eTAS: Enhanced Time-Aware Shaper for Supporting Nonisochronous Emergency Traffic in Time-Sensitive Networks

Moonbeom Kim, Doyeon Hyeon, and Jeongyeup Paek, *Senior Member, IEEE*

***Abstract*—To guarantee stringent real-time requirements of time-critical traffic in industrial systems, the IEEE time-sensitive networking (TSN) task group has standardized time-aware shaping (TAS) in IEEE 802.1Qbv, which schedules precise and periodic transmission times using preassigned traffic information. However, nonperiodic/unexpected but time-critical traffic, such as emergency events or alarms, may occur in real industrial scenarios, and TAS does not provision for performance of traffic that are unknown *a priori*, nor the impact thereof on prescheduled traffic. Moreover, recalculating the schedule for every sporadic, nonisochronous event traffic is extremely difficult, complex, and costly. To address these challenges, we propose a novel enhancement to TAS, referred to as *eTAS*, which defines a new scheduling rule for immediate forwarding of emergency traffic to guarantee real-time performance, while dynamically extending the scheduled time windows to protect scheduled time-critical traffic from the interference of emergency traffic. We evaluate *eTAS* through extensive simulations on OMNeT++ under an advanced driver assistance system (ADAS) scenario for autonomous driving to show that *eTAS* effectively allows rapid transmission of event traffic with minimal impact on scheduled traffic, even for highly congested networks.**

***Index Terms*—IEEE 802.1, IEEE 802.1Qbv, Internet of Things (IoT), time-aware shaper (TAS), time-sensitive network (TSN).**

I. Introduction

REAL-TIME networking and differentiated Quality of Service (QoS) are becoming more critical in many recent Internet of Things (IoT) applications and industrial systems, such as automotive, aviation, and factory. In these systems, multiple traffic flows with different types, priorities, data rates, and requirements coexist in a single local area network. Then, satisfying the demands of time-critical (also known as time-sensitive) traffic with extremely low latency and jitter requirements, while allowing support for best-effort (BE) traffic and high network utilization, becomes a challenge.

With the goal of providing standard-based real-time local area networking (LAN), the IEEE time-sensitive networking (TSN) task group [1] is actively working on the IEEE 802.1 TSN standards that extend the previous Ethernet (IEEE 802.3) [2], [3] and bridging/queueing (IEEE 802.1) [4] standards for real-time support while maintaining compatibility with the legacy Ethernet. TSN consists of several innovative standards and amendments, such as IEEE 802.1Qbv [5] (traffic shaping), IEEE 802.1AS [6] (clock synchronization), and IEEE 802.1Qcc [7] (network management). In particular, the IEEE 802.1Qbv defines the "*time-aware shaper*" (TAS) mechanism that controls the transmission times of frames within a switch using a scheduled gating scheme as shown in Fig. 1. Then, a coordinated set of TAS schedules for switches in the network can be computed to guarantee stringent low-latency and low-jitter requirements of time-critical traffic.

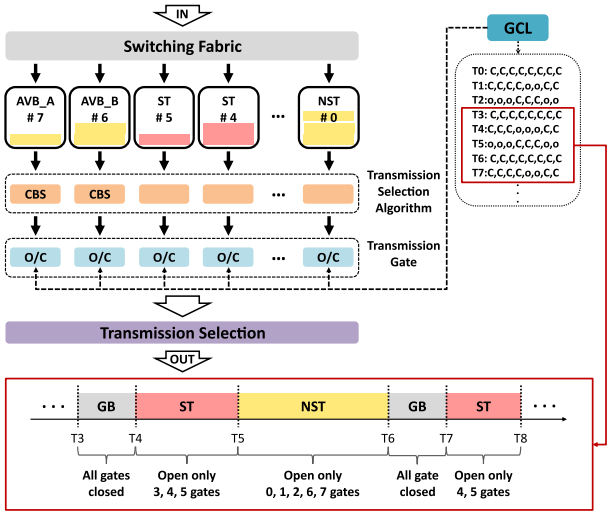

Fig. 1. TAS of IEEE 802.1Qbv amendment.

However, although the IEEE 802.1Qbv amendment defines how the TAS mechanism handles the transmission of frames within a scheduled switch [5], it does not specify an algorithm to compute those schedules. Computing correct and coordinated TAS schedules for switches along the path of flows in the network (generally suggested as the role of CNC in IEEE 802.1Qcc) is critical for achieving the goals of TSN. However, it is challenging and expensive because it should take into account several factors, such as path, link capacity and utilization, transmission interval, frame size, deadline, latency, and jitter [8], [9]. Thus, TAS scheduling is classified as an NP-complete problem [10].

Manuscript received May 10, 2021; revised September 14, 2021; accepted October 16, 2021. Date of publication November 2, 2021; date of current version June 23, 2022. This work was supported in part by the National Research Foundation of Korea (NRF) grant funded by the Korea Government (MSIT) under Grant NRF2021R1A2C100884011, and in part by the Chung-Ang University Graduate Research Scholarship in 2020. *(Corresponding author: Jeongyeup Paek.)*

The authors are with the Department of Computer Science and Engineering, Chung-Ang University, Seoul 06974, Republic of Korea (e-mail: mbkim@cau.ac.kr; nochedeuyuni@cau.ac.kr; jpaek@cau.ac.kr).

Digital Object Identifier 10.1109/JIOT.2021.3124508

Many prior studies have investigated this issue of optimal and/or efficient TAS scheduling (related work in Section VI). These studies have proposed constraints for correct scheduling [10]–[12], methods to reduce enormous scheduling costs and complexity using heuristics [13]–[19], methods to operate under limited resources [20]–[22], and suggest suitable scheduling schemes according to applications [23]–[27] or network environments [28]–[31]. This is still an active and ongoing research topic.

However, most prior work on TAS scheduling assumes isochronous time-critical traffic, which are known *a priori*. None of them have considered nonisochronous time-critical traffic that may occur unexpectedly in real industrial systems. For example, when an emergency event (e.g., fire) is detected, the system will need to propagate corresponding messages over the network immediately in real time to alert people and actuate countermeasure devices (e.g., sprinklers) in order to address the problem. These *emergency event traffic* (ET) are time sensitive and critical, as any delays or losses may result in fatal damage to the system. With the standard TAS scheme, however, these event traffic can suffer serious delays, and can also interfere with and adversely affect the performance of *prescheduled isochronous time-critical traffic* (ST). Therefore, an enhancement to TAS that can guarantee not only the performance of prescheduled traffic but also sporadic event traffic is needed, which has not been addressed yet.

The contributions of this work are summarized as follows.

1) Identify and analyze the problems that arise when nonscheduled but high-priority event traffic is introduced into a prescheduled TSN network. We show that sporadic high-priority event traffic may experience excessive delays due to a lack of schedule for itself in a TSN network, and adversely affect the performance of prescheduled time-critical traffic.
2) Propose *eTAS*, a novel dynamic scheduling scheme that enhances TAS. *eTAS* defines a scheduling rule for immediate forwarding of ET, and guarantees stringent requirements of prescheduled scheduled traffic (ST) from the interference of ET by temporarily extending the scheduled time windows of time-critical traffic when ET occurs.
3) Through extensive evaluations, we demonstrate that *eTAS* allows real-time transmission of high-priority event traffic with minimal impact on scheduled TSN traffic performance.

The remainder of this article is organized as follows. We first introduce TAS in Section II, and describe the problem that we are addressing in Section III. We present the design of the proposed *eTAS* in Section IV, and evaluate *eTAS* in Section V. We summarize related work in Section VI, and finally, conclude this article in Section VII.

II. Background

In TSN, network traffic types are typically divided into time critical, semitime critical (audio/video), and BE. Time-critical traffic is defined as flows with stringent low-latency, low-jitter, and zero-congestion-loss requirements, and is usually assumed to have periodicity and high priority. The other traffic types have relatively characteristics of lenient requirements and periodicity or sporadicity. With the legacy IEEE 802.1 standard "before TSN," time-critical traffic could not meet their requirements even with "strict priority scheduling" [nor "credit-based shaper (CBS)" in *audio video bridging* (AVB) standards] due to the potential interference from other traffic flows. In order to solve this problem, TSN introduced "TAS," which isolates transmission times of time-critical traffic from other traffic types by scheduling transmission "gates" based on preconfigured information of flows and network.

Fig. 1 illustrates an example operation of the TAS in a TSN switch. An egress port of a switch can have up to eight queues, each corresponding to a *traffic class*. When a frame enters the switch, the switch identifies its "priority" in *priority code point* (PCP) of the VLAN identifier located in IEEE 802.1Q [32] Ethernet header. Then, each frame is mapped to a traffic class according to the priority based on the mapping recommended in IEEE 802.1Q [32], after which the frame is inserted into the queue corresponding to the traffic class.[1] The PCP value of a frame can vary depending on the traffic type. However, the AVB standards [33], [34] recommend that audio and video traffic have PCP values 3 and 2 that correspond to "SR class" AVB-A and AVB-B, respectively, and are mapped to the highest traffic classes. For example, if the number of queues is eight, the frames of AVB-A and AVB-B types are assigned traffic classes 7 and 6, and are inserted into 8th and 7th queue, respectively.

