TSN-VM: A Real-time and Distributed Algorithm for Scheduling-Violation Mitigation in Time-Sensitive Networking

Boyang Zhou \(^1\) and Liang Cheng \(^2\)

\(^1\)Lehigh University
\(^2\)Affiliation not available

December 7, 2023

Abstract

Time-Sensitive Networking (TSN) is designed to provide deterministic performance for real-time systems, which usually have hard delay requirements for Time-Triggered (TT) traffic. In order to guarantee deterministic delays, TSN scheduling assigns specific time slots to TT frames for their transmissions by TSN switches on their paths. However, several factors may lead to scheduling violations, meaning that TT frames are not sent as scheduled. Scheduling violations can cause the increment of delays experienced by TT frames, which may breach their delay requirements.

Existing TSN scheduling methods cannot deal with scheduling violations. In our research, we propose TSN-Violation Mitigation (TSN-VM), which dynamically reconfigures the schedule at each TSN switch distributedly on a periodic basis, to mitigate the occurrence of scheduling violations. TSN-VM is built upon our new real-time detection mechanism to discover local violation information in each port of TSN switches. TSN-VM is the first real-time algorithm that can deal with scheduling violations in TSN. Its convergence conditions are demonstrated in the paper. Using TSN-VM, we can reduce the number of lag violations by more than 90.74\% and decrease the average and median delays by more than 95.57\% in the simulated avionic system and industrial Ethernet compared with systems without TSN-VM.
TSN-VM: A Real-time and Distributed Algorithm for Scheduling-Violation Mitigation in Time-Sensitive Networking

Boyang Zhou  
Electrical and Computer Engineering Department  
Lehigh University  
Bethlehem, US  
boz319@lehigh.edu

Liang Cheng  
Electrical Engineering and Computer Science Department  
University of Toledo  
Toledo, US  
liang.cheng@utoledo.edu

Abstract—Time-Sensitive Networking (TSN) is designed to provide deterministic performance for real-time systems, which usually have hard delay requirements for Time-Triggered (TT) traffic. In order to guarantee deterministic delays, TSN scheduling assigns specific time slots to TT frames for their transmissions by TSN switches on their paths. However, several factors may lead to scheduling violations, meaning that TT frames are not sent as scheduled. Scheduling violations can cause the increment of delays experienced by TT frames, which may breach their delay requirements.

Existing TSN scheduling methods cannot deal with scheduling violations. In our research, we propose TSN-Violation Mitigation (TSN-VM), which dynamically reconfigures the schedule at each TSN switch distributedly on a periodic basis, to mitigate the occurrence of scheduling violations. TSN-VM is built upon our new real-time detection mechanism to discover local violation information in each port of TSN switches. TSN-VM is the first real-time algorithm that can deal with scheduling violations in TSN. Its convergence conditions are demonstrated in the paper. Using TSN-VM, we can reduce the number of lag violations by more than 90.74% and decrease the average and median delays by more than 95.57% in the simulated avionic system and industrial Ethernet compared with systems without TSN-VM.

Index Terms—Time-sensitive networking, Distributed scheduling and monitoring, Real-time reconfiguration

I. INTRODUCTION

As an emerging technology, Time-Sensitive Networking (TSN) is widely applied to real-time systems [1]–[3]. TSN is designed to provide deterministic service for real-time systems, in which there is Time-Triggered (TT) traffic with hard delay requirements to guarantee the timeliness of critical data. TT traffic usually contains safety-critical, or time-critical information [4], and it has the highest priority among all different types of traffic in TSN networks.

TSN offers the deterministic service to TT traffic by scheduling deterministic time slots for the transmissions of TT frames. However, several factors may induce scheduling violations, which cause TT frames to be sent in the wrong time slots. The occurrence of such violations impacts the scheduling in TSN, leading to the breach of delay requirements. Note that even if the scheduling violations are caused by temporary issues, they will lead to a sustained effect on the system, which is discussed in Section III. In our paper, we design TSN-Violation Mitigation (TSN-VM) to reduce the occurrence of and the impact caused by scheduling violations. When uncertainties happen in TSN networks, the deterministic service in TSN will be violated and cannot be recovered until the uncertainties are eliminated. Even though TSN-VM cannot guarantee deterministic service under uncertainties, it can resolve the sustained effect of scheduling violations and dramatically reduce the packet delay (See Section III and Section VII).

Existing TSN scheduling methods [5]–[17] lack mechanisms to deal with scheduling violations. To the best of our knowledge, this is the first paper on real-time mitigation for scheduling violations in TSN. There has been a paper focusing on the mitigation of scheduling violations in TSN using DDPG [18]. However, it does not provide real-time mitigation of violations. There is also a paper focusing on monitoring the violations [19], which proposes a centralized approach that induces extra delays. Zhang et al. Meanwhile, the detection algorithm in [19] cannot detect lag violations in real-time.

The main contributions of our research are summarized as follows.

- We design TSN-Violation Mitigation (TSN-VM), which updates the TSN scheduling periodically in a distributed way to reduce the number of scheduling violations in TSN networks based on local information.
- We prove the convergence and discuss the convergence conditions of TSN-VM.
- We propose a distributed per-packet detection mechanism that can detect local violation information in each output port of TSN switches, which is then used by TSN-VM.
- Compared with the traditional scheduling methods, TSN-
VM can significantly decrease the number of scheduling violations by more than 90.74% and reduce the median and average delays by more than 95.57% in the simulated avionic system and industrial Ethernet.

In this paper, Section II introduces the background knowledge and related work of our research. Section III identifies two research problems of the TSN scheduling-violation mitigation. Our solutions are described in Section IV, which discusses a distributed mechanism for violation detection, and Section V, which discusses how TSN-VM works. Section VI analyzes the performance and convergence of TSN-VM. Section VII shows the evaluation results of TSN-VM. Section VIII concludes the paper.

