master clock. The master clock, in turn, responds by sending both the `Pdelay_Resp` and the `Pdelay_Resp_Follow_Up` messages. As is common in PTP descriptions, we use the terms "clock" and "node" interchangeably. The reception of the `Pdelay_Resp_ Follow_Up` message by the slave clock enables the slave clock to compute the propagation delay between the master node (clock) and the slave node (clock), as

$$d_p = \frac{\eta(t_4 - t_1) - (t_3 - t_2)}{2}; \tag{9}$$

whereby, we consider the physical link propagation delay d_p (of the direct master-slave link). Our model does not consider multi-hop scenarios (with intermediate nodes between master and slave) since the INET gPTP does currently not implement the multi-hop gPTP functionality completely; for example, the correction field is always set to zero, which inherently introduces errors into the multi-hop master-slave synchronization. The extension of INET gPTP to the multi-hop gPTP functionality and then the addition of our gPTP modeling modification to a functional multi-hop INET gPTP are important directions for future research.

In order for readers to understand and compare to the IEEE 802.1AS standard [38], we also provide the official names of these timestamps according to IEEE 802.1AS [38], e.g., t_1 = `PdelayReqEventEgressTimestamp`.

First, the slave clock sends the `Pdelay_Req` message at timestamp t_1 = `PdelayReqEventEgressTimestamp`. The time at which the master clock acknowledges receipt of the `Pdelay_Req` message is designated as timestamp t_2 = `requestReceiptTimestamp`. After the master node [i.e., the node that runs the gPTP protocol with the master clock (oscillator)] receives the `Pdelay_Req`

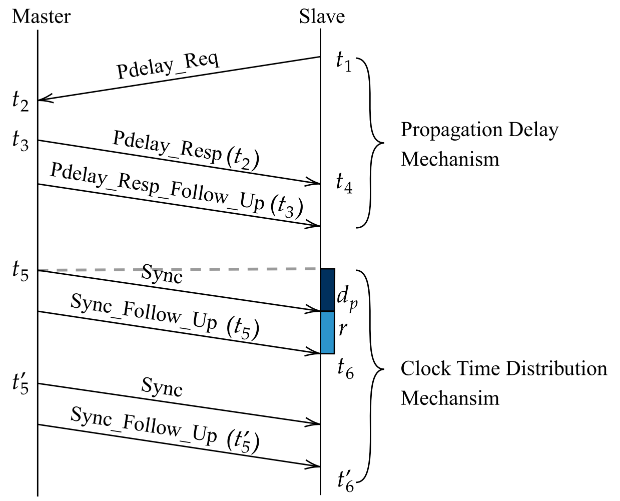


FIGURE 5. Illustration of gPTP time-synchronization according to INET implementation.

message, the master node transmits the `Pdelay_Resp` message and the `Pdelay_Resp_FollowUp` message, which encapsulates timestamp t_3. This specific timestamp is denoted as t_3 = `responseOriginTimestamp`. Lastly, the instant at which the slave clock receives the `Pdelay_Resp` message is precisely marked as timestamp t_4 = `PdelayRespEventgressTimestamp`. Since the time passing in the slave clock (based on the slave clock oscillator) differs from the master clock (with its master clock oscillator), the difference between t_4 and t_1 needs to be multiplied by a compensation value η.

The INET implementation calculates the neighbor rate ratio parameter η in (9), which represents the ratio between the oscillator frequency of the master clock and the oscillator frequency of the slave clock, using consecutive `Sync` and `Sync_FollowUp` messages:

$$\eta = \frac{t'_5 - t_5}{t'_6 - t_6}. \tag{10}$$

As the oscillators of the two clocks differ, the slave clock should multiply its own timestamp by the ratio η to compensate for the difference. For instance, if $t'_5 - t_5$ equals 1 second and $t'_6 - t_6$ equals 1.2 seconds, then η would be 0.83. This indicates that the slave clock has a higher oscillator frequency than the master clock. i.e., the slave clock is faster than the master clock.

In INET gPTP, the `Sync` message distributes the master clock time, and the slave clock adjusts its time when receiving the `Sync_FollowUp` message based on

$$T'_s = T_m + d_p + c + r. \tag{11}$$

The new time of the slave clock, T'_s, is the sum of the master clock `Sync` message sending time T_m (t_5 in Figure 5 as an example), the propagation delay d_p, the correction field c, and the residence time r. In the current INET gPTP, the correction field c is set to zero as only master-slave (no intermediate mode) clock synchronization is considered The correction field will be updated if intermediate nodes are on the gPTP synchronization path (which is a future development for INET). INET's calculation of the correction field is underway and will be included in a forthcoming release. The residence time r equals the time duration between receiving the `Sync` message and receiving the `Sync_Follow_Up` message by the slave node, and the slave can adjust its clock time; for example, from timestamp t_6 to T'_s each time when it receives `Sync_Follow_Up` messages.

During the clock time distribution mechanism, the slave clock updates its clock time and its oscillator frequency compensation following the principles of the common proportional-integral (PI) control of the clock servo [60], [61], [62]. The new oscillator frequency compensation of the slave clock is:

$$F^s_n = (\eta - 1) + (\eta \times F^s_{n-1}), \tag{12}$$

which is derived based on the current frequency compensation F^s_{n-1}. The integer parameter n, $n = 1, 2, \ldots$, represents the n^{th} `Sync_Follow_Up` message received by the slave clock. The initial frequency compensation value is $F^s_0 = 0$. For example, assuming a neighbor rate ratio of 0.83 (slave clock is faster than master clock), $\eta - 1$ would be negative. As a result, the new frequency will be lower, indicating that the slave clock will become slower. Through this recursive process, the slave clock time will gradually approach the master clock.

2) gPTP MODEL MODIFICATION 1

After investigating the Linux-PTP operation of the clock time of the hardware TSN switch, we found that the slave clock time of the hardware TSN switch was gradually updated by changing the oscillator frequency instead of being instantly changed by the gPTP protocol. Based on the Linux-PTP implementation, we modify the INET gPTP by introducing a threshold Γ. If the offset between the slave and master clock is above the threshold Γ, then we enable the reset slave clock function. Otherwise, we disable the functionality of setting the slave clock time directly in the INET program to reduce the gap between the simulation and hardware since the direct setting would cause an abrupt step behavior.

3) gPTP MODEL MODIFICATION 2

Additionally, we introduce two weighting parameters γ and δ in the simulation to adjust the frequency of oscillator compensation; specifically, to allow for faster convergence of the slave clock to the master clock. The parameter γ controls the weight of the offset between the slave clock and the master clock, whereby the offset is represented by the neighbor rate ratio η. Also, δ weighs the contribution of the previous frequency compensation:

$$F^s_n = \gamma \times (\eta - 1) + \delta \times (\eta \times F^s_{n-1}). \tag{13}$$

D. FRAME PREEMPTION (FP) AND CUT-THROUGH (CT) SWITCHING MODIFICATION

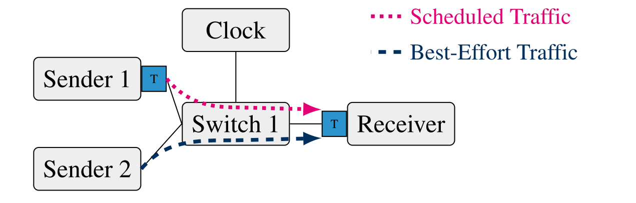
We note that the original INET cannot simulate CT combined with FP due to differences in implementing the MAC and PHY layers. For example, if FP is enabled, then the `EthernetPreemptingPhyLayer` and the `EthernetPreemptingMacLayer` are required. However, to support CT, users need to configure the layers as `EthernetStreamingPhyLayer` and as `EthernetMacLayer`. This module conflict prevents simulating the combined CT and FP in the original INET. The MAC and PHY layers of the original INET need to be modified to enable simulating the combined CT and FP.

Specifically, in our simulation topology, see Figure 6b, we encountered the issue that the CT ceased at Switch 2, which means that Switch 2 is waiting to receive a full packet and then transmits the full packet to the Receiver. INET handles two types of data: signals and streams. A signal can be a packet-start or packet-end marker, whereas a stream represents the bytes on the physical layer (wire). Signals are mainly handled inside network nodes, whereas streams represent the data which is transmitted over the edges between the nodes. Normally, the conversion between signal and stream is conducted by the PHY layer. In the FP case, i.e., when using the `EthernetPreemptingMacLayer`, this conversion is already conducted in the sub-MAC layers inside the `EthernetPreemptingMacLayer`. This violation of the layer concept makes it hard to merge the CT and FP components.