Each queue may have an associated "transmission selection algorithm" (orange box immediately below each queue in Fig. 1), which determines whether the queue should provide a frame for transmission if it has any [32]. CBS [35] and *asynchronous traffic shaper* (ATS) [36] are examples of such algorithms, where their role is to throttle the queues based on some stream reservation [34] criteria and regulate transmissions to protect other lower priority traffic. Each queue additionally has a "transmission gate" (blue box below each queue in Fig. 1), which is the key component of the TAS. A "gate" has two states, open(o) and closed(c), and frames in each queue can be transmitted only when the gate for that queue is open. If multiple gates are open simultaneously and if more than one of those queues have frames available for transmission, then the "transmission selection" (purple box below the gates in Fig. 1) selects and transmits a frame using strict priority criteria (i.e., in decreasing order of traffic class).

The gates are controlled by the *gate control list* (GCL). The GCL ensures that specified traffic classes are transmitted at specific times by defining how long each gate is open and closed (top right box in Fig. 1). The opening and closing periods of the gates are computed based on several constraints, such as network topology, configurations, traffic types, number of flows, and frame size and interval of each flow in the

[1]To be precise, traffic class and priority are two different, but related terms in the IEEE 802.1Q standard. The recommended mapping from priority to traffic class depends on the number of queues supported by the switch, and is configurable. OMNeT++, the simulator used in this work, adopts a one-to-one mapping between the PCP and traffic class, so we use this as is.

network. To compute the GCL, TAS first assigns time-critical periodic traffic to ST time windows. The duration of ST time windows is assigned just enough[2] to accommodate the sum of transmission times of frames that need to be transmitted during that time window, while allowing the remaining time to be used by less-time-critical non-ST (NST), such as AVB or BE. The schedule is determined based on ST because, by definition, the amount and schedule of NST are unknown and need not be known, and would prefer to maximize the bandwidth utilization.

During the ST time windows, only the gates relevant to the ST queues are opened, and only the frames in those queues are transmitted into the network. Furthermore, to avoid delay in ST due to interference from a transmission of a long NST frame just before the beginning of a ST time window, *guard band* (GB) is added in front of the ST time windows as shown at the bottom of Fig. 1. The length of the GB is set to be as long as the transmission time of the largest Ethernet frame[3] in the network, and all gates are closed during GB. Therefore, even if the longest frame is transmitted just before the GB starts, the transmission is completed within the GB time window and does not interfere with ST.

Finally, all remaining time other than those for ST and GB are allocated as NST time windows for transmitting all other traffic types. During these time windows, as opposed to ST time windows, all gates relevant to ST traffic are closed, and only the other gates are opened. This is the default "exclusive gating" method recommended in the IEEE 802.1Q standard. However, the standard does also allow deviation from exclusive gating, and *eTAS* exploits this opportunity as we will describe in Section IV.

III. Problem and Motivation

TAS can satisfy the requirements of time-critical traffic given that it has an appropriate set of GCLs computed using the topology and flow information configured in advance (i.e., frame size, interval, path, etc.). In other words, TAS cannot guarantee performance if the necessary information is not known in advance or varies over time. However, emergency events, such as accidents, intrusions, or failures, may occur without prior notice in real industrial systems, and these ET must be forwarded as quickly as possible to protect lives and property [39]. To this end, we identify the problems that arise when ET is introduced to a TSN with a standard TAS, which motivates our work. We assume that ET has the highest priority that maps to the highest traffic class within the switch, and we do not consider the frame preemption technique in IEEE 802.1Qbu/3br [37], [38] to keep the discussion concise for ease of understanding.

We will look at three cases as follows.

[2]Fitting the ST time window size tightly to the required ST amount by scheduling ST frames back-to-back (if possible) is called "compact scheduling." The standard does not prohibit relaxing the time window to give some extra room in ST, but (to the best of our knowledge) there have not been any proposals to do so since it will reduce the NST bandwidth and overall bandwidth utilization.

[3]e.g., Maximum Ethernet MTU plus interframe gap, 1542 bytes, when the frame preemption technique is not used [37], [38].

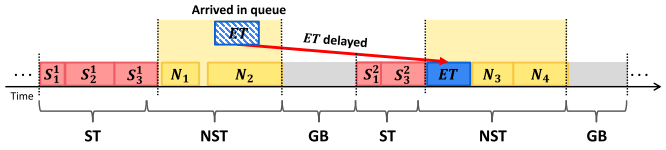

Fig. 2. ET is transmitted in NST time windows (*ET-in-NST*).

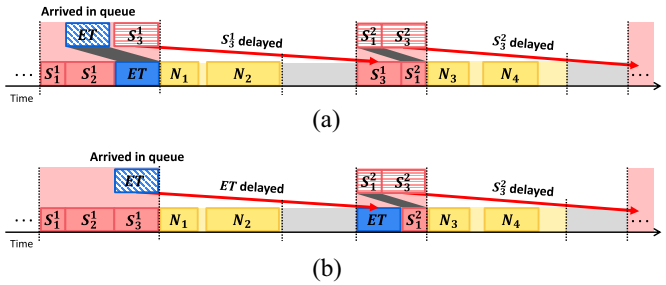

Fig. 3. ET is transmitted in ST time windows (*ET-in-ST*). (a) Cascading delay problem of ST due to the occurrence of ET. (b) Critical delay for ET, followed by cascading delay for ST.

1) ET in NST time windows (*ET-in-NST*).
2) ET in ST time windows (*ET-in-ST*).
3) ET in both ST and NST time windows (*ET-in-ST&NST*).

Case-1 (ET-in-NST): Since TAS does not take ET into account when scheduling, ET will naturally and inevitably be sent during NST time windows in a standard TSN. Then, if ET has the highest priority,[4] it will generally be transmitted before other NST frames (e.g., AVB and BE) queued in the switch. However, even the highest priority frame must wait until the current ongoing transmission is completed if there is one. This may lead to a significant delay in ET if the ET frame arrives in the queue near the end of an NST time window while there is an NST frame currently being transmitted. For example, in Fig. 2, an ET frame (blue shaded box) arrives while an NST frame (N_2) is being transmitted. Even when N_2 transmission is completed, ET waiting in the queue cannot start its transmission because all gates are closed by the next GB. Therefore, ET has to wait until the next NST time window, possibly on every switch along the path of the flow, which may result in a significant delay for ET.

Case-2 (ET-in-ST): The TSN standard is flexible in the sense that it allows configurations other than what is widely accepted as the basic/default TSN behavior within the scope of the standard. It is possible for the network administrator to configure the GCL such that the TAS gates for ET are open during ST time windows without allocating a dedicated (and possibly wasted) time schedule for ET. However, Fig. 3 illustrates the problems that may arise for both ST and ET in the *ET-in-ST* scenario.

First, suppose that an ET frame arrives in the queue within the ST time window, as shown in Fig. 3(a), before the transmission of the last ST frame within that ST time window (e.g., S_3^1). When the transmission of S_2^1 is completed, ET occupies the time reserved for S_3^1 even though S_3^1 is waiting to be transmitted. Then, S_3^1 is deprived of its opportunity to be

[4]We mean "highest traffic class" interchangeably in this context.

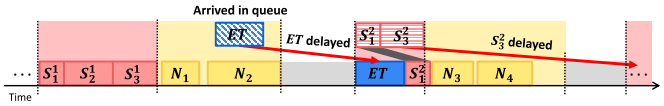

Fig. 4. ET is transmitted in both ST & NST time windows (*ET-in-ST&NST*).

transmitted, and is delayed until the next ST time window. Moreover, if S_3^1 is pushed and shifted to the next ST window, there will be a chain reaction of "*cascading delays*" on the following ST frames (e.g., S_1^2 and S_3^2), which were supposed to be transmitted in subsequent ST time windows. Furthermore, if an ET frame arrives after the beginning of the last ST frame within an ST time window [as shown in Fig. 3(b)], the ET frame will be delayed significantly until the next ST time window, followed again by the same *cascading delay* problem for subsequent ST frames [as in Fig. 3(a)]. This problem will only get worse and worse as more ET frames steal the reserved time windows of the ST frames.

Allocating dedicated time slots for ET when computing the TAS schedule could be one way to alleviate this problem. This approach will guarantee that ST traffic is not affected by the ET. However, it will mostly likely waste bandwidth (due to unused ET slots), increases the delay for NST traffic (by ET slot size), and ET frames may still experience significant delays when they miss their slot and wait for the next one. Moreover, because ET is rare and occurs sporadically, it is extremely difficult to properly allocate time slots for ET across the entire timeline.

Case-3 (ET-in-ST&NST): Fig. 4 illustrates the problem that may occur when ET transmission is allowed in both ST and NST time windows. This approach will reduce the average delay of ET, but ET can still be delayed by the GB periods. In the worst case, if an ET frame arrives when there is an ongoing transmission of maximum Ethernet frame size near the end of NST time window, ET can be delayed by up to 246.72 μs, twice the GB period (GB period is 123.36 μs on a 100-Mb/s link, the transmission time of max Ethernet frame size). More importantly, the same *cascading delay* problem for subsequent ST frames will follow as in the *ET-in-ST* case.