II. BACKGROUND KNOWLEDGE AND RELATED WORK

In this section, we introduce the structure of TSN switches, including the QoS mechanism used in TSN and the scheduling methods of TSN. We also discuss factors that can impact the scheduling in TSN. Moreover, a detailed discussion on the classification of violations is provided in this section.

Figure 1: Gating and transmission selection in a TSN-switch port using TAS

A. TSN

TSN is developed by the IEEE TSN task group to support real-time applications with guaranteed delay bounds for TT traffic. It uses Time-Aware Shaper (TAS), which separates the communication in TSN in repeating cycles. TAS performs per-queue-based scheduling. It realizes deterministic delay by allocating time slots for the transmission of TT traffic in each output port on its path. There are eight queues corresponding to eight different priorities in TSN switches. Each queue has a gate status used to control whether the frame of this queue can be transmitted. The gate statuses in the switch are controlled by Gate Control Lists (GCLs) [20].

Each item in GCLs contains a time instance t and a bit vector b. t defines the changing time of gate statuses, and b shows the new gate statuses of eight queues at the time instance. If the ith bit of the bit vector b[i] is 1, it means that the gate of the ith queue is open, and the frames in the ith queue can be transmitted. Otherwise, the gate of the queue is closed. When several non-empty queues open their gates at the same time, a selection algorithm is used to choose which frame should be transmitted. The selection algorithm usually implements strict priority scheduling in TSN. TT Gate Control Vector (GCV) in Figure 1 is discussed in Section V-A.

Figure 1 depicts the gating and transmission selection in the output port of TSN switches [21]. TT traffic has the highest priority in TSN and is followed by Audio Video Bridging (AVB) flows. Credit-Based Shaper (CBS) [22] is used to balance the transmission of two types of AVB flows. In this paper, since we focus on TT traffic, we do not need to take CBS into consideration because it is only used for AVB flows.

B. Scheduling in TSN

The main objective of the scheduling in TSN is to find all requirements of TT traffic. Since TT traffic is periodic, TSN defines a hyperperiod, which is the least common multiple of all periods of TT flows. All TT flows have the same traffic patterns in different hyperperiods. If a scheduling result can satisfy all requirements in one hyperperiod, it should work for all hyperperiods.

Existing work for the scheduling in TSN can be classified into two categories. The first category of the existing work solves optimization problems generated based on the traffic pattern, delay and jitter requirements, and switch capacity [5]–[12]. A feasible solution to the optimization problem indicates that all requirements can be fulfilled theoretically.

Since solving an optimization problem is time-consuming, the second category of TSN scheduling works on real-time reconfiguration for GCLs [13]–[17]. Mostly, these methods use heuristic algorithms for real-time scheduling [13]–[16]. Huang et al. [17] propose an incremental routing and scheduling (IRAS) algorithm to schedule new TT traffic based on the current scheduling to reduce time consumption. However, real-time reconfiguration in these methods is used for adapting to new traffic patterns in TSN. When there is no new TT traffic coming, GCLs are still static. They do not solve the same research problems as those in this paper, which is mitigating scheduling violations in TSN.

Solutions to the TSN scheduling contain the offsets of flows and the durations and locations of windows in each switch in a hyperperiod. The definition of a window is the duration when a gate is open. A queue can only transmit frames in its windows. Offsets of flows define the sending time of frames at their sources. Each TT frame is assigned a window in each output port on its path for the transmission. All existing scheduling methods provide static GCLs if the traffic pattern is unchanged and do not deal with scheduling violations.

C. Causes of Scheduling Violations

The scheduling in TSN provides a theoretical solution to meet all the requirements of TT traffic. However, things may not work as expected. There are several factors that may impact the scheduling in TSN, causing violations, including time-synchronization errors in TSN networks, transmission jitters at hosts, and dynamics of the scheduling in the switch fabric as investigated by [19].

Time-synchronization error causes the drift of windows in different switches. TSN utilizes Precision Time Protocol (PTP)
to implement time-synchronization [23]. PTP can achieve a time-synchronization error in the order of nanoseconds. However, master failures can happen in PTP, which causes asynchronization among clocks [24].

The scheduling in TSN determines the offsets of different flows. The offset of a flow refers to the frame sending time at its source host in each period. However, the Direct Memory Access (DMA) process with PCI-E may bring up to 10 μs jitter for a frame. This leads to the dynamics in the arrival time of frames at Network Interface Card (NIC) [25] influencing the sending time of frames.

Dynamics of the scheduling in the switch fabric is another factor, which can impact the scheduling in TSN. Current TSN switches usually contain a crossbar to connect all their ports [26]. When several frames arrive at different input ports at the same time and compete for the same output port, the priorities of these frames are determined by the scheduling method in the switch fabric. A well-known scheduling algorithm for the switch fabric is iSLIP [27]. In iSLIP, round-robin arbitration is used to decide the priority of the frames. If an output port receives several requests from different input ports, it will grant the one that has the highest priority in the round-robin schedule. Thus, iSLIP cannot guarantee the order of frames when they come from different input ports at the same time. All these factors can cause the emergence of scheduling violations, which cannot be mitigated using static GCLs.

D. Types of Scheduling Violations

Without considering the packet loss in TSN, there are two types of scheduling violations in TSN networks. When a frame is sent in a window later than its scheduled window, a lag violation occurs. If a frame is sent before its scheduled window, we note it as a lead violation.

Figure 2 shows an example of lead and lag violations. Figure 2 (a) depicts the topology and traffic pattern. The network contains three hosts, which are H1, H2, and H3, two TSN switches S1 and S2, and three flows f1, f2, and f3. Figure 2 (b) illustrates that all frames are sent in their assigned windows. Figure 2 (c) depicts the scenario where frame 3 encounters a jitter, leading to its arrival earlier than frame 2 at S1. Thus, frame 3 occupies the time slots in windows w1 and w6. Lead violations occur because frame 3 is transmitted before its assigned windows. Meanwhile, frame 2 encounters lag violations because it is transmitted in windows w2 and w4.