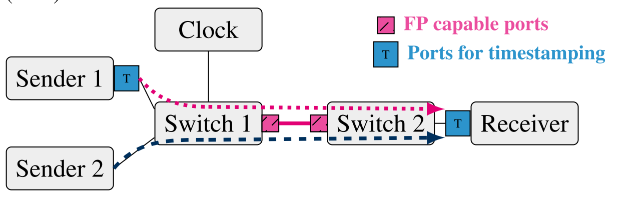
Our modification is based on integrating the PHY and MAC layers from CT and FP and creating new PHY and MAC layers to support the CT and FP features together. We first investigate the MAC layer. In order to support FP, the `EthernetFragmentingMacLayer` is configured by the original INET, whereby the `Ethernet-FragmentingMacLayer` consists of two sublayers, namely `expressMacLayer` and `preemtableMac Layer`. The `expressMacLayer` schedules the higher-priority traffic, whereas the `preemtableMacLayer` preempts the lower-priority traffic. The `expressMacLayer` module in the original INET is inherited from the INET `EthernetStreamingMacLayer` module, while the `preemtableMacLayer` module in the original INET is inherited from the INET `EthernetFragmenting-MacLayer` module.

We configure the ingress port of Switch 2 in Figure 6b to our new MAC layer, namely the `EthernetFragmenting-ThroughMacLayer`, which reuses the original INET `EthernetFragmentingMacLayer` module to coordinate with our new PHY layer.

Moving on to the PHY layer, we discovered that in the default INET CT implementation, there is a module called `cutthroughSource`. This `cutthroughSource` module enables the sending of the CT Ethernet frame in the default INET CT PHY layer, which is not present in the default INET FP PHY layer.



(a) Topology for evaluating the gPTP clock synchronization, the SF and CT forwarding latencies, as well as the Time-Aware Shaper (TAS).



(b) Topology for evaluating Frame Preemption (FP): the second FP-capable Switch 2 is added to preempt low-priority traffic.

FIGURE 6. Network topologies for evaluations: Each link is 10 m long and has a bitrate of $1\,\mathrm{Gbit\,s^{-1}}$. The best effort traffic load is 200% of the link capacity, while an ST frame is transmitted every $1\,\mathrm{ms}$. Frame Preemption (FP) occurs between Switch 1 and Switch 2 in Figure 6b. For the gPTP validation, Clock is the master and Switch 1 is the slave. For the SF vs. CT evaluation, no synchronization is involved (perfect sync is assumed). For the TAS evaluation, Switch 1 is the gPTP master, while Senders 1 and 2 as well as the Receiver are the gPTP slaves. For the FP evaluation, we assume perfect sync.

Therefore, based on the current INET FP PHY layer, we design a new PHY layer called `Ethernet-PreemptingThroughPhyLayer`, wherein we have incorporated the missing `cutthroughSource` module. Furthermore, we encountered an incompatibility between the `cutthroughSource` module used in the default INET CT PHY layer and the receiver module `DestreamingReceiver` used in the default INET FP PHY layer. Therefore, in our new PHY layer, we replaced the INET `DestreamingReceiver` with the INET `StreamThroughReceiver` to address the incompatible issue, see [36] for details.

V. TESTBED AND SIMULATION SETUP

This section first describes the hardware testbed setup, followed by the simulation setup.

A. HARDWARE TESTBED SETUP

The setup of our hardware testbed follows the TSN-FlexTest testbed [28] and employs a Kontron D10 MMT Series switch and a FibroLAN Falcon-RX/G switch. Nodes built on COTS hardware generate high-priority and best-effort traffic. By sending all traffic through a single port of the TSN switch we provoke a bottleneck situation which is the base of our measurements. All nodes are synchronized via gPTP and hardware one-step time stamping is used for first-bit to first-bit end-to-end time measurement. As illustrated in Figure 6, we extend the setup from [28] with an additional TSN switch to enable FP measurements. Both switches support FP, so the second switch acts basically as a transparent preemption terminator.

B. SIMULATION SETUP

1) gPTP SETUP

We investigate the difference in clock synchronization between the master clock and the slave clock by configuring the Clock as the master clock and Switch 1 as the slave clock for the gPTP evaluation, see Fig. 6. The master clock is configured as an ideal clock with an ideal oscillator to precisely match the simulation time. The slave clock is configured as a settable clock, allowing the gPTP protocol to adjust its time. Specifically, we set the oscillator of the slave clock to be a random drift oscillator, which means that the oscillator frequency changes randomly from time to time. The detailed settings of the random drift oscillator are: the `Change Interval` is 5 ms, the `Drift Rate Change` is uniformly distributed between −0.05 ppm to 0.05 ppm, and the maximum drift rate change is set to 0.1 ppm, while the minimum is set to −0.1 ppm. In addition, the oscillator frequency compensation weighting parameters are $\gamma = 1$ and $\delta = 0.7$. The threshold Γ to enable the reset slave clock function is 10 ns.

2) TIME STAMPING

To enable a valid comparison between our hardware testbed measurements and the INET simulations, we patch the same time stamping mechanism used for the hardware testbed into OMNeT++. A timestamper module writes the actual `simtime` into the Ethernet packet payload at a predefined offset. We add the timestamper module to our `UDP-Source` and `UDP-Sink` modules and capture the packets using the pcap recorder. Thus, a packet can be timestamped in different locations in the network. At the packet destination, all timestamps can be extracted from the packet payload. Both, the simulations and the hardware testbed measurements produce pcap files which can be evaluated with the tools described in [28].

VI. MEASUREMENT AND SIMULATION RESULTS

This section compares the INET simulation results with measurements for two state-of-the-art hardware switches from Kontron and FibroLAN. We start with gPTP clock deviation comparisons in Section VI-A and evaluate the improvements for our modified version of the INET gPTP clock adjustment method. Then, in Section VI-B, we evaluate the differences of Store-and-Forward (SF) versus Cut-Through (CT) and the effects of our INET modifications. In Section VI-C, we evaluate the Time-Aware Shaper (TAS). In Section VI-D, we examine combined TSN features, including Frame Preemption (FP).

A. PRECISION TIME PROTOCOL

Figure 7 shows two measurement results for the gPTP time synchronization. In Figure 7a, we show boxplots of the difference in time (clock deviation) between the slave clock and the master clock. A boxplot represent the interquartile range from the first quartile Q_1 to the third quartile Q_3 as a box with the median marked by a horizontal line; the lower (upper) whisker indicates the minimum (maximum) latency that was measured within the range $[Q_1,\ Q_1 - 1.5\{Q_3 - Q_1\}]$ ($[Q_3,\ Q_3 + 1.5\{Q_3 - Q_1\}]$); outliers below (above) this range are indicated by dots. In Figure 7b, we compare gPTP clock adjustment methods by plotting sample paths of the deviation between the slave clock and the master clock.

TABLE 3. Three variations of the INET gPTP library.

Label	Description
Original	Default INET implementation.
Modified 1	Disabled function to immediately set slave clock time to master clock, Sec. IV-C2.
Modified 2	*Modified 1* with modified oscillator calculation according to Eqn. (13), Sec. IV-C3.

1) HARDWARE: KONTRON AND FIBROLAN

In Figure 7a, we examine five different configurations: two hardware-assisted clocks (Kontron and FibroLAN) and three modifications of the INET gPTP implementation. We vary the frequency of `Sync` messages (cf. Figure 5), ranging from 8 to 128 messages per second. The `Pdelay_Req` message is sent once per second. The two hardware switches perform almost identically, with the clock deviations being at most 13 ns and 14 ns for the Kontron and FibroLAN switches, respectively, at the rate of 8 `Sync` messages per second. At the rate of 128 `Sync` messages per second, the maximum clock deviations are reduced to 9 ns for both switches.