One way to alleviate the excessive delay of ET and ST would be to discard the TAS and go back to the legacy strict priority transmission scheme with an appropriate configuration of traffic classes. For example, traffic class mapping may be configured in the order of ET > ST > AVB > BE [23], [24], [40], [41] such that ET and ST are always transmitted before other types. If we discard the TAS, we can eliminate the GB and improve both the total bandwidth and average latency of nontime-critical traffic. However, this approach would nullify the whole purpose of TAS; guaranteeing low-latency and low-jitter for "ST" traffic. Thus, we need a mechanism that enhances TAS such that ET is delivered with low-latency while minimizing the impact on ST, which we describe next.

IV. eTAS Design

eTAS extends TAS in the IEEE TSN standard to support ultralow-latency nonisochronous ET with minimal impact on prescheduled time-critical traffic. This section first describes the *queueing rule modification for ET*, and then the *dynamic scheduled time window extension* of *eTAS* that achieves this goal.

TABLE I
Recommended Priority to Traffic Class Mappings for ET (Blue), AVB (Red), and Other Traffic

		Number of available traffic classes						
		2	3	4	5	6	7	8
Priority	0	0	0	0	0	0	0	1
	1	0	0	0	0	0	0	0
	2	0	1	1	2	3	4	5
	3	0	1	2	3	4	5	6
	4	0	0	0	1	1	1	2
	5	0	0	0	1	1	2	3
	6	0	0	0	1	2	3	4
	7	1	2	3	4	5	6	7

A. Queueing Rule Modification for ET

We first propose a few modifications to the queueing/scheduling rules in the IEEE 802.1Q standard that are needed to achieve the goals of the *eTAS*.

ET Should Have the Highest Priority and Traffic Class, With an Independent Queue: If multiple frames from multiple egress queues are available for transmission after passing the "transmission selection algorithm" and the "transmission gate" (Fig. 1), then the final decision on which frame is transmitted next generally follows the strict priority transmission selection rule, i.e., a frame with the highest traffic class is selected and transmitted from the queue. This is the standard and is a widely accepted and well-known behavior. Under the assumption and claim that real industrial systems will have critical emergency events that must be propagated over the network immediately with the lowest possible latency, we propose that ET should be designated an *independently highest priority* that maps to an *independently highest traffic class* so that it is always selected for the next transmission when available.

This may sound obvious at a glance. However, two points are worth mentioning. First, the current IEEE 802.1Q-2018 [32] standard defines AVB traffic (SR class "A" and "B") to have the highest traffic classes while their priority values are defined as 3 and 2 (from amendments 1Qat [34] and 1Qav [35]). For example, if the number of available queues in a switch is 8, then AVB-A and AVB-B traffic are assigned classes 7 (highest) and 6, respectively. Second, if multiple priority values are mapped to the same traffic class, then frames with those priority values will be place into the same egress queue. In this case, a frame with lower priority may interfere with and delay another frame with higher priority because they are in the same traffic class queue. Therefore, to transmit ET with the lowest latency, it must be assigned not only the highest traffic class but also an "independent" queue of its own.

To this end, Table I redefines the priority-to-traffic class mapping table in the IEEE 802.1Q standard [32] for *eTAS*. The part shaded in blue indicates the traffic classes (i.e., queue number) for ET to which the ET priority is mapped according to the number of queues (traffic classes) supported by the switch. The red shading indicates the traffic classes to which the priorities for AVB traffic types should be remapped without modifying the priorities for AVB. AVB is mapped to the

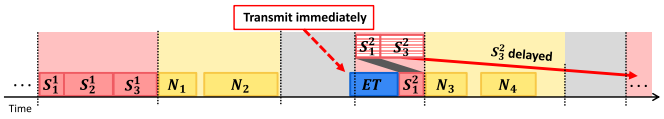

Fig. 5. ET is transmitted in all time windows including GB (*ET-in-ALL*).

second highest traffic class after ET, unlike the standard that maps the highest traffic class to the AVB. This is to dedicate an "independent" traffic class (queue) to ET, without any overlap with other priorities, regardless of the number of queues supported by the switch (Table I). For example, if the number of queues supported by the switch is 5 and ET has a priority value of 7, ET is mapped to traffic class 4 (highest). AVB-A and AVB-B are mapped to the next highest values of 3 and 2, respectively.

TAS Gate Corresponding to ET Must Always Be Open Including GB Time Window: As discussed in Section III, if ET can be transmitted only in either the ST or NST time windows, ET will experience serious delays waiting for the corresponding time window, and may adversely affect other traffic as shown in Figs. 2 and 3. Even if ET is allowed to be transmitted in both ST and NST time windows as shown in Fig. 4, it may still experience a worst case delay of twice the size of the GB at every switch. Therefore, to reduce the delay of ET, we propose that the gate for ET is always open, and transmission is allowed in any time window, including the GB. Then, the only remaining source of delay for ET is the time waiting for the preceding transmission to be completed, which can be handled by the frame preemption technique [37], [38].

By allowing the transmission of ET in all time windows, ET can be transmitted immediately without waiting even when it arrives during the GB time window, as shown in Fig. 5, resulting in lower latency. However, it may still interfere with ST traffic, which we discuss next.

B. Dynamic Scheduled Time Window Extension

If an ET frame starts transmission near the end of a GB window, it may invade the ST time window, as exemplified in Fig. 5. In this case, the ST time window may expire before the transmission of S_3^2 due to delayed S_1^2, and thus, S_3^2 will be pushed to the next ST window, resulting in significant and cascading delays for subsequent ST frames, as in the *ET-in-ST* and *ET-in-ST&NST* cases. To address this problem, we propose the "dynamic scheduled time window extension" (TWE) technique, which "temporarily" extends the ST time window, when needed, while adhering to the original predefined GCL schedule.

Fig. 6 illustrates the proposed operation procedure for the dynamic scheduled TWE in *eTAS*.

1) An ET frame arrives at its queue.
2) "Transmission selection" selects ET over other frames according to the rules defined in Section IV-A.
3) Then, 3-1) signals the GCL by sending a "*TWE message*," which includes the size of the ET frame to be transmitted, and 3-2) transmits the ET frame.
4) Next, GCL calculates the transmission time of the ET frame and accumulates this to a variable τ_{ext}, which indicates the duration of time by which the ST window may require an extension.

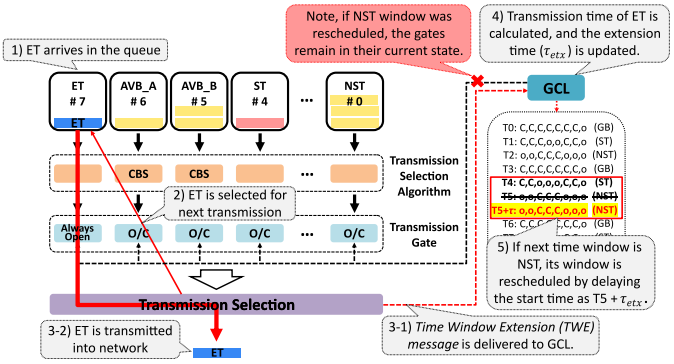

Fig. 6. *eTAS* operation procedure overview.

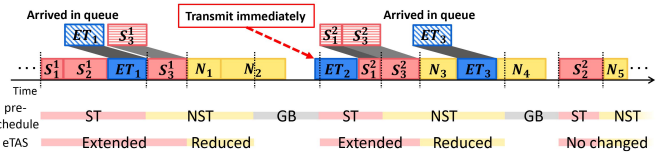

Fig. 7. ET transmission examples when applying *eTAS* scheme.

5) After that, when it is time for the GCL to change the state of the gates to the next state and if the next state is the NST time window, the GCL delays the transition by τ_{ext}.

In other words, the state of the gates does not change for another τ_{ext}, and thus, the ST time window duration is temporarily extended by the time amount used by ET frames.

Algorithm 1 is the pseudocode of how the *eTAS* extends the scheduled time window within the GCL. Consider the frame arrival example shown in Fig. 7. At the bottom of Fig. 7, the *preschedule* line and *eTAS* line each represent a predefined (fixed) schedule and a schedule dynamically adapted by *eTAS*, respectively. Recall that when an ET frame arrives in the queue and there is no ongoing transmission, the "transmission selection" component selects the ET frame and delivers a *TWE message* to the GCL just before starting transmission. When the GCL receives the *TWE message* from "transmission selection," GCL calculates the transmission time of the ET frame and updates τ_{ext} (lines 1–9 in Algorithm 1).

1) *If the transmission of the ET frame starts in an ST time window* (e.g., ET_1 in Fig. 7), the GCL accumulates the ET transmission time in τ_{ext} (line 6).
2) In contrast, *when the transmission of ET starts in the GB or NST time window* (e.g., ET_2 or ET_3 in Fig. 7), the GCL replaces τ_{ext} with the transmission time of ET and records the current time to *lastTimestamp* (lines 8 and 9).

eTAS repeats this process until it transits to the next time window.