III. RESEARCH PROBLEMS

In this section, we mainly discuss (i) the necessity of our research and (ii) the research problems we solve in this paper. In Section II, we have discussed the causes and types of scheduling violations in TSN. Scheduling violations are caused by uncertainties in the TSN network. In some scheduling methods in TSN [10], [11], they will reserve redundancies for the uncertainties in each window. However, it is not guaranteed that users have access to evaluate all uncertainties in TSN networks, and sometimes unexpected events may happen, such as the unavailability of master clocks [24]. Thus, it is impossible to guarantee that scheduling violations will not occur. Meanwhile, the scheduling violations are destructive due to their cascading and sustained effects.

The cascading and sustained effect of the scheduling violation refers to situations where a lag violation can lead to a chain of violations in the network. If a frame misses its assigned windows, becoming a lag violation, it requires some later time slots for its transmission. However, it is likely that these time slots have already been assigned to other TT frames. Therefore, more TT frames are lagged because their time slots are occupied by previous lagged frames, leading to more lag

Fig. 2: An example of scheduling violations [18].
violations. Note that these lag violations happen in the same switch/output port. Moreover, if a frame encounters a lag violation in one switch, it is more likely that the same frame also encounters lag violations in its downstream switches, causing more violations in other switches.

Even if a lag violation is caused by a temporary issue, its cascading effect is sustained if we do not allocate a new time slot to the lagged frame as it will still occupy other frames' time slots. Figure 2 (d) shows an example of the sustained effect of lag violations. In Hyperperiod 1, frame 2 encounters a jitter, causing its transmission delayed to window \( w_2 \). Thus, frame 3 of Hyperperiod 1 is delayed to Hyperperiod 2. In Hyperperiod 2, frame 2 is sent at the expected time from \( H_1 \). However, frame 2 still encounters a lag violation because frame 3 from the previous hyperperiod occupies \( w_1 \). Frame 3 in Hyperperiod 2 still lags to the next hyperperiod. Thus, there will be two lag violations happening in each hyperperiod forever after Hyperperiod 1. This phenomenon is the sustained effect of lag violations. It is destructive because a temporary issue may cause forever lag violations.

Lag violations are more destructive than lead violations because lead violations do not have cascading and sustained effects if they do not cause lag violations. Moreover, lag violations have a larger impact on the delay performance than lead violations since the lead violations guarantee that the lead frames must be sent in or before their assigned hyperperiods. Thus, we set the mitigation of lag violations as the goal in TSN-VM.

TSN can actively drop frames/packets to reduce lag violations without adjusting its current scheduling. However, there are several drawbacks caused by dropping. First, dropping frames will cause the permanent loss of the information. If dropped frames contain safety-critical information, such as a fire alarm, dropping them leads to severe safety issues. Second, dropping frames may not be necessary. Even though uncertainties ruin the deterministic service provided by TSN, it is still possible that all frames can arrive before their deadlines. For example, even though frame 2 is delayed by one window in the example shown in Figure 2 (c), it is still possible that frame 2 encounters a smaller delay than its requirement. In this case, there is no need to drop any frames. Third, dropping frames may cause a loss of flows. Still take the example in Figure 2 (c), if the jitter of frame 3 lasts forever, in order to mitigate lag violations encountered by frame 2, all frames from \( f_3 \) should be dropped, meaning that \( f_3 \) is totally lost in TSN. Last, dropping packets cannot fundamentally solve the problem. If the uncertainties are too large so that all frames are lagged, dropping does not help.

Based on the previous discussion, two research problems are formulated due to scheduling violations in TSN networks: i) How to detect scheduling violations happening in the network? ii) How to mitigate the cascading and sustained effect caused by scheduling violations without using pure dropping? We propose solutions to the two problems in Section IV and Section V.

IV. Violation Detection

In this section, we introduce our distributed detection mechanism used to discover violations occurring in each output port. The local violation information is the prerequisite for using TSN-VM. Two types of violations should be detected, which are lead violations and lag violations.

Figure 3 shows the detection mechanism that we design to discover scheduling violations. The mapping between TT frames and their scheduled windows is stored in the local memory of each switch. The space complexity of the window depends on the number of TT frames in each hyperperiod. To look up the assigned window of a TT frame, we should know the flow ID and the sequence number of the frame. Once a frame arrives at the switch, the frame parser can extract its flow ID and sequence number. Then, the switch can search for its scheduled window \( w_1 \) from the mapping table. When a frame is transmitted, the switch records the sending window \( w_2 \). Knowing \( w_1 \) and \( w_2 \), we can detect scheduling violations by comparing the two windows.

The detection of lead violations is straightforward. If \( w_1 \geq w_2 \), it means that a lead violation occurs. However, lag violations in a hyperperiod might not be able to be detected directly in real-time because frames of lag violations may not be sent in the scheduled hyperperiod. Note that we need to collect all the violation information of all windows by the end of each hyperperiod because TSN-VM needs the information to update GCLs in each hyperperiod. For example, suppose a frame is scheduled to be sent in hyperperiod \( i \). If it encounters a lag violation, it might be sent in hyperperiod \( j \), where \( j > i \). In this case, this lag violation can only be detected at hyperperiod \( j \) if we use the comparison of \( w_1 \) and \( w_2 \) to discover lag violations. Then, TSN-VM will miss this lag violation for \( w_1 \) when it updates GCLs at the end of hyperperiod \( i \). In the centralized monitoring system proposed in [19], Zhang et al. simply detect lag violations by comparing \( w_1 \) and \( w_2 \), which influences the real-time performance of lag violation detection.