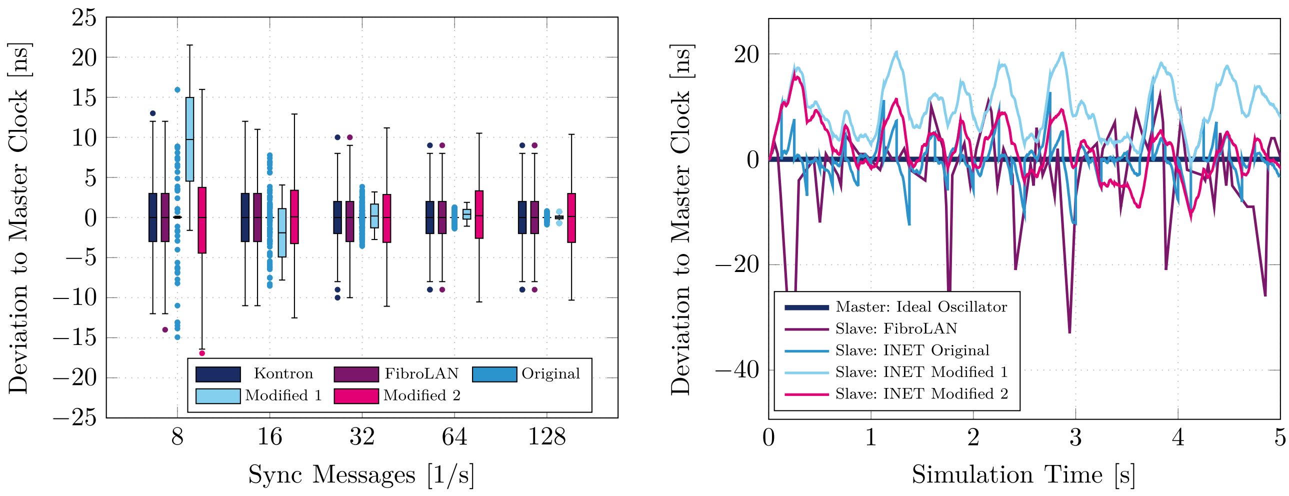
Note: The resolution of the hardware testbed measurements is 1 ns, whereas it is 1 ps for the INET simulation results. Therefore, we show in the following more decimal digits for the simulations than for the hardware measurements.

2) INET SIMULATION

a: ORIGINAL

When evaluating the original INET gPTP implementation, a significant difference to the hardware was noticeable. Therefore, we have modified INET in two different ways, see Section IV-C, in order to reduce the gap. Table 3 summarizes the three INET configurations.

According to the default INET gPTP implementation, the slave clock time is periodically reset to the master clock time based on the design following (11). The procedure is also called a *step jump of the clock time* or *stepping the clock*. This stepping of the clock results in many of the sample points being zero, causing the height of the box to be very small, almost zero, but with many outliers. For instance, for 8 `Sync` messages per second, there are 3928 outliers for 7998 sample points in a 500 s simulation, the mean deviation to the master clock is 0.003 ns while the standard deviation is 4.999 ns compared to 3.940 ns and 3.932 ns for the Kontron and FibroLAN switches, respectively, for the same configuration.



(a) Box plots for deviation between master clock and slave clock for gPTP as a function of sync message frequency.

(b) Comparison of INET gPTP clock adjustment modifications (see Table 3) for 8 "Sync" messages per second.

FIGURE 7. Time synchronization measurements for two hardware TSN switches and three simulation implementations.

b: MODIFICATION 1

To overcome this gap, we disable the function of *stepping the clock time and setting it immediately to the master clock* and only use the original oscillator frequency compensation function to change the time gradually. We call this implementation "Modified 1". Figure 7 indicates that when the rate of the `Sync` messages is low (8 and 16), the slave clock cannot perfectly synchronize to the master clock. For instance, in case of 8 `Sync` messages per second, we observe a positive shift of the clock with a mean clock deviation of 9.852 ns. We notice a less intense, but negative shift for 16 `Sync` messages. This means that for low synchronization rates, the reaction of the PI controller of the clock servo [60], [61], [62] in the INET gPTP Modification 1 is not strong enough to compensate for the oscillator drift and delay variation. Therefore, we decided to tweak the servo for a faster response function resulting in our gPTP Modification 2 (Sec. IV-C3).

With more `Sync` messages per second, the deviations to the master clock stabilize, i.e., they oscillate around the desired mark of 0 ns. However, with the upper and lower whiskers of 0.686 ns and −0.661 ns for 128 `Sync` messages for Modified 1, the deviations are significantly lower than with real hardware.

c: MODIFICATION 2

The right-most boxplot for each frequency of sync messages in Figure 7 shows the clock deviations of our second modified INET version (labeled as Modified 2). With Modified 2, the behavior of the simulation is similar to the hardware testbed. Especially at the rate of 128 `Sync` messages per second, the span of the box and the whiskers are close to the hardware results. For example, the upper quartile and upper whisker of Modified 2 for 128 `Sync` messages per second are 2.987 ns and 10.377 ns. The differences of the upper quartile and upper whisker to the FibroLAN switch in the same scenario are 0.987 ns and 2.377 ns, respectively.

3) SUMMARY OF BOX PLOT COMPARISON

Comparing all results together, we observe that more gPTP `Sync` messages per second can reduce the deviation of the slave clock from the master clock. However, the absolute reduction of the deviation when increasing the sync message frequency from 8 to 128 `Sync` messages per second is relatively small. The default 8 `Sync` messages from the IEEE standard are generally sufficient, and we cannot observe clear advantages for higher frequent `Sync` messages in our measurements.

4) SAMPLE PATH COMPARISON

In order to further examine the effects of our gPTP library modifications in INET, we plot in Figure 7b the sample paths of the deviation of the slave clock from the master clock over 5 s for 8 `Sync` messages per second. The master clock exists in both the hardware testbed and simulation environment. As reference, the dark blue, horizontal line depicts the *ideal* master time, that the slave clocks try to synchronize to (in contrast to the universal time). The plot shows a 5 s snapshot out the middle of a 500 s evaluation to highlight the general behavior of the different clock adjustment methods.

The sample points are generated by the OMNeT++ simulation environment. If the time changes due to a re-set or step of the clock in the slave clock module, the slave clock module triggers a sample; therefore the sampling rate differs between the three INET gPTP library versions; see Table 3. However, this improves the resolution of the sampled data since, in the hardware measurements, we can only observe the system when the hardware slave clock receives the `Sync` message.

In theory, there is clock drift due to the settable slave clock configuration, see Section V-B1. We can see from Figure 7b, that the slave clocks immediately start to deviate from the master clock. The internal oscillator must be adjusted from the outside to run faster or slower; or in case of too much deviation to be stepped (as explained in Section IV-C1).

The original INET implementation exhibits frequent *step jumps* back to the master clock as evidenced by the frequent abrupt vertical line segments back to the master clock horizontal line in Figure 7b. This behavior is in contrast to the behavior of the actual gPTP implementation in real FibroLAN switch hardware, which reduces the deviation gradually (via sloped line segments). Our Modification 1 disabled this stepping, as can be clearly observed from Figure 7b. However, as a side effect, the oscillator is not adjusted in the correct manner, which causes a more or less constant offset from the master clock (as also observed in Figure 7a). We observe from Figure 7a that our Modification 2 oscillates around the master reference time while, similar to the real hardware, reducing the deviation gradually.

B. FORWARDING LATENCY

Figure 8 compares the mean one-way first-bit to first-bit (1 - 1 bit) delay of SF and CT for the network topology in Figure 6 (from Sender 1 to the Receiver). Figure 8 also compares the INET simulations (without and with the modifications of Sec. IV-B2) with the hardware testbed measurements.

1) LATENCY PLATEAU

In general, we expect a linear 1-1 bit delay increase for SF with increasing frame size, as the entire packet needs to be received by the switch before it can be processed. With CT, the plot should also have a linear increase for small frames. However, there should be a point where the latency reaches a plateau, as discussed in Sec. IV-B2. This turning point, i.e., the frame size threshold F_T, is implementation-specific. The linear 1-1 bit delay increases with increasing frame size for the entire SF line and for the CT line segment below the threshold F_T correspond to one frame transmission delay d_t, see Section IV-B1c.

We briefly note that the end-to-end (E2E) delay from the beginning of the first byte transmission at the sender to the complete reception of the last byte at the receiver, which we do not include in the plots to avoid clutter, would behave as follows. The SF E2E delay increases linearly with increasing frame size with double the slope of the corresponding SF 1-1 bit delay since the SF E2E latency includes two frame transmission delays d_t, see Fig. 3a and Eqn. (3). The CT E2E delay has the same doubled slope for the line segment below the threshold F_T; at the threshold F_T, the slope halves and the linear increase continues with a slope corresponding to one frame transmission delay d_t, see Fig. 3b and Eqn. (4).