In *eTAS*, GCL determines whether to extend the time window when it receives a *time window transition (TWT) message* from the "gate scheduler" (lines 12–29). Gate scheduler is the component that iterates through GCL using "cycle timer" based on the predefined schedule, and it sends a TWT message to the GCL when it is time to transition to the next state in the original schedule. Then, *eTAS* first copies the information for the next time window (next row in GCL) to keep the predefined schedule intact (line 13).

Algorithm 1 Scheduled Time Window Extension

Input: *TWE_message or TWT_message*

```
1: if TWE_message from transmissionSelection then
2:     bitsSize ← TWEmessage.bytesSize_ET * 8
3:     transmissionTime ← bitsSize / getLinkRate()
4:
5:     if currentWindow_ST then
6:         τ_ext += transmissionTime
7:     else
8:         τ_ext ← transmissionTime
9:         lastTimestamp ← currentTime
10:    end if
11:
12: else if TWT_message from gateScheduler then
13:    nextWindow ← getNextTimeWindow()
14:
15:    if currentWindow_ST and τ_ext ≠ 0 then
16:        delay(nextWindow.start, τ_ext)
17:        reschedule(nextWindow.start)
18:        initialization(τ_ext)
19:        return
20:    else if currentWindow_GB or NST and τ_ext ≠ 0 then
21:        if τ_ext + lastTimestamp > nextWindow.start then
22:            τ_ext ← τ_ext + lastTimestamp − nextWindow.start
23:            initialization(lastTimestamp)
24:        else
25:            initialization(τ_ext, lastTimestamp)
26:        end if
27:    end if
28:
29:    changeTimeWindow(nextWindow)
30: end if
```

1) *When the current time window is for ST and there is an ET that has transmitted during this window* (lines 15–19), GCL delays the start time of the next time window (i.e., NST time window) by τ_{ext}, reschedules the next transition time, and initializes τ_{ext} to zero. This means that the state of the gates do not change at this time because the GCL did not transition to the next time window.
2) *When the current time window is for GB or NST*, the GCL checks whether there is an ET frame that started transmission in the current time window and will complete in the next window (line 20). Taking ET_2 in Fig. 7 as an example, if the sum of *lastTimestamp* (the time when ET_2 transmission started) and τ_{ext} (transmission time of ET_2) is greater than the start time of the next window, this means that ET_2 will use some time of the next time window. Therefore, GCL recalculates τ_{ext} to obtain only the amount of time that ET_2 transmission will overlap with the next time window, and initializes *lastTimestamp* to zero (lines 21–23).
3) *Otherwise* (e.g., ET_3), τ_{ext} and *lastTimestamp* are reinitialized (lines 24 and 25).

After the above process, the GCL changes the state of the gates to the next time window (line 29).

Thus, as shown in the *eTAS* timeline in Fig. 7, the ST time windows affected by the occurrence of ET_1 and ET_2 are temporarily extended, and conversely the NST time windows are reduced by that amount. By doing this, the ST frames delayed due to ET can be transmitted within the extended ST time windows with delays of only the size of the ET frames. Therefore, the ST delay problem mentioned in Section III and Fig. 5 can be alleviated, and also the *cascading delay* problem is resolved. If ET does not occur, *eTAS* will not invoke the above process and act in the same way as the standard TAS, changing the state of the gates to the next time window strictly based on the prescheduled GCL (line 12 → line 29).

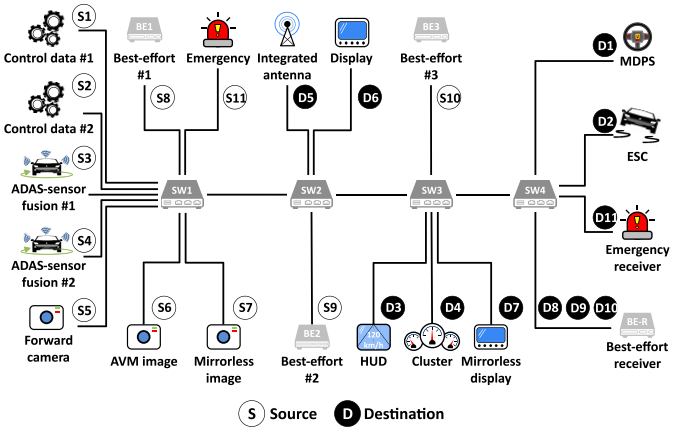

Fig. 8. Network topology for simulation of ADAS scenario. Traffic flows from sender(source) S* to receiver(destination) D*.

V. Evaluation

In this section, we first describe the simulation setup, and demonstrate that ET has serious negative impact on ST in TSN with TAS. We then evaluate the effectiveness of *eTAS* compared to that of standard TAS through an extensive set of simulations under various scenarios and configurations, as well as burstiness.

A. Simulation Setup

We use the OMNeT++ simulator [42] with INET 3.6.6 [43] and CoRE4INET [44], [45] frameworks for our evaluation. We simulate a TSN-based "advanced driver assistance system" (ADAS) scenario for autonomous driving. We have designed a TSN-based ADAS scenario by referring to several prior work on in-vehicle network (IVN) and TSN [39]–[41], [46], [47]. Fig. 8 depicts the network topology for our ADAS scenario, and Table II presents the configuration of traffic flows for our simulations. The ADAS network consists of 11 transmitters, 9 receivers, and 4 switches, and all nodes in the network are interconnected each by a meter long 100-Mb/s bandwidth Ethernet link. Each flow is assigned a fixed value of priority and traffic class according to our proposed mapping in Table I with eight queues, such that ET has the highest traffic class as shown in Table II.

We aim to simulate a very congested network because achieving low latency on a lightly loaded network is not a challenging goal. To match the total link utilization between switches to approximately $\approx 80\,\%$ for a fair and controlled experiment, we add three BE traffic transmitters and a BE traffic receiver, and adjust the transmission interval (datarate) of each BE flow as shown in Table III.[5] *"Non-isochronous*

[5]Unlike other traffic types, BE frames have no IEEE 802.1Q VLAN header. Thus, the L2 header size is 4 bytes smaller than other traffic types.

TABLE II
TRAFFIC CONFIGURATION OF ADAS SCENARIO

Traffic type	Traffic class	Sym	Source node	Interval (μs)	Payload size (Byte)	Link util (≈%)
CDT (ST)	4	S1	Control data #1	500	625	10.48
		S2	Control data #2	500	625	10.48
AVB-A (NST)	6	S3	ADAS-sensor #1	125	46	4.86
		S4	ADAS-sensor #2	125	46	4.86
AVB-B (NST)	5	S5	Forward camera	250	250	8.96
		S6	AVB image	250	256	9.15
		S7	Mirrorless image	250	256	9.15
BE (NST)	0	S8	Best-effort #1	550	1500	22.19
		S9	Best-effort #2	675	1500	18.08
		S10	Best-effort #3	646	1500	18.89
ET	7	S11	Emergency event	Random	625	-

TABLE III
LINK UTILIZATION BETWEEN SWITCHES

(≈%)	SW1 ⇒ SW2	SW2 ⇒ SW3	SW3 ⇒ SW4
Without BE	57.95	38.84	20.96
With BE	80.15	80.12	80.14

emergency traffic" (i.e., ET) transmitter generates an ET frame uniform randomly within each T seconds, once per T seconds during the simulation time, where we vary T (T = 1 s in our baseline simulations). The simulation time is set to 60 s.

Finally, the cycle time of the GCL is set to the least common multiple of all frame intervals of the ST flow (i.e., the cycle is set to 500 μs). The GCL of each switch is calculated taking into account the transmission, propagation, and processing delays. In the framework we use, the propagation delay ($\text{Delay}_{\text{prop}}$) is set to length(m)/($2 \times 10^8$ m/s) [43] by default, and also the processing delay ($\text{Delay}_{\text{proc}}$) is set to 8 μs on the switches [44], [45]. Therefore, the expected end-to-end latency (E-latency_{e2e}) can be calculated as follows:

$$\text{E-latency}_{e2e} = \sum_{\text{hop}=1}^{Cnt_{\text{hop}}} \left(\text{Delay}_{tx\_\text{hop}} + \text{Delay}_{\text{prop\_hop}}\right) + \sum_{sw=1}^{Cnt_{sw}} \left(\text{Delay}_{\text{proc}\_sw} + \text{Delay}_{\text{queue}\_sw}\right). \quad (1)$$

Among the cycle time of 500 μs in the GCL, the ST time window is set to be equal to the sum of the transmission times of two ST frames (106.72 μs), and the GB time window is set to the transmission time of the largest frame (i.e., BE, 123.04 μs) in the network. The remaining time in the GCL is allocated for the NST time window (270.24 μs).

B. Adverse Impacts of ET on ST With Standard TAS

We first evaluate the baseline performance of the ADAS scenario without ET (*w/o-ET*), and then compare this with the results from each case discussed in Section III.