In order to solve this problem, we use a counter to record the number of frames sent correctly in each window of each hyperperiod. If \( w_1 \) equals \( w_2 \), we can add the counter corresponding to \( w_1 \) by 1. Using these counters, we can know the number of frames sent correctly in each window. Since we have the mapping table between TT frames and windows, the number of frames assigned to each window is known. The number of lag violations of a window can be calculated by using Equation 1. In the equation, \( \text{num}_{\text{lag}} \) is the number of lag violations of the window, \( \text{num}_{\text{sche}} \) is the number of frames scheduled in the window, \( \text{num}_{\text{corr}} \) is the number of frames sent correctly in the window, and \( \text{num}_{\text{lead}} \) is the number of lead violations of the window.

\[
\text{num}_{\text{lag}} = \text{num}_{\text{sche}} - \text{num}_{\text{corr}} - \text{num}_{\text{lead}} \tag{1}
\]

If \( w_1 < w_2 \), it means that a lag violation occurs. Since the number of lag violations of the window is calculated using Equation 1, we just record the sending window \( w_2 \) to help find bounds for the right and left boundaries of windows in TSN-VM. Since our detection mechanism does the per-packet inspection, every packet should be classified into lag violations, lead violations, or correctly sent packets.
The limitation of our detection algorithm is that it does not take packet loss into consideration. Considering packet loss, Equation 1 should be converted to

\[ \text{num\_lag} + \text{num\_loss} = \text{num\_sche} - \text{num\_corr} - \text{num\_lead}. \] (2)

Note that \text{num\_loss} is the number of lost packets. Our current detection mechanism cannot distinguish lag violations and packet loss in real-time (by the end of each hyperperiod). Thus, we treat all packet loss as lag violations to assume the most pessimistic scenarios. The real-time distinction between packet loss and lag violations will be a part of our future work.

There is another research on violation detection in TSN by Zhang et al., which is TSN-Peeper [19]. Compared with our detection mechanism, TSN-Peeper does not emphasize real-time detection. It detects lag violations by directly comparing \( w_2 \) and \( w_1 \), which does not meet the requirement of TSN-VM. Moreover, TSN-Peeper is a centralized detection algorithm that focuses on the total number of violations during the runtime, while our detection mechanism is a distributed one that emphasizes the number of violations in each hyperperiod.

V. TSN-VM

In this section, we introduce TSN-VM, which can update GCLs at the end of each hyperperiod based on the violation information. TSN-VM is an ad-hoc heuristic algorithm designed for mitigating the cascading and sustained effects caused by scheduling violations and dealing with temporary issues using a short-term memory mechanism. In this paper, we do not consider actively dropping packets in TSN-VM. However, TSN-VM can cooperate with active dropping to achieve better performance. TSN-VM can mitigate the negative effects caused by active dropping since it adjusts the scheduling of TSN, reducing the number of frames dropped. We will include the active dropping in our future work.

Since TSN requires real-time performance, heuristic algorithms might be a suitable solution for violation mitigation. However, traditional heuristic algorithms are inefficient in TSN networks. Traditional heuristic algorithms, including genetic algorithm [28], simulated annealing [29], and tabu search [30], have several limitations. First, they have a large number of random trials (TSN scheduling candidates) to test in each iteration. Inappropriate scheduling in trials can cause significant system degradation. Second, trials in existing heuristic algorithms are assumed to be independent. However, trials are not independent of each other in TSN. The result of the current trial depends on the previous trials (i.e., whether there are remaining frames in the queue at the beginning of the current hyperperiod). In order to address these limitations, we design TSN-VM that will not generate a lot of random trials. In each hyperperiod, TSN-VM only adjusts each window once based on the violation information collected by the real-time monitoring mechanism described in the previous section.

A. TT Gate Control Vector

Before introducing TSN-VM, we need to define the Gate Control Vector (GCV) for the highest priority queue (TT GCV). A TT GCV is a 0-indexed array of numbers showing the gate close periods and gate open periods of the TT gate in each hyperperiod. Suppose that TT GCV starts with a gate close period in our research. Then, the elements of the GCV with even indices indicate the gate close periods, and the elements with odd indices represent the gate open periods, which are windows, for the TT traffic queue in each hyperperiod. When a TT GCV = \([10\mu s, 20\mu s, 30\mu s, 40\mu s]\), it means that the hyperperiod is 100 \( \mu s \), and there are two windows, which have lengths of 20 \( \mu s \) and 40 \( \mu s \), in each hyperperiod.

TT GCVs can be translated to GCLs as shown in Figure 1. Note that TT GCVs only influence the first bit of each GCL item because TT GCVs are only used to control the gate status of TT traffic queues. TSN-VM periodically updates TT GCVs, and thus GCLs, to mitigate scheduling violations in TSN.

B. Three Stages of TSN-VM

Since scheduling violations may not happen in the original scheduling of TSN, we design a trigger mechanism for TSN-VM. If there is no lag violation detected in the network, TSN-VM will not start running. This trigger mechanism guarantees that TSN-VM will not interfere with the original scheduling of the network unless violations happen.

TSN-VM runs in three stages: i) moving right boundaries of windows; ii) moving left boundaries of windows; iii) relocating the locations of windows. TSN-VM updates GCVs in each output port at the end of each hyperperiod. TSN-VM moves the right boundary first because violations caused by late arrivals of frames can only be mitigated by moving the right boundary.

The first two stages are used to find the tight boundaries of a window. Tight boundaries of windows help prevent the waste of bandwidth and the occurrence of lead violations. The last stage is used to reduce the queuing delay of frames. During the first two stages, TSN-VM keeps recording upper and lower bounds for left boundaries and right boundaries of windows. However, the bound information will be cleared after stage 3 so that extreme bounds caused by temporary issues will not influence the next round running of TSN-VM. This short-term memory mechanism helps cope with changing situations or temporary issues in the network.

1) Stage 1-moving the right boundaries: We introduce the moving criterion and the stopping criterion for stage 1 of TSN-VM as follows. When there is no lag violation in the current window, and the window size is larger than its scheduled size, we move the right boundary to the left to reduce the size of the window. If the window size is larger than the threshold parameter times the scheduled size of the window, we reduce the window to 1/2 of its scheduled size. Otherwise, we decrease the window size by a fixed small amount called a moving step (1 \( \mu s \) in this paper). The moving step is determined by window sizes and the required granularity. The reason for decreasing the window size when there is no lag...
violation is that we want to find the smallest size of the window so that more bandwidth can be allocated to low-priority traffic and avoid leaving space for the lead violation occurrence.