The vendor Kontron informed us in discussions, that the forwarding latency of the switch varies over the ports; specifically, ports 5 and 6 (denoted "5/6" for brevity) should have a lower latency. A reason was not disclosed. In a first measurement series, we used ports 1 and 3 (1/3), and in Figure 8, we show results for both port combinations.

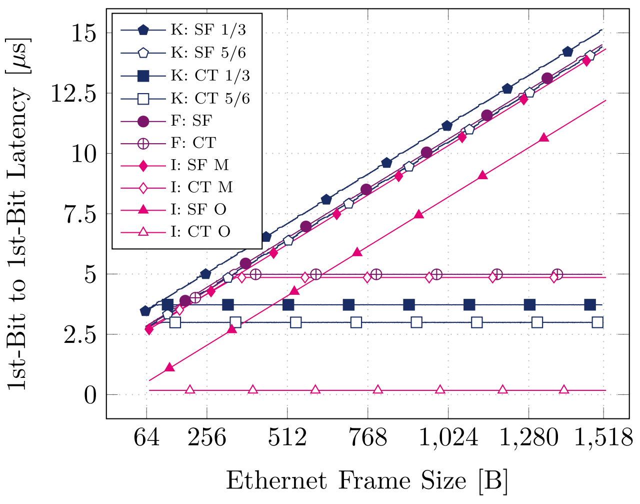


FIGURE 8. Mean first-bit to first-bit delay for Store-and-Forward (SF) and Cut-Through (CT) as a function of Ethernet frame size in the link layer. Legend: *K*: Kontron, *F*: FibroLAN, *I*: INET simulation, with *O*: Original and *M*: Modified. 1/3 and 5/6 for Kontron switch indicates the used Ethernet ports.

2) HARDWARE: KONTRON VS. FIBROLAN

We observe for the two hardware switches the expected linearly increasing 1-1 bit delay with increasing frame size for SF; respectively, the linear latency increase and then the latency plateau for CT. For SF, the Kontron and FibroLAN switches have similar latencies. Especially, if ports 5/6 are used on the Kontron switch, the latencies are almost identical. For the Kontron switch ports 1/3, there are little periodic "bumps", which are an example of implementation-specific characteristics and a demonstration of the high precision of the TSN-FlexTest testbed. For instance, for a frame size of 1514 B, we measured mean Kontron SF forwarding latencies of 15.13 µs for ports 1/3 and 14.43 µs for ports 5/6. Similar results were obtained in the CT mode: For a 1514 B frame, the mean Kontron CT latencies are 3.72 µs for ports 1/3 and 3.00 µs for ports 5/6. For the FibroLAN switch, for a frame size of 1514 B, the mean latencies are 14.51 µs for SF and 4.99 µs for CT.

The mean SF 1-1 bit delay of the FibroLAN switch is very similar to ports 5/6 of the Kontron switch. However, there is a difference in the CT mode: the CT latency of the FibroLAN switch (2.85 µs for 64 B) is higher than the Kontron ports 5/6 CT latency (2.74 µs for 64 B); however, for the small 64 B frames, the Kontron ports 1/3 CT latency is 3.47 µs, i.e., longer than the corresponding FibroLAN latency. For the Kontron switch, the frame size threshold F_T for entering the CT plateau is approx. $F_T^K \approx 113$ B for port 5/6, and 93 B for port 1/3, while for the FibroLAN switch $F_T^F \approx 340$ B. A reason for both observations can be again implementation-specific internal processing pipelines, which

TABLE 4. Parameter values of modified INET SF model, see (7), and CT model, see (8), measured with TSN-FlexTest testbed for two TSN switches (and two port combinations for Kontron switch).

Switch	β_{sf} [ns]	α_{ct} [ns]	β_{ct} [ns]	F_T [B]
FibroLAN	2336.57	7.50	2130.43	340
Kontron, ports 1/3	2981.47	8.79	2907.58	93
Kontron, ports 5/6	2249.07	5.16	2409.77	113

we cannot examine from the outside. Moreover, this means that frame sizes below these frame size thresholds F_T will not benefit from CT.

3) MODIFIED INET SIMULATION

Figure 8 also shows the improvement of our INET modifications. The original INET implementation (see triangular markers) gives the following mean latencies for a 1514 B frame: 12.18 µs for SF and 0.18 µs for CT. However, these latencies are not realistic (they are too small) compared to the hardware measurements. Also, for the CT mode, the original INET simulation entirely misses the characteristic hardware CT latency behavior consisting of linear increase, frame size threshold, and then latency plateau.

The original INET SF model only considers the packet transmission delay d_t, without considering the switch processing delay $d_l + d_f^{\mathrm{frame}}$, see Eqn. (2), resulting in a gap between the original INET simulations and the hardware measurements. Therefore, we modified the INET SF models by adding a delay offset β_{sf} that models the switch processing delay, see Eqns. (6) and (7). The β_{sf} values of the FibroLAN and Kontron switches as obtained from our measurements are given in Table 4. Figure 8 indicates that this INET SF model modification vastly reduces the gap between the simulations and the hardware measurements for the SF 1-1 bit latency.

For CT, we modified the INET code (see Sec. IV-B2) to implement our CT delay model. Based on our measurements, we configured the model parameters α_{ct}, β_{ct}, and F_T see Eqn. (8), as summarized in Table 4. The results obtained with the modified INET (see diamond markers) are very close to the latencies of the hardware switches, whereby the FibroLAN switch measurements essentially coincide with the modified INET simulations for CT switching. For a 1514 B frame, the modified INET gives an SF latency of 14.24 µs and a CT latency of 4.86 µs.

C. TIME-AWARE SHAPER (TAS)

We present the results of the TAS evaluations in Figure 9 as Complementary Cumulative Distribution Function (CCDF). On the x-axis, the figure shows the one-way first-bit to first-bit latency, and the y-axis represents the proportion of delay samples with the respective latency. The intention of using a TAS is to reduce variations and thus to increase determinism. A characteristic for this deterministic behavior is a straight vertical CCDF line.

TABLE 5. Gate Control List (GCL) configuration for Time-Aware Shaper (TAS) measurements, with: $\tau_1 = 500$ µs, $\tau_2 = 485$ µs, $\tau_3 = 15$ µs. Open gates are marked with a blue cross and the guard band with a gray *G*.

Priority \ Time	τ_1	τ_2	τ_3
Scheduled Traffic (ST)	X		G
Best-Effort Traffic (BE)		X	G

1) TAS CONFIGURATION

For both hardware testbed and INET simulation, we use the same GCL configuration provided in Table 5. First, the gate is only opened for ST for $\tau_1 = 500$ µs, followed by the gate being exclusively open for Best-Effort Traffic (BE) for $\tau_2 = 485$ µs. To protect the ST in the next cycle from the BE in the preceding cycle, there is a guard band of $\tau_3 = 15$ µs during which no traffic is allowed to be transmitted (see Figure 1). The 15 µs roughly correspond to one Maximum Transmission Unit (MTU) of 1522 B. Note: We use ports 1 and 3 of the Kontron switch for the TAS measurements (see Section VI-B for more explanation).

Overall, Figure 9 demonstrates that applying the TAS can achieve very small latency variations. We compare the TAS of the Kontron (K) and FibroLAN (F) switches for SF and CT for both small (128 B) and large (1518 B) Ethernet frames. Furthermore, the figure contains the results of the INET TAS implementation, labeled as I. In this evaluation, we employ our modified gPTP (see Sec. IV-C2) and modified CT simulation model (see Sec. IV-B2) with the original INET TAS implementation.

2) HARDWARE: FIBROLAN AND KONTRON

The FibroLAN switch has a lower latency compared to the Kontron switch for SF in Figure 9 (see filled markers), which is consistent with the mean latencies without TAS in Section VI-B. For SF and 128 B, we observe from Figure 9 and Table 6 a maximum latency of 4.09 µs for the Kontron switch and 3.41 µs for the FibroLAN switch.

For CT, the results are again consistent with Section VI-B: the FibroLAN switch has a lower latency for small frames, but a higher latency for larger frames compared to the Kontron switch ports 1/3. For CT and 128 B (resp. 1518 B) frames, the maximum Kontron ports 1/3 latency is 3.81 µs (resp. 3.81 µs) while the maximum FibroLAN latency is 3.41 µs (resp. 5.03 µs).