TABLE IV
RESULT OF ADAS SCENARIO WITHOUT ET (*w/o-ET*)

Traffic type	End-to-end latency (μs)			Jitter avg. (μs)	Throughput (Mbps)
	Min.	Median	Max.		
CDT (ST)	295.82	322.5	349.18	2.64e-4	20.7
AVB-A (NST)	48.34	545.22	731.35	42.83	8.7
AVB-B (NST)	84.66	461.46	723.98	4.01	26.5
BE (NST)	382.26	875.59	1814.94	131.98	58.87

Baseline Without ET (w/o-ET): Table IV shows the latency, jitter, and throughput performance of each traffic type in our ADAS scenario "without ET". In our simulation setup, because two ST flows begin transmission at the same time, frames from the two ST flows arrive at SW1 simultaneously. Thus, one ST frame can be transmitted immediately without queueing delay, but the other will wait for a transmission time of an ST frame (i.e., 53.36 μs). Consequently, the expected minimum and maximum end-to-end latency (E-latency_{e2e}) of the ST frames from (1) are 294.025 and 347.385 μs, respectively. The simulation results in Table IV are consistent with these estimations, with only a slight difference of +1.79 μs. The throughput of ST also matches our intended data rate in Table II, which implies that there are no losses nor notable queueing delays.

Fig. 9 plots the latency, jitter, and throughput results for all scenarios, including *w/o-ET*, *ET-in-ST*, *ET-in-NST*, *ET-in-ST&NST*, as well as *eTAS*.

ET in ST Time Windows (ET-in-ST): Fig. 9(a) and (b) confirms that ST suffers most significant delays in the *ET-in-ST* case than other cases when ET occurs. Worst case latency and jitter of ST are measured as ≈15349 and ≈3755 μs, respectively, which are about 44 times higher than *w/o-ET*, and jitter is incomparably high. This is because ET takes away the transmission opportunity of the ST and causes a *cascading delay* problem as in Fig. 3, and the queueing delay of ST is continuously accumulated whenever ET occurs. For ET, the max. latency and jitter are measured as ≈736 and ≈121 μs, respectively, higher than the others scenarios. This is because ET has to wait until the next ST time window if ET arrives in the queue when the last ST is transmitting as in Fig. 3(b). Therefore, the queueing delay of ET can be as large as ≈447 μs (sum of the transmission time of an ST frame and duration of NST and GB time windows), so the maximum E-latency_{e2e} of an ET frame can be up to ≈741 μs in the worst case.

The throughput results in Fig. 9(c) show that for the *ET-in-ST* case, the throughput achieved by ET (5.176 kb/s) is exactly what has been lost by the ST compared to the *w/o-ET* case. The throughput of other NST flows has not changed at all. This result confirms that ET has been transmitted in, and has stolen the opportunity of, only the ST time windows.

ET in NST Time Windows (ET-in-NST): In this case, ET does not interfere with the transmission of ST, and thus, there is no impact on the ST by definition. However, ET can still

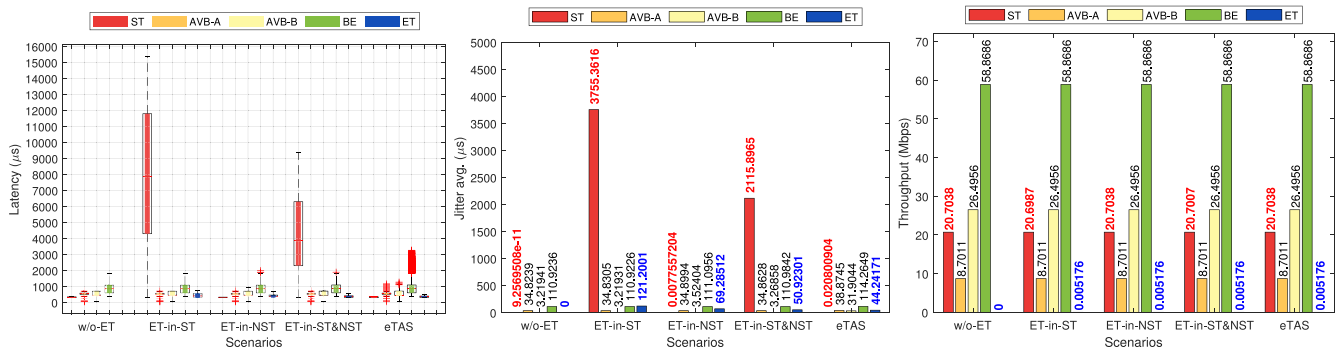

Fig. 9. Performance results for the five different ET transmission scenarios: *w/o-ET*, *ET-in-ST*, *ET-in-NST*, *ET-in-ST&NST*, and *eTAS*. (a) End-to-end latency for each traffic type. (b) Jitter average for each traffic type. (c) Throughput for each traffic type.

suffer significant queueing delays due to ongoing NST transmissions in switches along the path of ET flow, as well as ST and GB periods. Therefore, in the worst case, the maximum E-latency$_{e2e}$ of ET can be as large as ≈893 μs, 3 hops × 123.04 μs (max. BE frame size) greater than the minimum of ≈524 μs (sum of NST and GB time windows plus transmission/processing/propagation delay of ET frames). In the simulations, the max. latency and jitter of the ET are measured as ≈656 and ≈69 μs, respectively, which are ≈ 11 % and ≈ 43 % lower than the *ET-in-ST* case, but still approximately twice larger than what can be achieved as the theoretical minimum (≈ 294 μs, explained in the next section). The latency and jitter of other NST flows also increase slightly compared to the *w/o-ET* case.

For throughput, all flows achieve their demands, that is, no loss in throughput. This is because our ADAS simulation setup was configured to have a baseline link utilization of 80 % by configuring the datarate of BE flows appropriately (Table III). The remaining available times are in the NST time window by definition, and ET utilizes those times for its transmissions.

ET in Both ST and NST Time Windows (ET-in-ST&NST): Since ET can be transmitted in both ST and NST time windows, ET is expected to have lower latency than the previous two cases, but ET can also interfere with ST's transmissions. ET can be delayed by ongoing transmissions and GB, resulting in an estimated worst case maximum E-latency$_{e2e}$ of up to ≈786 μs. In the simulation results, ET has max. latency of ≈549 μs and jitter of ≈51 μs, which are lower than other cases with standard TAS as expected and shown in Fig 9(a) and (b). However, ST still suffers serious delay due to the *cascading delay* problem, as in the *ET-in-ST* case. The worst case latency and jitter of the ST are measured as ≈9349 and ≈2116 μs, respectively. Even though these are ≈ 39 % and ≈ 44 % lower than the *ET-in-ST* case, the latency is still about 27 times higher than *w/o-ET* and *ET-in-NST* cases, and jitter is also too high to satisfy the requirements of TSN.

The throughput of ET and NST is the same as other cases, but the throughput of ST is ≈3 kb/s lower than the *w/o-ET* case as shown in Fig. 9(c). This is because, since ET is allowed transmission in either ST or NST windows, ST lost some of its opportunities due to ET but NST had extra space in the link utilization (80 %).

C. Performance of eTAS Versus Standard TAS

So far, we have shown that the standard TAS has serious performance problems not only for ET but also for ST when ET is introduced into a TSN network. We now compare the performance of *eTAS* with standard TAS.

eTAS allows ET transmission in all time windows, including GB, unlike standard TAS. Therefore, assuming that ET is transmitted as soon as it arrives in its queue (zero queueing delay), the theoretical minimum end-to-end latency [E-latency$_{e2e}$ in (1)] of the ET in our topology is 294.025 μs. The delay that ET can experience due to ongoing transmission is 369.12 μs (for 3 hops in our scenario), so the estimated maximum E-latency$_{e2e}$ is up to ≈663 μs.

The simulation results for *eTAS* in Fig. 9 show that min. and max. latency of ET are ≈294 and ≈486 μs, respectively, with a median of 346.152 μs. The minimum is consistent with the estimation, and the maximum is ≈177 μs lower than the worst case, which is approximately 34 % / 26 % / 11 % lower than the *ET-in-ST*/*ET-in-NST*/*ET-in-ST&NST* scenarios, respectively. Median is closer to the minimum value. Moreover, the jitter of ET is about 44 μs, which is 63 % / 36 % / 13 % lower than each scenario, respectively. In other words, *eTAS* can deliver ET with a latency close to the theoretical minimum with low jitter.

By temporarily extending the ST time windows, *eTAS* can also minimize the ET's impact on ST. The maximum latency of ST is measured as ≈403 μs with *eTAS*, which is approximately 97 % and 96 % lower than the *ET-in-ST* and *ET-in-ST&NST* cases, respectively, while being only 15 % higher than the *w/o-ET* and *ET-in-NST* cases. The median latency is 348.096 μs, which is only 17.7 % (≈52 μs) higher than the minimum (295.785 μs). Furthermore, the max. increase in ST latency is exactly 53.36 μs, the size of an ET frame. This means that an ST frame is transmitted immediately after the completion of an ET frame transmission, which implies that the *eTAS* is operating as designed. The average jitter of ST is measured as ≈ 0.02 μs, which is small enough to satisfy the requirements of TSN. Also, the throughput of ST has not been impacted by ET. In other words, the results confirm that *eTAS* is capable of delivering ET with the lowest latency and minimal impact on ST, and it achieves better performance than all other cases with standard TAS.