TSN-VM maintains lower bounds for the right boundaries of windows. If a right boundary reaches its lower bound, it cannot be moved to the left anymore. For any window in each hyperperiod, if there is no frame assigned to previous windows sent in the current window, and frames assigned to the current window are lagged, the current right boundary of the window should be the lower bound under the current left boundary. Whether there are lagged frames sent in a window can be detected because we record the sending window $w_2$ of lagged frames in our detection mechanism.

TSN-VN guarantees that right boundaries should be larger than their lower bounds in stage 1. It will not reduce the window size if the new right boundary of a window after the operation is smaller than its lower bound. The lower bound of the right boundary is non-decreasing.

If there are lag violations occurring in a window, we move the right boundary to the right by half of its scheduled size so that the new window can deal with lag violations and frames left in the queue caused by lag violations in the next hyperperiod. However, there are circumstances that the next gate close period is not large enough for the increment of the window size. We can seize time sequentially from all gate close periods later than the window that needs the adjustment.

Algorithm 1 shows how to move the right boundary of a window to the right. Inputs of the algorithm are the current TT Gate Control Vector $GCV$, the index of the window $i$, the amount of movement $val$, the hyperperiod of TT traffic $hyperperiod$. Note that $i$ must be odd because only elements with odd indices refer to windows. The output of the algorithm is the new TT GCV after moving the boundary. The search for the right boundary ends at a fixed number of hyperperiods (iteration number), which is predefined.

2) Stage 2-moving the left boundaries: Stage 2 of TSN-VM is to move the left boundaries of windows. If there is no lag violation in the current window, we move the left boundary to the right by a moving step (1 $\mu$s) if the window size is larger than its scheduled size. Otherwise, we move the left boundary of the window to the left by half of its scheduled size. Note that when we move the left boundary of a window to the left, only the space in the preceding gate close period can be occupied by the window due to the dependencies between the window and its previous windows. If the preceding gate close period is smaller than half of the scheduled size of the current window, we move the left boundary to the start of the close period and set the length of the gate close period to 0. Algorithm 2 shows the algorithm of moving the left boundary of a window to the left. Note that it has the same inputs and output as those in Algorithm 1.

Similar to stage 1, we can also find the upper bound for the left boundary during the searching process. For any window in each hyperperiod, if there is no frame assigned to previous windows sent in the window, and lag violations of the window emerge, the left boundary of the window should be an upper bound under the current right boundary. When TSN-VM moves the left boundary of a window to the right to shrink the size of the window, it should be guaranteed that
Algorithm 1: Moving the right boundary to the right

Input: Index \( i \), Value \( val \), TT Gate Control Vector \( GCV \), Hyperperiod \( hyperperiod \).
Output: TT Gate Control Vector \( GCV \), i.e. the new TT Gate Control Vector after moving the right boundary.

\[\begin{align*}
    & s \leftarrow 0 \\
    & \text{for } j \leftarrow 0 \text{ to } i \text{ do} \\
    & \quad s \leftarrow s + GCV[j] \\
    & \text{if } s + val > hyperperiod \text{ then} \\
    & \quad val = hyperperiod - s \\
    & \quad GCV[i] = GCV[i] + val \\
    & \quad residue = val \\
    & \text{for } j \leftarrow i + 1 \text{ to } (GCV.length - 1) \text{ do} \\
    & \quad \text{if } j \mod 2 = 0 \text{ then} \\
    & \quad \quad \text{if } GCV[j] \geq residue \text{ then} \\
    & \quad \quad \quad GCV[j] = GCV[j] - residue \\
    & \quad \quad \text{return } GCV \\
    & \quad \quad residue = residue - GCV[j] \\
    & \quad \quad GCV[j] \leftarrow 0 \\
    & \text{for } j \leftarrow (GCV.length - 1) \text{ to } i + 1 \text{ do} \\
    & \quad \text{if } j \mod 2 = 1 \text{ then} \\
    & \quad \quad \text{if } GCV[j] \geq residue \text{ then} \\
    & \quad \quad \quad GCV[j] = GCV[j] - residue \\
    & \quad \quad \text{return } GCV \\
    & \quad \quad residue = residue - GCV[j] \\
    & \quad \quad GCV[j] \leftarrow 0 \\
    & \text{return } GCV
\end{align*}\]

Algorithm 2: Moving the left boundary to the left

Input: Index \( i \), Value \( val \), TT Gate Control Vector \( GCV \), Hyperperiod \( hyperperiod \).
Output: TT Gate Control Vector \( GCV \), i.e. the new TT Gate Control Vector after moving the left boundary.

\[\begin{align*}
    & val \leftarrow \min(GCV[i - 1], val) \\
    & GCV[i] = GCV[i] + val \\
    & GCV[i - 1] = GCV[i - 1] - val \\
    & \text{return } GCV
\end{align*}\]

the new left boundary is still lower than its upper bound. The upper bound for the left boundary is non-increasing in stage 2. There are two stopping criteria for stage 2. First, stage 2 will stop when there are lag violations happening in a window, but there is no space to increase the window size. This means that there is no enough space to deal with these lag violations under the current right boundary of the window, meaning that the converge condition for stage 2 might be violated, discussed in Section VI-B. Second, stage 2 will end at a fixed number of iterations (iteration number).

3) Stage 3 - relocate the windows: The first two stages are used to minimize the number of violations and the total size of windows. Stage 3 aims to decrease the queuing delay of TT frames. We will relocate the windows by moving the windows to the left if there is any available space in the preceding gate close periods of the windows by a moving step (1 \( \mu s \)). Once there are lag violations occurring, stage 3 of the window terminates, and TSN-VM restarts from stage 1 for the window again. Once the search of a window restarts, TSN-VM abandons the information on the bounds for the left and right boundaries of the window. This short-term memory helps to eliminate the impact of extreme scenarios caused by temporary issues.

