Schedule-Based Attack Against TSN TAS with Frame Preemption

Omolade Ikumapayi
University of Colorado, Colorado Springs, CO, USA

Vijay Banerjee
Washington State University, Pullman, WA, USA

Sena Hounsinou
Metro State University, St Paul, MN, USA

Gedare Bloom
University of Colorado, Colorado Springs, CO, USA

Abstract

Time-Sensitive Networking (TSN) achieves deterministic communication using mechanisms like Time-Aware Shaping, which manages the timing of network traffic streams using a Gate Control List (GCL). The GCL operates according to a cyclic schedule by opening and closing gates for priority (egress) queues in out-bound ports. However, the cyclic schedule in the GCL introduces potential security vulnerabilities to schedule-based attacks. This type of attack exploits TSN's deterministic schedules to manipulate traffic flow and can impact availability and safety. Traditional intrusion detection systems (IDS) are commonly employed in TSN to detect malicious activities by monitoring traffic patterns and bandwidth usage. However, schedule-based attacks can align malicious packets with legitimate traffic, making the attack more stealthy and harder to detect by rate-based IDS. This paper presents a novel schedule-based attack that synchronizes malicious traffic to exploit predictability in TSN's GCL schedule. This attack causes an adversarial blocking in which low-priority traffic delays higher-priority traffic without being detected using a rate-based IDS. We demonstrate the feasibility and impact of this schedule-based attack on TSN with off-the-shelf hardware.

2012 ACM Subject Classification Security and privacy → Network security; Computer systems organization → Real-time system architecture

Keywords and phrases schedule-based attack, time-sensitive networking, real-time system, adversarial blocking

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2026.12

Funding This work is supported by NSF CNS-2046705 and Colorado Bill SB18-086.

1 Introduction

The IEEE Time-Sensitive Networking (TSN) extends Ethernet's capabilities by introducing precise timing synchronization, traffic scheduling, and resource reservation mechanisms. TSN is pivotal for modern industrial automation and vehicular networks, for which timely and deterministic communication is paramount. With TSN, network devices (i.e., end nodes) assign priorities to transmitted frames. Traffic shaping mechanisms such as Time-Aware Shaper (TAS) achieve deterministic timing by controlling message frame transmission using time slots. TAS enforces transmission schedules using a Gate Control List (GCL) that opens and closes gates similar to how livestock stampedes are prevented using gated fences. These gates sit between priority queues, which store pending message frames, and the media access control (MAC) layer, which handles the transmission of message frames on the physical medium. Thus, the gates manage frame transmission by ensuring that only designated priority queues connect to the MAC and transmit within GCL-specified cyclic time slots.

Using frame preemption introduces two additional scheduling classes: preemptable MAC (pMAC) frames and express MAC (eMAC) frames. When an eMAC frame arrives for transmission at the MAC layer during the transmission of a pMAC frame, the MAC suspends the pMAC frame and begins to transmit the eMAC frame instead. After the eMAC frame completes, fragmented pMAC frames resume upon gate closure, which can delay higher-priority pMAC traffic, causing preemption-induced blocking and deadline misses [1, 13]. TAS mitigates blocking for eMAC by inserting guard bands before scheduled transmissions, depending on configuration, but none are used within pMAC transmissions.

Although the focus of TSN is to ensure safety by guaranteeing that real-time deadlines can be met, the expanded scope of communication makes TSN a vulnerable target for security threats [8, 24]. One relatively underexplored type of vulnerability and threat involves TSN's schedule predictability. This predictability largely stems from the use of statically configured, periodically repeating GCLs, through which deterministic latency guarantees are typically realized in practical TSN deployments. Standard TAS analyses assume known traffic and do not account for malicious or unforeseen traffic injection [4]. The ability to schedule frame transmission based on priorities and preemption to support real-time communications creates an opportunity for schedule-based attacks (SBAs) that leverage predictability to exploit a real-time system [17].

The effectiveness of SBAs lies in adversaries' ability to observe, analyze, and predict the timing characteristics of specific traffic flows in real-time networks [12]. During the schedule exploitation phase of an SBA, malicious packets are strategically aligned with normal traffic schedules in a manner that integrates with expected network behavior while causing secondary effects, i.e., achieving the attacker's objectives, such as selective denial-of-service, spoofing, or information leakage. While TSN mechanisms such as Per-Stream Filtering and Policing (PSFP) can restrict unauthorized or non-conformant traffic, schedule inference may still be possible from a compromised but otherwise legitimate end node.