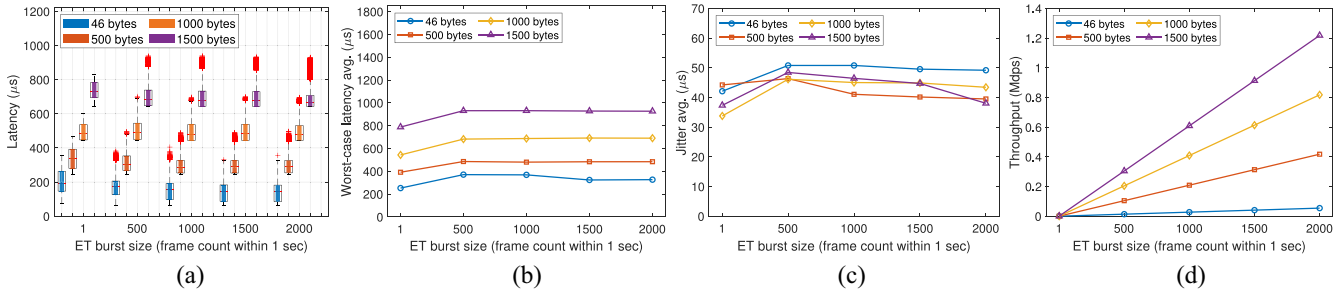

Fig. 10. ET performance results with varying ET burst and frame sizes. (a) Overall end-to-end latency (ET). (b) Worstcase end-to-end latency (ET). (c) Jitter average (ET). (d) Throughput average (ET).

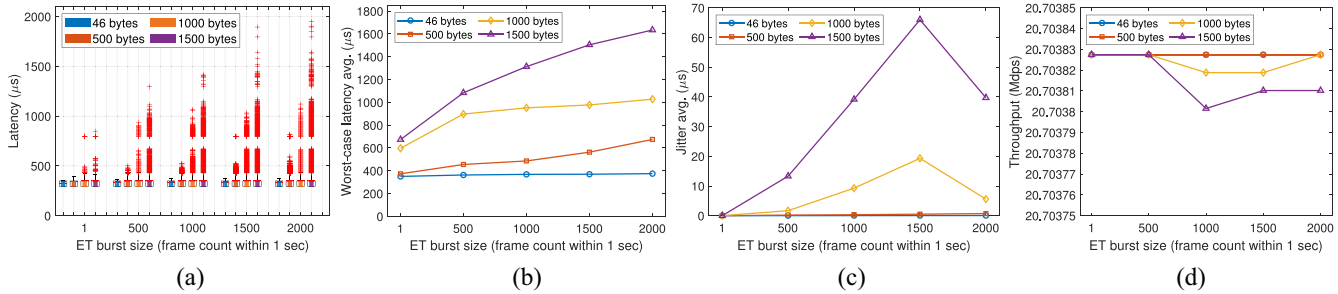

Fig. 11. ST performance results with varying ET burst and frame sizes. (a) Overall end-to-end latency (ST). (b) Worst-case end-to-end latency (ST). (c) Jitter average (ST). (d) Throughput average (ST).

Finally, the adequacy of the NST performance is worth mentioning. With *eTAS*, the worst case latency of AVB-A and AVB-B is measured to be approximately 66 % and 78 % higher than *w/o-ET* case, respectively. However, the AVB standard [33] defines the target maximum worst case latency for AVB-A and AVB-B as 2 and 50 ms, respectively, for 7 hops when a 100-Mb/s Ethernet link is used. When we extrapolate our results to 7 hops, *eTAS* is still well below the target latency suggested by the AVB standard, thus satisfying the requirements of TSN. The jitter of AVB traffic has increased slightly with *eTAS*, but the standard does not mandate any requirements on the jitter of the AVB as long as the max. latency requirement is met. Furthermore, the throughput of all NST flows with *eTAS* is the same as those for the *w/o-ET* case, thus no impact at all.

D. eTAS With Bursty Event Traffic

According to the documentation from the "Industrial Internet Consortium" [39], there are three types (alarm, operator command, and control) of sporadic event traffic with different requirements in industrial systems. In case of the "alarm" type, a burst of up to 2000 frames should be guaranteed within 1 s, after which some losses may be tolerated. Therefore, to verify the effectiveness of *eTAS* with bursty ET flows, we perform simulations with varying ET burst size B (number frames in a second, 1–2000) and payload size P (46–1500, min.–max. Ethernet payload size). The simulation time is set to 60 s, and each scenario is repeated ten times. During the simulation, the ET transmitter randomly selects a burst occurrence time every 20 s from the start of the simulation. Then, it uses a uniform real distribution to randomly generate B frames with P size at selected times. The flows of all other traffic types are the same as previous as in Table II. Note that when P is 1500 bytes and B is 2000, link utilization exceeds 100% in the worst case, where some losses are inevitable.

ET Performance With Bursty ET: Fig. 10 plots the latency, jitter, and throughput performance of ET from the simulations when we introduce bursty ET with varying burst and frame sizes. Based on the same analysis as we have done so far, the estimated worst case end-to-end latency (E-latency$_{e2e}$) of ET is 1013.465 μs when P is 1500 bytes. The estimated worst case E-latency$_{e2e}$ of ET decreases by 200 μs whenever P is reduced by 500 bytes. Fig. 10(b) shows that the measured maximum latency of ET has increased slightly when B is greater than 1, but is still lower than theoretical worst case E-latency$_{e2e}$ in (1). Moreover, even if B increases from 500 to maximum of 2000, the latency [Fig. 10(a)] and jitter [Fig. 10(c)] are both relatively uniform. This implies that the increase in latency (from $B = 1$) is only due to self-interference among ET frames and *eTAS* is capable of supporting ET without any interference from other traffic types. We can also see from Fig. 10(d) that the average throughput of ET increases appropriately in proportion to P and B for each scenario as desired. These results show that *eTAS* can handle ET bursts of 2000 frames per second, and can guarantee low-latency regardless of burst size without performance degradation.

ST Performance With Bursty ET: Fig. 11 plots the performance of ST with *eTAS* according to the burstiness of ET. The results show that as P and B increase, the worst case latency and jitter of ST increase gradually [Fig. 11(b) and (c)]. This is because ST frames may need to wait for back-to-back frames of bursty ET flow, and the intensity increases as P and B increase. However, although max. latency of ST

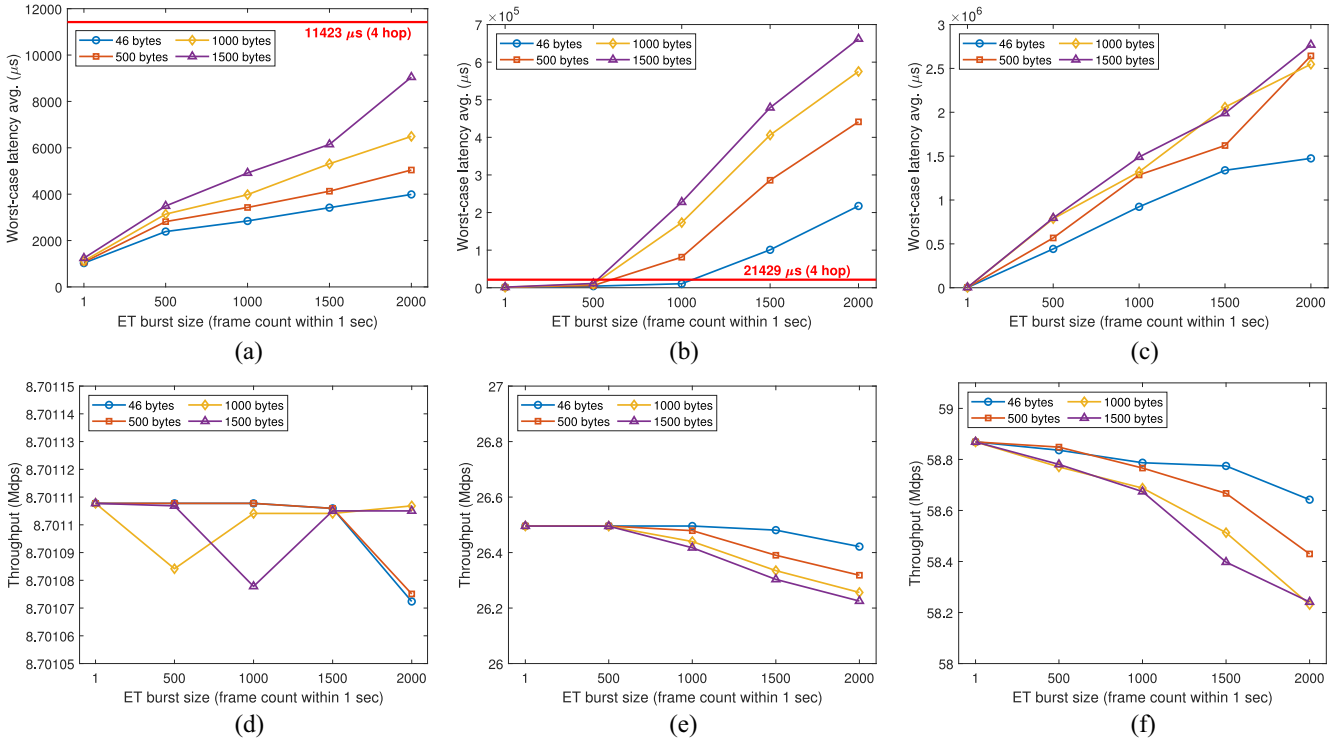

Fig. 12. NST performance results with varying ET burst and frame sizes. (a) Worst case end-to-end latency (AVB-A). (b) Worst case end-to-end latency (AVB-B). (c) Worst case end-to-end latency (BE). (d) Throughput average (AVB-A). (e) Throughput average (AVB-B). (f) Throughput average (BE).

is measured as high as ≈1634 μs when P is 1500 bytes and B is 2000 (when link utilization is 100 %), it is still significantly (≈ 83 %) lower than the *ET-in-ST&NST* case where only one ET per second is transmitted. The worst case jitter (≈ 66 μs) is also ≈ 97% lower than the *ET-in-ST&NST* case. For the throughput of ST [Fig. 11(d)], there are slight decreases of up to ≈26 b/s (note the y-axis scale) when P is over 1500 bytes and B is over 1000. However, this is only an instantaneous and tiny decrease due to ET's burstiness, and ST regains its performance back after the burst of ET passes through. Eventually, every frame of the ST is delivered to the destination without accumulation in the queue.