C. Moving bounds of TSN-VM

In the first two stages, TSN-VM continuously updates the upper bound for the left boundary and the lower bound for the right boundary of all windows. However, the upper bound of the right boundary and the lower bound of the left boundary of the windows are still missing. In order to restrict the movement of windows and help filter the lagged frames, TSN-VM allows users to manually set the two bounds for the boundaries of each window based on the delay and jitter requirements of flows in TSN.

VI. PERFORMANCE ANALYSIS OF TSN-VM

In this section, we will discuss the performance of TSN-VM in two parts. The first part describes the features of TSN-VM in terms of its adaptability, overhead, and real-time performance. The second part illustrates the convergence of TSN-VM.

A. TSN-VM Features

TSN-VM has high adaptability, low overhead, and real-time features. It is a distributed algorithm that can deal with changing situations in TSN.

1) High adaptability: TSN-VM has high adaptability since it can be applied to different network topologies and traffic patterns. Users only need to modify the mapping tables between frames and windows based on TSN scheduling.

2) Real-time performance and low overhead: TSN-VM has a low overhead in terms of time complexity and network usage. The worst time-complexity of TSN-VM, which happens in Algorithm 1, is \( O(mn) \) in each hyperperiod, where \( m \) is the number of windows in each hyperperiod, and \( n \) is the length of the TT GCV for each output port. We run the worst-case scenario and test the running time of TSN-VM. Figure 4 shows how the running time varies with the number of windows in...
TABLE I: Table of Notations

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>Number of windows in one hyperperiod</td>
</tr>
<tr>
<td>$l$</td>
<td>Length of the hyperperiod</td>
</tr>
<tr>
<td>$w_{i0}$</td>
<td>Left boundary of window $i$</td>
</tr>
<tr>
<td>$w_{i1}$</td>
<td>Right boundary window $i$</td>
</tr>
<tr>
<td>$s_{i1}$</td>
<td>Scheduled left boundary of window $i$</td>
</tr>
<tr>
<td>$w_{s0}$</td>
<td>Scheduled right boundary of window $i$</td>
</tr>
<tr>
<td>$LB_i$</td>
<td>Lower bound and convergence point of the right boundary of window $i$</td>
</tr>
<tr>
<td>$UB_i$</td>
<td>Upper bound and convergence point of the left boundary of window $i$</td>
</tr>
<tr>
<td>$len_i$</td>
<td>Scheduled length of window $i$, $\text{len}<em>i = w</em>{s1} - w_{s0}$</td>
</tr>
<tr>
<td>$d_{max}$</td>
<td>Transmission time for the maximum frame size of TT traffic</td>
</tr>
<tr>
<td>$t_{ij}$</td>
<td>Transmission time of the $j$th TT frame in window $i$</td>
</tr>
<tr>
<td>$d_{ij}$</td>
<td>Latency encountered by the arrival time of the $j$th TT frame in window $i$</td>
</tr>
<tr>
<td>$s_{ij}$</td>
<td>Scheduled start time of the transmission of the $j$th TT frame in window $i$</td>
</tr>
</tbody>
</table>

the TT GCV. The machine we use is a laptop with an i7-10870H CPU and 32 GB RAM. As we can see, even when there are 100 windows in each hyperperiod, TSN-VM costs less than 4 µs in the worst case. Note that the avionic system and the industrial Ethernet examples used in the paper have at most fifteen windows in all switches. The space complexity of TSN-VM is $O(k)$, where $k$ is the number of TT frames in each hyperperiod, which is used to record the mapping information between TT frames and windows.

Moreover, we have discussed that TSN-VM only uses local violation information of each output port. Thus, no extra bandwidth and delay are induced as there is no need to transmit violation information through the network, which is required by the centralized monitoring system proposed in [19].

B. Convergence of TSN-VM

In this section, the convergence of TSN-VM is discussed. We will prove that the boundaries of windows will converge to certain values in the first two stages of TSN-VM. The proof assumes that the arrival order of TT frames is the same as the scheduled order, while the arrival time may be delayed. Suppose that there is no manually set bounds in TSN-VM in this proof.

Lemma 1. If convergence holds, the left and right boundaries of windows will converge to one of their bounds.

Proof: The boundaries of windows will keep unchanged when two conditions are reached: i) there is no lag violation in the window; ii) the boundaries of windows reach their bounds. Based on condition ii), the left and right boundaries must converge to their bounds, respectively, if convergence happens.

In stage 1 of TSN-VM, the lower bound of the right boundary of a window is updated when the window cannot transmit all frames assigned to it without interference with frames from other windows. A special lower bound $LB_i = \max(w_{s1} + \max(d_i), w_{s0} + \text{len}_i)$ of the right boundary of window $i$, where $\max(d_i)$ is the maximum delay encountered by the frame in window $i$, guarantees that there will be no lag violation if there is no lagged frame from previous windows. All frames assigned to window $i$ can be transmitted in the window even though they encounter delays when $w_{i1} \geq LB_i$. Meanwhile, there are guaranteed lag violations if $w_{i1} < LB_i$.

Theorem 1. The right boundary of window $i$ will converge to $LB_i$ under two premises, including i) $s_{ij} + d_{max} + t_{ij} \leq l$ for all $i$ and $j$; ii) $l - LB_{N-1} \geq t_{maxS}$, where $LB_{N-1}$ is the convergence value of the right boundary of the last window in each hyperperiod.

Proof: Since there are guaranteed lag violations if $w_{i1} < LB_i$, $w_{i1}$ will increase to a value greater than or equal to $LB_i$. After that, $w_{i1} - LB_i$ is always non-increasing when there is no lag violation in the window because $w_{i1}$ always decreases in this scenario. Since it is guaranteed that there will be no lag violations if there are no lagged frames from previous windows of window $i$ when $w_{i1} \geq LB_i$, the convergence condition of TSN-VM becomes that all lagged frames from previous windows should be resolved for each window. Premise i) guarantees that all frames should be able to be transmitted by the end of the hyperperiod. If it is not met, there is no way to eliminate all lag violations. Premise ii) indicates that there will always be available space to deal with at least one lag violation in each hyperperiod.