3) INET SIMULATION

The simulation outcomes demonstrate the effectiveness of the TAS in achieving small jitter, aligning closely with the hardware results. Specifically, in the SF 128 B scenario, the simulation's maximum of 3.72 µs is very close to the FibroLAN switch measurement, differing only by 0.31 µs. For CT and 128 B, the offset of the maximum latency between

TABLE 6. Summary statistics of TAS first-bit to first-bit latency evaluation for SF and CT switching of small 128 byte frames and large 1518 byte frames.

Scenario	Kontron HW	FibroLAN HW	INET Simul.
SF 128 B	Max: 4.09 µs, Min: 3.92 µs, Stdev: 49.53 ns	Max: 3.41 µs, Min: 3.30 µs, Stdev: 30.55 ns	Max: 3.72 µs, Min: 3.65 µs, Stdev: 14.34 ns
CT 128 B	Max: 3.81 µs, Min: 3.64 µs, Stdev: 49.23 ns	Max: 3.41 µs, Min: 3.30 µs, Stdev: 31.08 ns	Max: 3.77 µs, Min: 3.74 µs, Stdev: 7.99 ns
SF 1518 B	Max: 15.23 µs, Min: 15.06 µs, Stdev: 48.09 ns	Max: 14.59 µs, Min: 14.49 µs, Stdev: 31.03 ns	Max: 14.31 µs, Min: 14.24 µs, Stdev: 13.90 ns
CT 1518 B	Max: 3.81 µs, Min: 3.64 µs, Stdev: 48.64 ns	Max: 5.03 µs, Min: 4.93 µs, Stdev: 31.06 ns	Max: 4.92 µs, Min: 4.89 µs, Stdev: 8.27 ns

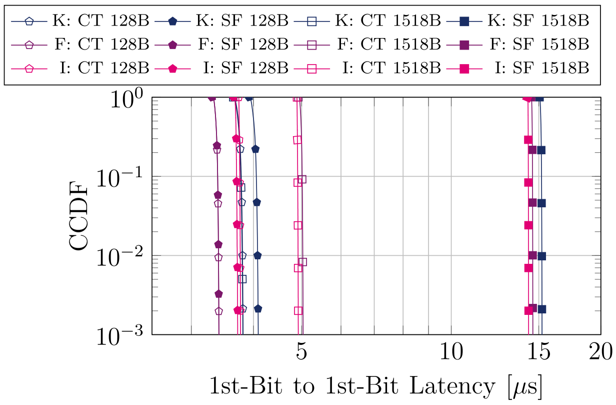


FIGURE 9. Time-Aware Shaper (TAS) first-bit to first-bit forwarding latency plotted as Complementary Cumulative Distribution Function (CCDF). Legend: *Device: Switching technology, Ethernet frame size* with *K*: Kontron, *F*: FibroLAN, *I*: INET simulation.

the Kontron switch and the simulation is only 0.04 µs. Similarly, SF 1518 B reveals a minuscule difference, with the simulation's maximum at 14.31 µs in contrast to FibroLAN's 14.59 µs. Moreover, the CT 1518 B simulation results with a maximum of 4.92 µs closely mirror the FibroLAN switch measurements. In general, the TAS in the INET simulation performs close to the hardware testbed results, reflecting the real hardware behavior.

D. FRAME PREEMPTION (FP)

We investigate FP exclusively and in combination with other TSN features (different switching modes and TAS) via boxplots in Figure 10. We measure the one-way first-bit to first-bit latency for the topology in Figure 6b. Specifically, we utilize the FibroLAN switch as Switch 1 and the Kontron switch as Switch 2. Table 7 shows the legend of Figure 10. While previous subsections have focused on individual TSN features in isolation, we now highlight the advantages for increasing determinism when features are used together. Generally, high-priority ST is always impaired by BE, and we examine how combining FP with other TSN features reduces the ST latency and latency variation.

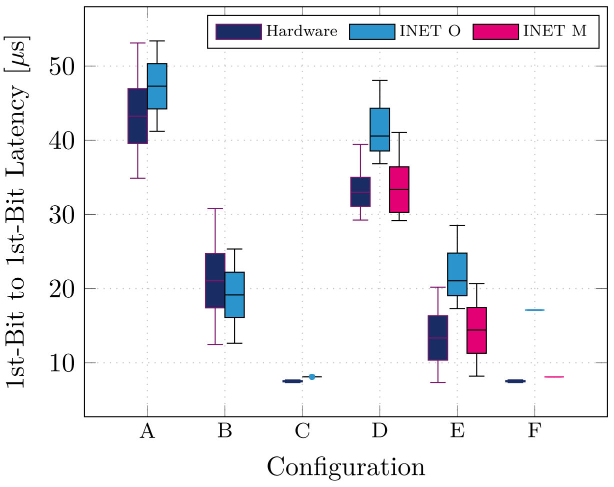


FIGURE 10. Frame Preemption (FP) in isolation and in combination with other TSN features; Explanation of x-axis legend in Table 7, frame size 1518 B.

We start with configurations A and B, that use only Strict Priority Queuing (SPQ) to distinguish the streams. This means that frames of different priorities are handled by the switch in different queues, but there is no advanced mechanism, such as TAS, to protect them from each other. As a result, the ST may compete with BE, causing higher delays and variation for ST in comparison to subsequent configurations. From Figure 10, the benefit of CT is evident when comparing A (SF) and B (CT). The median INET latencies are 47.30 µs (which is 4.09 µs higher than the hardware median latency) for configuration A (SF) and 19.15 µs (1.91 µs lower than hardware) for configuration B (CT).

By implementing TAS in configuration C, the latency and jitter are significantly reduced. The median INET latency of box C is 8.10 µs, which is 0.60 µs higher than the hardware median latency. Compared to configurations A and B, where the differences between the upper and lower latency whiskers exceed 10 µs, both whiskers and the median coincide for box C; i.e., TAS can achieve essentially zero jitter.

Configurations D, E, and F use FP (see Section II-B4), which can reduce the latency compared to pure SF and CT. The median original INET latencies are 40.57 µs for configuration D and 21.05 µs for configuration E. Compared to the hardware SPQ measurements, we observe a decrease in the hardware latency for both SF and CT as well as a decrease of the hardware latency variation (smaller boxes and whiskers).

We observed that for configurations D, E, and F with FP, the latency values from the original INET simulation are about 10 µs higher than the corresponding hardware testbed measurements. More specifically, the medians of original INET for the configurations D, E, and F are 7.58 µs, 7.70 µs, and 9.60 µs higher than the medians of the corresponding hardware measurements. The higher simulation latencies can be attributed to the original INET implementation, where

TABLE 7. Legend details for Figure 10. In the measurement of the combined features, we disable the gPTP protocol and assume perfect synchronization because in the current INET release (4.5.2), FP is not compatible with gPTP.

Configuration	SPQ	Switching	TAS	FP
A	✓	SF	✗	✗
B	✓	CT	✗	✗
C	✓	CT	✓	✗
D	✓	SF	✗	✓
E	✓	CT	✗	✓
F	✓	CT	✓	✓

the CT at switch 2 in Figure 6b is not compatible with FP (see Section IV-D), increasing the first-bit to first-bit delay. With our INET modification in Section IV-D, the median latencies of the modified INET versions D, E, and F are 33.37 µs, 14.42 µs, and 8.10 µs, respectively. The medians of the modified INET simulation are only 0.38 µs, 1.07 µs, and 0.58 µs, respectively, higher than the corresponding medians of the hardware measurements.

VII. CONCLUSION

A. SUMMARY OF STUDY CONTRIBUTIONS

In recent years, TSN has become an important networking technology for enabling near-deterministic communication within the data link layer (layer two) of networks, attracting significant attention. A significant amount of research is currently centered around evaluating the performance of TSN networks under different scenarios and configurations, and developing various mechanisms to optimize the TSN network performance. Conducting hardware testbed evaluations can be challenging due to limited hardware availability, test environment setup complexities, and the high cost of TSN switches, especially in large-scale scenarios. As a result, research in this domain heavily relies on simulation as an evaluation method to refine and improve TSN tools and standards. Recent studies commonly employ the combination of OMNeT++ and the INET framework for simulating TSN networks.