In this paper, we introduce a novel SBA that exploits the GCL schedule and the frame preemption mechanism with TAS under specific TSN configurations. We show that an attacker with control of a node on a TSN segment can estimate the time slots – gate durations and cycle times – in a GCL schedule. The attacker can then exploit these time slots by synchronizing low-priority frames to transmit during the slots reserved for high-priority traffic, thereby strategically injecting frames at a critical time. We show that this injection can cause adversarial blocking, by which a malicious lower-priority pMAC frame, preempted by an eMAC frame, blocks a higher-priority pMAC frame. This blocking results in unintended queuing delays, thus disrupting TSN's real-time guarantees. This allows the attacker to conduct a denial-of-service attack with limited privileges and low network observability, because the attacker's traffic appears consistent (indistinguishable) from that of a benign node. Even with an intrusion detection system (IDS) in place, malicious traffic goes undetected because it exploits preemption timing rather than traffic rates.

Our contributions are as follows.

1. We show how an attacker can infer GCL scheduling information through passive network observations and timing analysis.
2. We introduce a novel SBA that uses the knowledge of the GCL to induce adversarial blocking.
3. We evaluate the effectiveness of the SBA in the presence of a rate-based IDS.
4. We identify potential mitigation strategies to counteract SBAs in TSN.

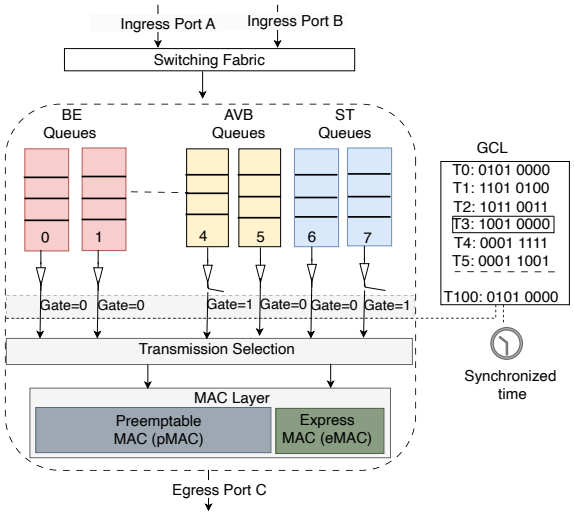


Figure 1 Switch Queue Model.

2 Background

The IEEE 802.1Qbv defines the TSN protocol for rapidly delivering critical communications with minimal queueing delay for reliable transmission. The 802.1Qbv standard utilizes a TAS implemented on the egress port to organize communication into repetitive cycles of fixed duration. This setup divides time into fixed-duration cycles and variable-length sequences, each allocated to a specific traffic class [5]. This ensures efficient handling of different types of traffic, such as Scheduled Traffic (ST), Best effort (BE), and AVB traffic (see Figure 1). Each entry in the GCL includes the state of the Time-Aware gates, either 0 or 1, to denote whether a gate is closed or opened, and the duration for which the entry remains active. This ensures precise control over each traffic queue's transmission timing and duration. If the gate for a queue is open, Ethernet frames are transmitted. Meanwhile, frames waiting in other queues will be considered for transmission based on the state of their respective gates. The Precision Time Protocol (PTP) maintains synchronization across all network devices, ensuring time slots are precisely aligned across switches and end nodes.

2.1 Queueing and Transmission Delays

During the transmission, each frame faces overheads in the physical layer due to the frames' queuing according to the IEEE 802.1Qbv standard. The propagation delay δ_{prop} is the time required for a signal to travel across a physical link between nodes. For instance, this delay can be estimated during transmission using the PTP, as described by Shrestha et al. [20], by leveraging the synchronization messages exchanged between the master and slave nodes. Each message carries timestamps that allow for the estimation of the delay. The processing delay δ_{proc} includes the time required by network devices to examine and process the packet headers. δ_{proc} can be estimated by measuring the round-trip time (RTT) using a TCP connection [19]. This approach isolates the processing delay by ensuring minimal queuing through an empty queue configuration. The queuing delay δ_{q_i} of the j-th frame of time-aware

traffic flow with the GCL configuration in place, as discussed by Thiele et al. [23], is derived by adding the blocking experienced due to a closed gate and same-priority frames in the queue waiting for transmission.