For each window $i$, when there are frames left from windows ahead of window $i$ in the queue, two scenarios might happen: i) window $i$ has enough space to transmit the frames left in the window and the frames assigned to it; ii) lag violation happens in window $i$. In the first scenario, the right boundary of window $i$ will keep moving to $LB_i$. In the second scenario, window $i$ will increase its size. In the extreme case, window $i$ can occupy the whole available space (i.e., $w_{i1} = l$) because it can seize the space of all latter windows and close periods. Because of premise ii), window $i$ can deal with at least one lagged frame from previous windows. Thus, after several hyperperiods, all lagged frames from previous windows can be eliminated. Then, the right boundary of window $i$ will converge to $LB_i$. After window $i$ converges, it is easy to prove that window $i+1$ also converges using a similar method. Thus, the convergence of TSN-VM stage 1 is proved. □

Then, we will discuss the convergence of the left boundaries of windows in stage 2 of TSN-VM.

Theorem 2. The left boundary of window $i$ will converge to $UB_i = w_{i1} - \text{len}_i$. There are two premises for the convergence of the left boundary, including i) $w_{i1} \geq LB_i$; ii) $UB_i - w_{i1-1} \geq t_{maxS}$ for all $i$, where $w_{i1-1} = 0$ when $i = 0$.

Proof: Premise i) is used to guarantee that $UB_i$ is a valid convergence value for the left boundary of window $i$. If $w_{i1} < LB_i$, TSN-VM will not converge because lag violations are guaranteed to happen in this situation. Premise ii) is used to deal with lagged frames from previous windows, which is similar to premise ii) in stage 1. The proof is similar to the one we used in stage 1. □

The premises for the convergence of TSN-VM can be met when the sum of the rate of TT frames is much lower than the bandwidth of the switch, which is usually the case by the
application design. If TT traffic consumes a large percentage of the TSN switch bandwidth, TSN-VM might fail to converge.

VII. Evaluation

In this section, we evaluate the TSN-VM in a simulated avionic system and industrial Ethernet environments. The simulation is based on the NeSTiNg project [21] in OMeT++ [31]. We integrate the scheduling of TSN and TSN-VM into the NeSTiNg project for the simulation. In the simulation, we use Length Aware Queues (LAQs) in TSN switches. LAQs calculate the maximum transferable size, which is the amount of data that can be sent during the remaining time of the window. Even though a frame is available for its transmission during a window, it will not be transmitted if the frame size is larger than the current maximum transferable size of the window. Active dropping is not integrated into TSN-VM in our evaluation. And there is no manually set bound for windows in TSN-VM.

In the evaluation, we compare the number of lag violations in the network using static GCLs and TSN-VM. Since TSN-VM has two parameters, we use TSN-VM($T$, $I$) to denote TSN-VM adopting $T$ as the threshold parameter and $I$ as the iteration number. In this section, we will evaluate the performance of TSN-VM with different parameter setups. Moreover, we study the distributions of delays of all frames in the network. We also calculate the median and average delays encountered by all frames in each mechanism so that we can evaluate the improvement in the delay performance using TSN-VM.

A. Evaluation of TSN-VM in the Avionic System

In Avionics Full-Duplex Switched Ethernet (AFDX) systems, all flows are periodic flows with maximum frame size requirements. Thus, flows in AFDX are compatible with the requirements of TT traffic in TSN. Li et al. [32] introduce an SDN-based avionics network model for TSN. Therefore, we evaluate TSN-VM in the avionic system.

Figure 5 shows the topology used for the simulation of an avionic system. Similar topologies are widely used in the delay analyses of AFDX [33]–[36]. In this topology, there are 5 switches, 10 end systems, and 11 virtual links (VLs). VLs are unidirectional paths from their sources to destinations, treated as TT flows in TSN. These VLs all meet the requirements of AFDX, having a hyperperiod of 64 ms and frame sizes from 64 bytes to 512 bytes. The bandwidth of each output is 100 Mbps, and we assume a 100 µs propagation delay for each hop. We run the simulation for 128 s, which is 2000 hyperperiods. In this evaluation, we test TSN-VM using jitters ranging from 20 µs to 100 µs. We use three setups if TSN-VM, which are TSN-VM(2, 200), TSN-VM(2, 300), and TSN-VM(3, 200).

Figure 6 shows how the number of lag violations varies with the jitter in the avionic system with static GCLs and TSN-VM. As shown in the figure, the network using static GCLs always experiences the most violations because static GCLs do not have the ability to deal with any uncertainties happening in TSN networks. There is an increment of lag violations when the jitter is 40 µs because there are several TT frames not lagged when the jitter is less than 40 µs. After that, all frames are lagged. The number of total lag violations is around 40000 in the network with static GCLs.

Avionic systems using TSN-VM dramatically outperform static GCLs in terms of the occurrence of lag violations in the network. Networks using TSN-VM(2, 200), TSN-VM(2, 300), and TSN-VM(3, 200) can decrease the number of lag violations by 90.74%, 91.60%, and 93.76% compared with the network using static GCLs on average in all scenarios with different jitters, respectively. As we can see, TSN-VM(3, 200) has the best performance of mitigating the lag violations in the avionic system. The reason for it is that a larger threshold gives TSN-VM more space to manipulate in this scenario. Sometimes the number of lag violations might decrease when the jitter increases in Figure 6. This phenomenon is caused by two reasons. First, there might be less time in stage 3 when the jitter is large, leading to more time in stage 1 and stage 2. Since stage 1 and stage 2 help mitigate lag violations, it is possible that frames with a larger jitter encounter fewer lag violations. Second, a larger jitter may cause a further increase in the sizes of windows in the first two stages, resulting in fewer lag violations.