NST Performance With Bursty ET: So far, we have confirmed that *eTAS* guarantees better performance for ET as well as ST compared to what can be achieved by standard TAS. Finally, Fig. 12 plots the results for NST flows, those that are not protected by TAS nor *eTAS* by design. First, the latency of AVB-A is affected by the increases in P and B, but the worst case latency of all configurations is sufficiently lower than the requirement in the AVB standard [red horizontal line in Fig. 12(a)]. The throughput of AVB-A may decrease slightly as in Fig. 12(d), but the difference from *w/o-ET* is only up to ≈35 b/s in all scenarios, which is negligible in the context of audio traffic. On the other hand, the performance of AVB-B and BE traffic is significantly affected by both P and B. In particular, latency of AVB-B is higher than the target value for maximum E-latency$_{e2e}$ of AVB-B [red horizontal line in Fig. 12(b)] when P is 1000 bytes or higher. The throughput of AVB-B and BE also exhibits a linear decrease as in Fig. 12(e) and (f).

The reason is obvious; we have increased the total bandwidth utilization by introducing large amounts of ET frame bursts, resulting in significant congestion. Recall that the link utilization exceeds 100% in the worst case (P = 1500 bytes, and B = 2000). As we increase the link utilization, lower priority traffic, such as AVB or BE, will have less chance of being selected for the next transmission. This is a typical starvation problem in priority-based queueing and transmission selection methods. Among AVB-A, AVB-B, and BE, AVB-A is affected the least because it has the highest traffic class among the NST flows. Nevertheless, the AVB standard requires performance guarantees only when the bandwidth utilization is below 75 % [34], [35], and no requirements beyond that. *eTAS* achieves ultralow-latency for ET while still satisfying the requirements of TSN for ST and AVB-A traffic even in highly congested (near 100%) scenarios.

VI. Related Work

Although the IEEE 802.1Qbv amendment defines how packet transmissions are scheduled within a switch using the gating mechanism, it does not define nor suggest how the GCL schedules are computed for the switches in the network. Because TAS scheduling is classified as an NP-complete problem, there has been several prior studies in the literature that attempts to address or alleviate the scheduling problem and guarantee real-time performance.

Craciunas *et al.* [11], Craciunas and Oliver [12] proposed formal constraints for TAS scheduling optimization by identifying and analyzing the factors affecting real-time

communication. Then, static schedules are computed using the proposed constraints with the satisfiability modulo theory [10]. Ansah *et al.* [13] proposed a schedulability algorithm to identify whether the frames in time-triggered applications can be scheduled, and whether bridge schedules can be computed. Jin *et al.* [20] proposed a heuristic algorithm for scheduling massive data transmissions. It uses satisfiability modulo theory and optimization modulo theory, but relaxes the TAS scheduling constraints and eliminates scheduling conflicts by combining schedule tables and packet injection control. The "No-wait packet scheduling scheme" [15] proposes a heuristic algorithm that enables scheduling of 1500 flows by adapting the Tabu search algorithm, and also a compression algorithm that minimizes the wasted time due to guard bands (GBs) by controlling the open times of transmission gates. "Fault-tolerance scheduling" [21] aims to guarantee that critical messages meet their deadlines when they need to be retransmitted due to a fault by taking into account both critical and noncritical messages when scheduling time windows. In addition, some studies propose synthesizing schedules for time-critical traffic under mixed-criticality industrial applications where time-critical ST and audio/video traffic co-exist [23]–[27].

Real-world network environments change dynamically due to reasons, such as node join and leaves, device failure, and reconfiguration. In these cases, the network devices must adaptively recalculate their schedules accordingly to ensure QoS, but this is extremely difficult, complex, and costly compared to static scheduling. To address this challenge, Alnajim *et al.* [28] proposed the incremental QoS-aware path selection and scheduling without time-slot (SWOTS) algorithm in order to preserve QoS and eliminate queueing delay even in the event of network environment changes. Nayak *et al.* [16] proposed an ILP based dynamic/incremental scheduling scheme for time-sensitive software-defined networks, which schedules and controls periodic flow transmissions on the network edge rather than on the switches. The work most closely related to ours is a study by Nasrallah *et al.* [29], which proposes adaptive bandwidth sharing (ABS) and adaptive slotted window (ASW) schemes. ABS allows the transmission of frames in the closed queues to achieve higher utilization when the queues with open gates are all empty in a specific time window. ASW shifts the ST-to-BE gating ratio according to runtime network statistics received from the receiver to upstream switches in order to adjust the latency achieved by high-priority traffic.

However, all of the aforementioned prior work assumes periodic/isochronous traffic with known datarates (i.e., packet size and periodicity) when scheduling, and none of them discuss sporadic and nonisochronous but time-sensitive ET with unknown datarates, nor their impact on prescheduled flows. *eTAS* is the first work and a novel scheme that effectively handles time-sensitive event traffic with minimal impact on ST traffic, which distinguishes itself from prior work.

VII. Conclusion

We proposed *eTAS* that seamlessly supports sporadic and nonisochronous time-critical ET in the time-sensitive network (TSN) without impacting regular and prescheduled time-critical isochronous traffic (ST). *eTAS* dynamically assigns the highest available traffic class to ET, allows the transmission of ET across all time windows, and temporarily and dynamically extends the ST time window to minimize the impact of ET on ST. Our extensive evaluation on OMNET++ simulator with a diverse set of scenarios and configurations shows that *eTAS* significantly reduces the end-to-end latency and jitter of ET while eliminating the significant impact ET had on ST when using the standard TAS algorithm. We believe that our work provides a reference to improve the IEEE 802.1Qbv TAS standard in TSN to be more robust and reliable. As our future work, we plan to implement *eTAS* and conduct experiments on real TSN switches as soon as appropriate hardware is available.

References

[1] "IEEE Time-Sensitive Networking Task Group." 2017. [Online]. Available: http://www.ieee802.org/1/pages/tsn.html (Accessed: Aug. 2020).

[2] "IEEE 802.3 Working Group." 2020. [Online]. Available: https://www.ieee802.org/3/ (Accessed: Aug. 2020).

[3] *IEEE Standard for Ethernet, Rev. IEEE Std. 802.3-2015*, IEEE Standard 802.3-2018, pp. 1–5600, 2018.

[4] "IEEE 802.1 Working Group." 2021. [Online]. Available: https://1.ieee802.org/ (Accessed: Apr. 2021).

[5] *IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks—Amendment 25: Enhancements for Scheduled Traffic*, IEEE Standard 802.1Qbv-2015, pp. 1–57, 2016.

[6] *IEEE Standard for Local and Metropolitan Area Networks—Timing and Synchronization for Time-Sensitive Applications, Rev. IEEE Std. 802.1AS-2011*, IEEE Standard 802.1AS-2020, pp. 1–421, 2020.

[7] *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, pp. 1–208, 2018.

[8] C. Simon, M. Maliosz, and M. Mate, "Design aspects of low-latency services with time-sensitive networking," *IEEE Commun. Stand. Mag.*, vol. 2, no. 2, pp. 48–54, Jun. 2018.

[9] W. Steiner, S. S. Craciunas, and R. S. Oliver, "Traffic planning for time-sensitive communication," *IEEE Commun. Stand. Mag.*, vol. 2, no. 2, pp. 42–47, Jun. 2018.

[10] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, "Scheduling real-time communication in IEEE 802.1Qbv time sensitive networks," in *Proc. 24th Int. Conf. Real-Time Netw. Syst.*, 2016, pp. 183–192.

[11] S. S. Craciunas, R. S. Oliver, and W. Steiner, "Formal scheduling constraints for time-sensitive networks," 2017, *arXiv:1712.02246*.

[12] S. S. Craciunas and R. S. Oliver, "An overview of scheduling mechanisms for time-sensitive networks," in *Proc. Real-Time Summer School LÉcole dÉté Temps Réel (ETR)*, 2017, pp. 1551–3203.

[13] F. Ansah, M. A. Abid, and H. de Meer, "Schedulability analysis and GCL computation for time-sensitive networks," in *Proc. IEEE 17th Int. Conf. Ind. Informat. (INDIN)*, vol. 1. Helsinki, Finland, 2019, pp. 926–932.