This study rigorously evaluated the accuracy of INET TSN simulations by comparing simulation results with measurements on our real-world TSN-FlexTest testbed [28], equipped with commercially available COTS TSN switches and interfaces. When disparities arose, we modified INET to more closely align with the real hardware behaviors. Our evaluation focused on specific TSN features in INET that were available in August 2023, namely time synchronization, CT, TAS, and FP. Our investigation initially delved into clock deviation among diverse network devices for various gPTP configurations, exposing disparities between simulation and testbed outcomes. To address these disparities, we enhanced the INET framework to simulate the gPTP of real hardware better. Our measurements validated that the modified INET version closely matches gPTP results from hardware-based measurements.

Furthermore, we observed disparities between simulations with the original INET and hardware measurements of first-bit to first-bit latency for both SF and CT switching. We conducted a mathematical analysis, identified the source of the difference, and modified INET accordingly. Simulations with our modified INET framework closely mirror the SF and CT forwarding latencies of hardware measurements. We also modified the FP MAC and PHY layers to support FP in conjunction with CT. Our modifications, which we make publicly available [36], aim to provide researchers with a more precise tool for evaluating TSN networks.

B. FUTURE RESEARCH DIRECTIONS

The evaluation and enhancement of the INET TSN simulation accuracy in this paper can form the foundation for several important future research directions on advancing the capabilities of INET TSN simulations. First, once the gPTP protocol in the current INET release 4.5.2, see Section IV-C1, has been extended to support gPTP over multiple hops, then follow-up research should evaluate the accuracy of end-to-end time synchronization for large-scale multi-hop networks.

This study has focused on four core TSN features, namely gPTP time synchronization, CT packet switching, TAS traffic shaping, and FP, which support a wide range of TSN use cases [29]. Future research should consider other TSN features, such as the Credit-Based Shaper (CBS, IEEE 802.1Qav), the Asynchronous Traffic Shaper (ATS, IEEE 802.1Qcr), Frame Replication and Elimination for Reliability (FRER, IEEE 802.1CB), and Per-Stream Filtering and Policing (PSFP, 802.1Qci, also referred to as Ingress Policing) [29], [63]. Since OMNeT++ is a modular simulator these features can directly use our modified gPTP and CT INET models. In particular, similar to TAS jointly operating with CT in our study, future studies can integrate CBS and ATS with CT. PSFP is a higher-layer management protocol, which can readily employ our lower-layer modifications of gPTP and CT. On the other hand, since CT is not compatible with the original INET FP, requiring the modifications in Section IV-D for integrating CT with FP, we anticipate that similar modifications will be required for integrating CT with FRER; specifically, when the duplicated CT streams reach the destination and need to be eliminated.

Since 5G integration with TSN is a promising technique for a wide range of applications [17], [18], [19], e.g., for smart factories, investigating the end-to-end latency of integrated 5G-TSN networks is important. Such investigations can employ combinations of our modified INET with open-source 5G projects that are based on OMNeT++, e.g., [17], [64], and [65]. Moreover, expanding the TSN-FlexTest testbed [28] to incorporate 5G capability is imperative to validate the simulation of integrated 5G-TSN networks. More broadly, the capabilities of simulators and testbeds of integrated TSN and wireless networks should be continuously expanded to accommodate emerging wireless communication paradigms for very-high-speed 5G systems and for 6G systems, such as visible light communication [66], [67], [68], [69], [70], [71].

The INET simulation framework releases to date lack control plane TSN components within the fully centralized TSN architecture. Thus, it is urgent to add specific control plane functionalities, such as Centralized User Configuration (CUC) and Centralized Network Configuration (CNC) [72], to the INET simulation framework and to validate these additions using commercial hardware.

ACKNOWLEDGMENT

The authors would like to thank the Open-Source Community in general for sharing their sophisticated work and in particular the OMNeT++ and INET developer teams, who supported this research.

REFERENCES

[1] Z. Feng, C. Wu, Q. Deng, Y. Lin, S. Gao, and Z. Gu, "On the scheduling of fault-tolerant time-sensitive networking with IEEE 802.1CB," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 43, no. 6, pp. 1715–1728, Jun. 2024.

[2] T. Fiori, F. G. Lavacca, F. Valente, and V. Eramo, "Proposal and investigation of a lite time sensitive networking solution for the support of real time services in space launcher networks," *IEEE Access*, vol. 12, pp. 10664–10680, 2024.

[3] D. L. Gomez, G. A. Montoya, C. Lozano-Garzon, and Y. Donoso, "Strategies for assuring low latency, scalability and interoperability in edge computing and TSN networks for critical IIoT services," *IEEE Access*, vol. 11, pp. 42546–42577, 2023.

[4] J. Jiang, Y. Li, X. Zhang, M. Yu, C. D. Lee, and S. H. Hong, "Assessing the traffic scheduling method for time-sensitive networking (TSN) by practical implementation," *J. Ind. Inf. Integr.*, vol. 33, Jun. 2023, Art. no. 100464.

[5] G. Wang, T. Zhang, C. Xue, J. Wang, M. Nixon, and S. Han, "Time-sensitive networking (TSN) for industrial automation: A survey," 2023, *arXiv:2306.03691*.

[6] S. Senk, M. Ulbricht, I. Tsokalo, J. Rischke, S.-C. Li, S. Speidel, G. T. Nguyen, P. Seeling, and F. H. P. Fitzek, "Healing hands: The tactile Internet in future tele-healthcare," *Sensors*, vol. 22, no. 4, Feb. 2022, Art. no. 1404.

[7] I. Turcanu and C. Sommer, "Poster: Potentials of mixing TSN wired networks and best-effort wireless networks for V2X," in *Proc. IEEE Veh. Netw. Conf. (VNC)*, Nov. 2021, pp. 135–136.

[8] Y. Xu and J. Huang, "A survey on time-sensitive networking standards and applications for intelligent driving," *Processes*, vol. 11, no. 7, p. 2211, Jul. 2023.

[9] F. H. P. Fitzek, S.-C. Li, S. Speidel, T. Strufe, M. Simsek, and M. Reisslein, *Tactile Internet: With Human-in-the-Loop*. Cambridge, MA, USA: Academic, 2021.

[10] M. Z. Islam, R. Ali, A. Haider, and H. S. Kim, "QoS provisioning: Key drivers and enablers toward the tactile Internet in beyond 5G era," *IEEE Access*, vol. 10, pp. 85720–85754, 2022.

[11] D. Rico, M.-D.-M. Gallardo, and P. Merino, "Verification of a multi-connectivity protocol for tactile Internet applications," *Comput. Commun.*, vol. 212, pp. 390–406, Dec. 2023.

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

[13] B. O. Akram, N. K. Noordin, F. Hashim, M. F. A. Rasid, M. I. Salman, and A. M. Abdulghani, "Joint scheduling and routing optimization for deterministic hybrid traffic in time-sensitive networks using constraint programming," *IEEE Access*, vol. 11, pp. 142764–142779, 2023.

[14] J. Walrand, "A concise tutorial on traffic shaping and scheduling in time-sensitive networks," *IEEE Commun. Surveys Tuts.*, vol. 25, no. 3, pp. 1941–1953, 3rd Quart., 2023.

[15] S. Oechsle, F. Frick, A. Lechler, and A. Verl, "A modular configuration and management framework for distributed real-time applications based on converged networks using TSN," *Proc. CIRP*, vol. 118, pp. 38–43, Jan. 2023.

[16] S. Bhattacharjee, K. Katsalis, O. Arouk, R. Schmidt, T. Wang, X. An, T. Bauschert, and N. Nikaein, "Network slicing for TSN-based transport networks," *IEEE Access*, vol. 9, pp. 62788–62809, 2021.

[17] R. Debnath, M. S. Akinci, D. Ajith, and S. Steinhorst, "5GTQ: QoS-aware 5G-TSN simulation framework," in *Proc. IEEE 98th Veh. Technol. Conf. (VTC-Fall)*, Oct. 2023, pp. 1–7.

[18] F. Luque-Schempp, L. Panizo, M.-D.-M. Gallardo, and P. Merino, "AutomAdapt: Zero touch configuration of 5G QoS flows extended for time-sensitive networking," *IEEE Access*, vol. 11, pp. 82960–82977, 2023.