We also compare the end-to-end delays of frames because the aim of mitigating violations is to guarantee the delay performance of the system. Figure 7 shows the delay distribution...
of frames using four mechanisms when the jitter is 100 $\mu$s. As shown in the figure, most frames have more than 64 $ms$ when using TSN-VM with static GCLs. Since the hyperperiod of the system is 64 $ms$, this means that most of the frames are delayed by at least one hyperperiod. Some frames are even delayed by six hyperperiods because their delays are close to 400 $ms$. In this scenario, data in all these frames expire because all frames have less than 1 $ms$ scheduled delay in the system. Thus, significant system degradation might happen.

Networks with TSN-VM have much better performance in terms of delays compared with the system using static GCLs. As we can see in the figure, most frames have less than 1 $ms$ delays in all three setups of TSN-VM. Only a small amount of frames have delays larger than 64 $ms$. Frames are lagged by at most one hyperperiod with TSN-VM.

Table II shows the average and median delays of the four mechanisms. All three setups of TSN-VM have good performance in terms of the average and median delay compared with the system using static GCLs. When using TSN-VM, we observe that there is a large gap between the average delay and the median delay. This is caused by the bipolar delay distribution in the system (most frames experience less than 1 $ms$ delays, but some have more than 64 $ms$ delays). Using TSN-VM, the average delay can be reduced by more than 95.57%, and the median delay can be reduced by more than 99.35% compared with the avionic system using static GCLs.

### B. Evaluation of TSN-VM in Industrial Ethernet

In this part, we evaluate TSN-VM in a simulated industrial Ethernet, PROFINET, which is an industrial Ethernet standard. TSN is also integrated with PROFINET in the PROFINET specification V2.4. We use the topology from the PROFINET topology white paper [37] as shown in Figure 8 to be the industrial Ethernet example in this section.

Different from the avionic system, industrial Ethernet has much faster switches. We will use switches that support 1 Gbps capacity for each output in the industrial Ethernet example. The propagation delay is 10 $\mu$s. In this scenario, there are 30 flows between the controller and devices. These flows have a hyperperiod of 1 $ms$, and frame sizes vary from 125 to 1500 bytes.

We use the same metric used in the avionic system to evaluate the performance of TSN-VM. Figure 9 shows how the number of lag violations varies with the jitter, ranging from 20 $\mu$s to 100 $\mu$s using static GCLs and different setups of TSN-VM. As shown in the figure, we can draw a similar conclusion to that of the avionic system: i) almost all frames are lagged in the system using static GCLs, leading to about 216000 lag violations. Different from the avionic system, all frames are lagged even when the jitter is 20 $\mu$s; ii) TSN-VM can significantly reduce the number of lag violations; iii) when the jitter increases, the number of lag violations might decrease in systems using TSN-VM because of the reasons discussed in Section VII-A. iv) TSN-VM $(2, 300)$ performs the best in terms of the occurrence of lag violations in most cases. The reason for it is that a larger $I$ can provide enough iterations for TSN-VM to converge and a longer time in the first two stages, leading to fewer lag violations. Systems using TSN-VM reduce at least 95.18% lag violations on average in all scenarios with jitters from 20 $\mu$s to 100 $\mu$s compared with the system using static GCLs.

Figure 10 shows the delay distribution of all frames in the network using four mechanisms when the jitter is 100 $\mu$s. With TSN-VM, most frames encounter less than 0.13 $ms$ delay, which is the maximum scheduled delay for all TT frames in the system. Without TSN-VM, almost all frames are lagged, and some frames are lagged more than 80 hyperperiods. This is unacceptable in the real-time industrial system.

Table III displays average and median delays in Profinet using different mechanisms. TSN-VM exhibits smaller discrepancy between average and mean delays compared to the avionic system. This is due to the industrial Ethernet having
Fig. 9: The number of lag violations varies with the jitter in the PROFINET example

Fig. 10: End-to-end delay distribution when the jitter is 100 $\mu$s in the PROFINET example

The number of lag violations varies with the jitter in the PROFINET example. In Profinet, if a frame lags by one hyperperiod, it incurs a maximum delay increment of 2 ms, while the avionic system can have a delay increment of up to 128 ms. TSN-VM significantly reduces the average delay in industrial Ethernet by over 97.71% compared to static GCLs, with the mean delay reduced by more than 99.77%

TABLE III: Average and median delays of four mechanisms when the jitter is 100 $\mu$s in the PROFINET example

<table>
<thead>
<tr>
<th></th>
<th>Static GCLs (2, 200)</th>
<th>TSN-VM (2, 200)</th>
<th>TSN-VM (3, 200)</th>
<th>TSN-VM (2, 300)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average delay</td>
<td>38.96</td>
<td>0.11</td>
<td>0.10</td>
<td>0.11</td>
</tr>
<tr>
<td>Median delay</td>
<td>36.03</td>
<td>0.08</td>
<td>0.07</td>
<td>0.08</td>
</tr>
</tbody>
</table>

Based on the simulation results, we conclude that TSN-VM can significantly reduce the number of lag violations and the average and median delays compared with static GCLs in both avionic systems and industrial Ethernet. TSN-VM can mitigate the sustained and cascading effect caused by lag violations and prevent most TT frames from expiration.

VIII. CONCLUSION

Even though TSN has been designed to support real-time applications with guaranteed delay bounds for TT traffic, factors such as time-synchronization errors, frame transmission dynamics, and scheduling dynamics in the switch fabric, may induce violations in TSN scheduling. In our research, to the best of our knowledge, we have designed the first real-time and distributed algorithm TSN-VM to reduce the number of violations and improve the delay performance in TSN networks based on the local violation information discovered by the distributed detection mechanism discussed in Section IV and Section V.

We have also proved the real-time performance and the convergence of TSN-VM in the paper. TSN-VM can reduce the number of lag violations and delays in the simulated avionic system and industrial Ethernet by more than 90.74%. Meanwhile, the average and median delays decrease by more than 95.57%. Using TSN-VM, the influence of the occurrence of violations can be significantly mitigated.

REFERENCES