[14] N. G. Nayak, F. Dürr, and K. Rothermel, "Time-sensitive software-defined network (TSSDN) for real-time applications," in *Proc. 24th Int. Conf. Real-Time Netw. Syst.*, 2016, pp. 193–202.

[15] F. Dürr and N. G. Nayak, "No-wait packet scheduling for IEEE time-sensitive networks (TSN)," in *Proc. 24th Int. Conf. Real-Time Netw. Syst.*, 2016, pp. 203–212.

[16] N. G. Nayak, F. Dürr, and K. Rothermel, "Incremental flow scheduling and routing in time-sensitive software-defined networks," *IEEE Trans. Ind. Informat.*, vol. 14, no. 5, pp. 2066–2075, May 2018.

[17] F. Smirnov, M. Glaß, F. Reimann, and J. Teich, "Formal timing analysis of non-scheduled traffic in automotive scheduled TSN networks," in *Proc. Design Autom. Test Eur. Conf. Exhibit. (DATE)*, Lausanne, Switzerland, 2017, pp. 1643–1646.

[18] W. Steiner, "An evaluation of SMT-based schedule synthesis for time-triggered multi-hop networks," in *Proc. 31st IEEE Real-Time Syst. Symp.*, San Diego, CA, USA, 2010, pp. 375–384.

[19] A. A. Atallah, G. B. Hamad, and O. A. Mohamed, "Fault-resilient topology planning and traffic configuration for IEEE 802.1Qbv TSN networks," in *Proc. IEEE 24th Int. Symp. On-Line Testing Robust Syst. Design (IOLTS)*, Platja d'Aro, Spain, 2018, pp. 151–156.

[20] X. Jin *et al.*, "Real-time scheduling of massive data in time sensitive networks with a limited number of schedule entries," *IEEE Access*, vol. 8, pp. 6751–6767, 2020.

[21] R. Dobrin, N. Desai, and S. Punnekkat, "On fault-tolerant scheduling of time sensitive networks," in *Proc. 4th Int. Workshop Security Depend. Critical Embedded Real-Time Syst. (CERTS)*, 2019, pp. 1–13.

[22] F. Heilmann and G. Fohler, "Size-based queuing: An approach to improve bandwidth utilization in TSN networks," *ACM SIGBED Rev.*, vol. 16, no. 1, pp. 9–14, 2019.

[23] V. Gavriluţ and P. Pop, "Scheduling in time sensitive networks (TSN) for mixed-criticality industrial applications," in *Proc. 14th IEEE Int. Workshop Factory Commun. Syst. (WFCS)*, Imperia, Italy, 2018, pp. 1–4.

[24] V. Gavriluţ, L. Zhao, M. L. Raagaard, and P. Pop, "AVB-aware routing and scheduling of time-triggered traffic for TSN," *IEEE Access*, vol. 6, pp. 75229–75243, 2018.

[25] A. A. Atallah, G. B. Hamad, and O. A. Mohamed, "Multipath routing of mixed-critical traffic in time sensitive networks," in *Proc. Int. Conf. Ind. Eng. Other Appl. Appl. Intell. Syst.*, 2019, pp. 504–515.

[26] V. Gavriluţ and P. Pop, "Traffic-type assignment for TSN-based mixed-criticality cyber-physical systems," *ACM Trans. Cyber-Phys. Syst.*, vol. 4, no. 2, pp. 1–27, 2020.

[27] F. Smirnov, M. Glaß, F. Reimann, and J. Teich, "Optimizing message routing and scheduling in automotive mixed-criticality time-triggered networks," in *Proc. 54th ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Austin, TX, USA, 2017, pp. 1–6.

[28] A. Alnajim, S. Salehi, and C.-C. Shen, "Incremental path-selection and scheduling for time-sensitive networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Waikoloa, HI, USA, 2019, pp. 1–6.

[29] A. Nasrallah *et al.*, "Performance comparison of IEEE 802.1 TSN time aware shaper (TAS) and asynchronous traffic shaper (ATS)," *IEEE Access*, vol. 7, pp. 44165–44181, 2019.

[30] J. Falk, F. Dürr, and K. Rothermel, "Exploring practical limitations of joint routing and scheduling for TSN with ILP," in *Proc. IEEE 24th Int. Conf. Embedded Real-Time Comput. Syst. Appl. (RTCSA)*, Hakodate, Japan, 2018, pp. 136–146.

[31] D. Hellmanns, A. Glavackij, J. Falk, R. Hummen, S. Kehrer, and F. Dürr, "Scaling TSN scheduling for factory automation networks," in *Proc. 16th IEEE Int. Conf. Factory Commun. Syst. (WFCS)*, Porto, Portugal, 2020, pp. 1–8.

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

[33] *IEEE Standard for Local and Metropolitan Area Networks—Audio Video Bridging (AVB) Systems*, IEEE Standard 802.1BA-2011, pp. 1–45, 2011.

[34] *IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks Amendment 14: Stream Reservation Protocol (SRP), Rev. IEEE Std. 802.1Q-2005*, IEEE Standard 802.1Qat-2010, pp. 1–119, 2010.

[35] *IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks Amendment 12: Forwarding and Queuing Enhancements for Time-Sensitive Streams*, IEEE Standard 802.1Qav-2009, p. C1-72, 2010.

[36] *IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks—Amendment 34: Asynchronous Traffic Shaping*, IEEE Standard 802.1Qcr-2020, pp. 1–151, 2020.

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

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

[39] "Time Sensitive Networks for Flexible Manufacturing Testbed-Description of Converged Traffic Types." Industry IoT Consortium. 2018. [Online]. Available: https://www.iicon sortium.org/pdf/IIC_TSN_Testbed_Char_Mapping_of_Converged_Traffic_Types_Whitepaper_20180328.pdf (Accessed: Aug. 2020).

[40] S. Thangamuthu, N. Concer, P. J. L. Cuijpers, and J. J. Lukkien, "Analysis of Ethernet-switch traffic shapers for in-vehicle networking applications," in *Proc. Design Autom. Test Eur. Conf. Exhibit. (DATE)*, Grenoble, France, 2015, pp. 55–60.

[41] D. Maxim and Y.-Q. Song, "Delay analysis of AVB traffic in time-sensitive networks (TSN)," in *Proc. 25th Int. Conf. Real-Time Netw. Syst.*, 2017, pp. 18–27.

[42] "OMNeT++." [Online]. Available: https://omnetpp.org/ (Accessed: Aug. 2020).

[43] "INET Framework." [Online]. Available: https://inet.omnetpp.org/ (Accessed: Aug. 2020).

[44] "Communication over real-time ethernet group (CoRE4INET)." [Online]. Available: https://core-researchgroup.de/projects/simulation.html (Accessed: Aug. 2020).

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

[46] J. Lee and S. Park, "Time-sensitive network (TSN) experiment in sensor-based integrated environment for autonomous driving," *Sensors*, vol. 19, no. 5, p. 1111, 2019.

[47] H.-T. Lim, D. Herrscher, and F. Chaari, "Performance comparison of IEEE 802.1Q and IEEE 802.1 AVB in an Ethernet-based in-vehicle network," in *Proc. IEEE Int. Conf. Comput. Technol. Inf. Manage. (NCM ICNIT)*, Seoul, South Korea, 2012, pp. 1–6.

Moonbeom Kim received the B.S. degree in computer and information communications engineering from Hongik University, Seoul, South Korea, in 2017, and the M.S. degree in computer science and engineering from Chung-Ang University, Seoul, in 2020, where he is currently pursuing the Ph.D. degree in computer science and engineering.

He is a Research Assistant with the Networked Systems Laboratory led by Dr. J. Paek, with research interests in wireless networking, localization, and time-sensitive networking.

Doyeon Hyeon received the B.S. degree from the School of Integrative Engineering, Chung-Ang University, Seoul, Republic of Korea, in 2020, where she is currently pursuing the M.S. degree with the Department of Computer Science and Engineering.

She is also a Research Assistant with the Networked Systems Laboratory led by Dr. J. Paek, with research interests in time-sensitive networking.

Jeongyeup Paek (Senior Member, IEEE) received the B.S. degree in electrical engineering from Seoul National University, Seoul, South Korea, in 2003, and the M.S. degree in electrical engineering and the Ph.D. degree in computer science from the University of Southern California, Los Angeles, CA, USA, in 2005 and 2010, respectively.

He worked as a Research Intern with Research and Development Labs USA, Deutsche Telekom Inc., Washington, DC, USA, in 2010, and then joined Cisco Systems Inc., San Jose, CA, USA, in 2011, where he was a Technical Leader with the Internet of Things Group, Connected Energy Networks Business Unit (CENBU, formerly the Smart Grid BU). In 2014, he was with the Department of Computer Information Communication, Hongik University, Seoul, as an Assistant Professor. He has been an Associate Professor with the School of Computer Science and Engineering, Chung-Ang University, Seoul, since 2015.

Dr. Paek is on the Editorial Board of *Journal of Communications and Networks* (JCN) and *Sensors*. He is an ACM member.