[19] S. Mostafavi, M. Tillner, G. P. Sharma, and J. Gross, "EDAF: An end-to-end delay analytics framework for 5G-and-beyond networks," in *Proc. 11th Int. Workshop Comput. Netw. Exp. Res. Using Testbeds (CNERT)*, 2024, pp. 1–7.

[20] S. Senk, H. K. Nazari, H.-H. Liu, G. T. Nguyen, and F. H. P. Fitzek, "Open-source testbeds for integrating time-sensitive networking with 5G and beyond," in *Proc. IEEE 20th Consum. Commun. Netw. Conf. (CCNC)*, Jan. 2023, pp. 1–7.

[21] G. P. Sharma, D. Patel, J. Sachs, M. De Andrade, J. Farkas, J. Harmatos, B. Varga, H.-P. Bernhard, R. Muzaffar, M. Ahmed, F. Dürr, D. Bruckner, E. M. De Oca, D. Houatra, H. Zhang, and J. Gross, "Toward deterministic communications in 6G networks: State of the art, open challenges and the way forward," *IEEE Access*, vol. 11, pp. 106898–106923, 2023.

[22] X. Tang, Q. Fu, X. Hu, Y. Zhu, S. Wu, J. Wang, and W. Li, "Simulation analysis on 5G-TSN scheduling algorithms based on OMNeT++," in *Proc. 8th Int. Conf. Comput. Commun. Syst. (ICCCS)*, Apr. 2023, pp. 463–468.

[23] J. Peeck and R. Ernst, "Improving worst-case TSN communication times of large sensor data samples by exploiting synchronization," *ACM Trans. Embedded Comput. Syst.*, vol. 22, no. 5, pp. 1–25, Oct. 2023.

[24] X. Zheng, Y. Liu, S. Zhan, Y. Xin, and Y. Wang, "A novel low-latency scheduling approach of TSN for multi-link rate networking," *Comput. Netw.*, vol. 240, Feb. 2024, Art. no. 110184.

[25] A. B. Muslim, R. Tönjes, and T. Bauschert, "Synchronizing TSN devices via 802.1AS over 5G networks," *Electronics*, vol. 13, no. 4, Feb. 2024, Art. no. 768.

[26] T. Stüber, L. Osswald, S. Lindner, and M. Menth, "A survey of scheduling algorithms for the time-aware shaper in time-sensitive networking (TSN)," *IEEE Access*, vol. 11, pp. 61192–61233, 2023.

[27] C. Xue, T. Zhang, Y. Zhou, M. Nixon, A. Loveless, and S. Han, "Real-time scheduling for 802.1Qbv time-sensitive networking (TSN): A systematic review and experimental study," 2023, *arXiv:2305.16772*.

[28] M. Ulbricht, S. Senk, H. K. Nazari, H.-H. Liu, M. Reisslein, G. T. Nguyen, and F. H. P. Fitzek, "TSN-FlexTest: Flexible TSN measurement testbed," *IEEE Trans. Netw. Service Manage.*, vol. 21, no. 2, pp. 1387–1402, Apr. 2024.

[29] Y. Seol, D. Hyeon, J. Min, M. Kim, and J. Paek, "Timely survey of time-sensitive networking: Past and future directions," *IEEE Access*, vol. 9, pp. 142506–142527, 2021.

[30] *OMNeT++ Discrete Event Simulator*. Accessed: Jun. 5, 2024. [Online]. Available: https://omnetpp.org/

[31] L. Meszaros, A. Varga, and M. Kirsche, "INET framework," in *Recent Advances in Network Simulation: The OMNeT++ Environment and Its Ecosystem*, A. Virdis and M. Kirsche, Eds. Cham, Switzerland: Springer, 2019, pp. 55–106.

[32] T. Steinbach, H. Dieumo Kenfack, F. Korf, and T. Schmidt, "An extension of the OMNeT++ INET framework for simulating real-time Ethernet with high accuracy," in *Proc. 4th Int. ICST Conf. Simulation Tools Techn.*, 2011, pp. 375–382.

[33] F. Rezabek, M. Bosk, T. Paul, K. Holzinger, S. Gallenmüller, A. Gonzalez, A. Kane, F. Fons, Z. Haigang, G. Carle, and J. Ott, "EnGINE: Developing a flexible research infrastructure for reliable and scalable intra-vehicular TSN networks," in *Proc. 17th Int. Conf. Netw. Service Manage. (CNSM)*, Oct. 2021, pp. 530–536.

[34] M. Bosk, F. Rezabek, K. Holzinger, A. G. Marino, A. A. Kane, F. Fons, J. Ott, and G. Carle, "Methodology and infrastructure for TSN-based reproducible network experiments," *IEEE Access*, vol. 10, pp. 109203–109239, 2022.

[35] M. Bosk, F. Rezabek, J. Abel, K. Holzinger, M. Helm, G. Carle, and J. Ott, "Simulation and practice: A hybrid experimentation platform for TSN," in *Proc. IFIP Netw. Conf. (IFIP Networking)*, Jun. 2023, pp. 1–9.

[36] Comnets TSN Team. (2020). *OMNeT CI Testbed*. [Online]. Available: https://github.com/5GCampus/omnet-ci

[37] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 8th ed. London, U.K.: Pearson, 2020.

[38] *IEEE Standard for Local and Metropolitan Area Networks—Timing and Synchronization for Time-Sensitive Applications*, IEEE Standard 802.1AS-2020, 2020, pp. 1–421.

[39] (2023). *Cut-Through Forwarding Bridges and Bridged Networks*. [Online]. Available: https://1.ieee802.org/tsn/802-1du/

[40] *IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks—Amendment 25: Enhancements for Scheduled Traffic*, IEEE Standard 802.1Qbv-2015, 2015.

[41] A. Arestova, K.-S.-J. Hielscher, and R. German, "Optimization of bandwidth utilization and gate control list configuration in 802.1Qbv networks," *IEEE Access*, vol. 11, pp. 115076–115090, 2023.

[42] S. Sudhakaran, C. Hall, D. Cavalcanti, A. Morato, C. Zunino, and F. Tramarin, "Measurement method for end-to-end time synchronization of wired and wireless TSN," in *Proc. IEEE Int. Instrum. Meas. Technol. Conf. (I2MTC)*, May 2023, pp. 1–6.

[43] *IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks—Amendment 26: Frame Preemption*, IEEE Standard 802.1Qbu-2016, 2016, pp. 1–52.

[44] *IEEE Standard for Ethernet Amendment 5: Specification and Management Parameters for Interspersing Express Traffic*, IEEE Standard 802.3br-2016, 2016, pp. 1–58.

[45] M. Ashjaei, M. Sjödin, and S. Mubeen, "A novel frame preemption model in TSN networks," *J. Syst. Archit.*, vol. 116, Jun. 2021, Art. no. 102037.

[46] *IEEE Standard for Local and Metropolitan Area Network—Bridges and Bridged Networks*, IEEE Standard 802.1Q-2014, 2018, pp. 1–1993.

[47] *CoRE4INET*. Accessed: Jun. 5, 2024. [Online]. Available: https://github.com/CoRE-RG/CoRE4INET

[48] J. Falk, D. Hellmanns, B. Carabelli, N. Nayak, F. Dürr, S. Kehrer, and K. Rothermel, "NeSTiNg: Simulating IEEE time-sensitive networking (TSN) in OMNeT++," in *Proc. Int. Conf. Networked Syst. (NetSys)*, Mar. 2019, pp. 1–8.

[49] P. Heise, F. Geyer, and R. Obermaisser, "TSimNet: An industrial time sensitive networking simulation framework based on OMNeT++," in *Proc. 8th IFIP Int. Conf. New Technol., Mobility Secur. (NTMS)*, Nov. 2016, pp. 1–5.

[50] Y. Huang, S. Wang, X. Zhang, T. Huang, and Y. Liu, "Flexible cyclic queuing and forwarding for time-sensitive software-defined networks," *IEEE Trans. Netw. Service Manage.*, vol. 20, no. 1, pp. 533–546, Mar. 2023.

[51] A. Arestova, K.-S. Jens Hielscher, and R. German, "Simulative evaluation of the TSN mechanisms time-aware shaper and frame preemption and their suitability for industrial use cases," in *Proc. IFIP Netw. Conf. (IFIP Networking)*, Jun. 2021, pp. 1–6.

[52] J. Ma, J. Zhang, J. Zou, H. Yu, T. Taleb, Y. Dong, and Y. Ji, "Demonstration of latency control label-based bounded-jitter scheduling in a bridged network for industrial Internet," in *Proc. Eur. Conf. Opt. Commun. (ECOC)*, Sep. 2021, pp. 1–4.

[53] O. Konradi, A. Mankowski, L. Wisniewski, and H. Trsek, "Towards an industrial converged network with OPC UA PubSub and TSN," in *Proc. IEEE 27th Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2022, pp. 1–4.

[54] G. Miranda, E. Municio, J. Haxhibeqiri, D. F. Macedo, J. Hoebeke, I. Moerman, and J. M. Marquez-Barja, "Time-sensitive networking experimentation on open testbeds," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, May 2022, pp. 1–6.

[55] A. G. Hagargund, N. S. V. Shet, and M. Kulkarni, "DTPF algorithm based open-source time-sensitive network leveraging SDN architecture," *IEEE Access*, vol. 11, pp. 71037–71047, 2023.

[56] J. Woods. (May 2017). *Cut-Through Considerations and Impacts to Industrial Networks*. Analog Devices. [Online]. Available: https://www.ieee802.org/1/files/public/docs2017/new-woods-cutthroughconsiderations-0518-v01.pdf

[57] M. Ulbricht, P. Dockhorn, C. Liss, and J. Wagner, "Highspeed hierarchical routing—A practical approach to single-clock routing," in *Proc. 11th Int. Symp. Commun. Syst., Netw. Digit. Signal Process. (CSNDSP)*, Jul. 2018, pp. 1–6.

[58] P. Congdon, M. Farrens, and P. Mohapatra, "Packet prediction for speculative cut-through switching," in *Proc. 4th ACM/IEEE Symp. Architectures Netw. Commun. Syst.*, Nov. 2008, p. 99108.

[59] C. Liß, M. Ulbricht, U. F. Zia, and H. Müller, "Architecture of a synchronized low-latency network node targeted to research and education," in *Proc. IEEE 18th Int. Conf. High Perform. Switching Routing (HPSR)*, Jun. 2017, pp. 1–7.

[60] A. Frankó and G. Hollósi, "Settling issues in IEEE 802.1AS networks in PI based clock servos," in *Proc. 19th Int. Conf. Netw. Service Manage. (CNSM)*, Oct. 2023, pp. 1–7.

[61] R. Maegawa, D. Matsui, Y. Yamasaki, and H. Ohsaki, "A discrete model of IEEE 1588-2008 precision time protocol with clock servo using PI controller," in *Proc. IEEE 43rd Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, vol. 2, Jul. 2019, pp. 531–536.

[62] V. Q. Nguyen, T. H. Nguyen, and J. W. Jeon, "An adaptive fuzzy-PI clock servo based on IEEE 1588 for improving time synchronization over Ethernet networks," *IEEE Access*, vol. 8, pp. 61370–61383, 2020.

[63] *Time Sensitive Networks for Flexible Manufacturing Testbed Characterization and Mapping of Converged Traffic Types*, Ind. Internet Consortium, Boston, MA, USA, Mar. 2019.

[64] G. Nardini, D. Sabella, G. Stea, P. Thakkar, and A. Virdis, "Simu5G—An OMNeT++ library for end-to-end performance evaluation of 5G networks," *IEEE Access*, vol. 8, pp. 181176–181191, 2020.

[65] T. Deinlein, R. German, and A. Djanatliev, "5G-Sim-V2I/N: Towards a simulation framework for the evaluation of 5G V2I/V2N use cases," in *Proc. Eur. Conf. Netw. Commun. (EuCNC)*, Jun. 2020, pp. 353–357.

[66] F. Aghaei and A. Avokh, "MMS: Multi-rate multicast scheduling in multi-radio singlecell CR-WMNs," *Majlesi J. Electr. Eng.*, vol. 13, no. 4, pp. 39–49, 2019.

[67] F. Aghaei, H. B. Eldeeb, L. Bariah, S. Muhaidat, and M. Uysal, "Comparative characterization of indoor VLC and MMW communications via ray tracing simulations," *IEEE Access*, vol. 11, pp. 90345–90357, 2023.

[68] F. Aghaei, H. B. Eldeeb, and M. Uysal, "A comparative evaluation of propagation characteristics of vehicular VLC and MMW channels," *IEEE Trans. Veh. Technol.*, vol. 73, no. 1, pp. 4–13, Jan. 2024.

[69] C.-W. Chow, "Recent advances and future perspectives in optical wireless communication, free space optical communication and sensing for 6G," *J. Lightw. Technol.*, early access, Apr. 9, 2024, doi: 10.1109/JLT.2024.3386630.

[70] S. Naser, L. Bariah, S. Muhaidat, P. C. Sofotasios, M. Al-Qutayri, E. Damiani, and M. Debbah, "Toward federated-learning-enabled visible light communication in 6G systems," *IEEE Wireless Commun.*, vol. 29, no. 1, pp. 48–56, Feb. 2022.

[71] G. Singh, A. Srivastava, and V. A. Bohara, "Visible light and reconfigurable intelligent surfaces for beyond 5G V2X communication networks at road intersections," *IEEE Trans. Veh. Technol.*, vol. 71, no. 8, pp. 8137–8151, Aug. 2022.

[72] *IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks—Amendment 31: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements*, IEEE Standard 802.1Qcc-2018, Amendment to IEEE Standard 802.1Q-2018 as Amended by IEEE Standard 802.1Qcp-2018, 2018, pp. 1–208.

HOW-HANG LIU received the B.Sc. degree in communication engineering from National Central University (NCU), Taoyuan, Taiwan, in 2014, and the M.Sc. degree in communication engineering from National Taiwan University (NTU), in 2016. He is currently pursuing the Ph.D. degree with the Electrical and Computer Engineering Department, Technische Universität Dresden, Germany. His research interest includes time-sensitive networking.

STEFAN SENK (Graduate Student Member, IEEE) received the Diplom-Ingenieur (Dipl.-Ing.) degree in electrical engineering from Technische Universität Dresden, in July 2019, where he is currently pursuing the Ph.D. degree with the Deutsche Telekom Chair of Communication Networks, with a focus on 5G non-public networks, deterministic communication, and human–machine-collaboration.

MARIAN ULBRICHT received the bachelor's and master's degrees in communication technology from Hochschule für Telekommunikation Leipzig (HfTL), with a focus on embedded systems and microcontroller programming. He is currently pursuing the Ph.D. degree with Technische Universität Dresden. Since 2015, he has been a Software Developer and a Project Engineer with InnoRoute GmbH, Munich, with a focus on TSN and network node design.

HOSEIN K. NAZARI received the B.Sc. degree in information technology engineering and the M.Sc. degree in computer science from the Institute for Advanced Studies in Basic Sciences (IASBS), Zanjan, Iran, in 2018 and 2021, respectively. He is currently pursuing the Ph.D. degree in electrical and computer engineering with Technische Universität Dresden, Germany.

TOBIAS SCHEINERT (Graduate Student Member, IEEE) received the Diplom-Ingenieur (Dipl.-Ing.) degree in electrical engineering from Technische Universität Dresden, in May 2023, where he is currently pursuing the Ph.D. degree with the Deutsche Telekom Chair of Communication Networks.

MARTIN REISSLEIN (Fellow, IEEE) received the Ph.D. degree in systems engineering from the University of Pennsylvania, Philadelphia, PA, USA, in 1998. He is currently a Professor with the School of Electrical, Computer, and Energy Engineering, Arizona State University (ASU), Tempe, AZ, USA. He is currently an Associate Editor of IEEE ACCESS and IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT.

GIANG T. NGUYEN received the Ph.D. degree in computer science from TU Dresden, Germany, in 2016. He is currently an Assistant Professor and heading the Haptic Communication Systems Research Group, Cluster of Excellence Center for Tactile Internet with Human-in-the-Loop (CeTI), and the Faculty of Electrical and Computer Engineering, TU Dresden.

FRANK H. P. FITZEK (Senior Member, IEEE) received the Ph.D. (Dr.-Ing.) degree in electrical engineering from Technical University Berlin, Germany, in 2002. He is currently a Professor and the Head of the Deutsche Telekom Chair of Communication Networks, Technische Universität Dresden, Germany, coordinating the 5G Laboratory. He is the Spokesman of the DFG Cluster of Excellence CeTI.