2.2 Frame Preemption

The IEEE 802.1Qbu frame preemption is designed to prioritize low-latency delivery of critical data by allowing high-priority frames to interrupt and preempt lower-priority transmissions. Each frame class is directed to the express and preemptable queues, respectively, as shown in Figure 1. Frames in the same class cannot preempt each other. This non-preemption between frame classes introduces the potential for blocking, where a lower-priority frame can delay a higher-priority frame *even when the lower-priority frame's gate is closed*, due to the resumption of a previously preempted pMAC fragment. Preemption overhead δ_{prem} incurs 24 bytes per event and introduces additional delay; preemptable frames may experience multiple preemptions, increasing overall transmission time. The worst-case blocking for a preemptable frame in flow f_ι occurs when the largest frame from a lower-priority stream within the same preemption class begins transmission immediately before a frame of f_ι, resulting in maximum blocking [22]. When such blocking recurs across consecutive cycles, its impact may accumulate at the flow level. In multi-switch networks, preemption can cause delays in frame arrival at subsequent switches if a delay occurs in any one switch.

Preemption operates in two modes [1]. In the preemption-with-hold mode, the pMAC is placed in a hold state, preventing new preemptable frame transmissions within the final 123 bytes before a gate boundary, thereby creating a guard band for timely eMAC transmission. In the preemption-without-hold mode, the pMAC remains in the release state and may initiate transmission subject to scheduling; upon arrival of an express frame, an ongoing preemptable transmission is preempted, fragmented, and later resumed. IEEE 802.1Qbv also enforces a gate admissibility check before transmission; however, under preemption, completion time may be uncertain due to interruptions by eMAC traffic. Frames are assumed to satisfy the admissibility check at transmission start; however, completion may still extend beyond the anticipated window, resulting in spillover. In practice, guard bands prevent new pMAC frames from starting near gate boundaries but do not constrain the resumption of preempted fragments. We therefore treat this behavior as implementation-dependent, as the standard leaves some room for interpretation.

3 Motivating Example

Schedule-based attacks exploit predictable scheduling patterns, enabling adversaries to time malicious actions using fixed or known schedules. According to Nasri et al. [17], such attacks typically begin once an attacker gains control of a task within the system scheduler, allowing unauthorized operations within a constrained execution window. This execution window is critical, as it determines when the attacking task must execute relative to the victim task to cause disruption.

In TSN, attackers can observe network behavior to gather insights into the GCL schedules and other timing details by observing network behavior. This knowledge enables attackers to inject malicious frames during the execution window of a high-priority frame, introducing delays that compromise network performance. The remainder of this paper evaluates the feasibility and impact of such attacks in TSN, focusing on scenarios where the attack is timed to execute just before the victim's task.

Table 1 Flow Configuration for Motivating Example.

Flow	Type	GCL Slot (μs)	Period (μs)
f_1	eMAC	0–15, 50–65, 100–115, 150–165	40
f_v	pMAC	15–30, 65–80, 115–130, 165–180	50
f_a	pMAC	30–50, 80–100, 130–150	N/A

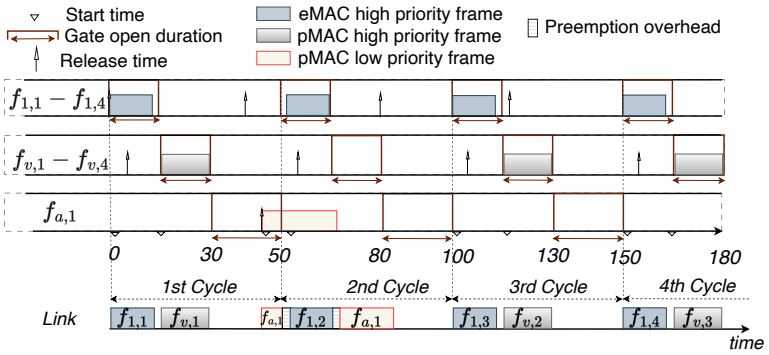


Figure 2 Example of a TSN SBA: the pMAC victim's frame (f_v) is delayed until the next cycle.

Consider a simple TSN configuration with three *network flows* (formally defined in Section 4.1) as shown in Table 1. These three flows comprise one eMAC flow (f_1) and two pMAC flows: the victim's flow (f_v) and the attacker's flow (f_a). f_v has a higher priority than f_a. The cycle time is $50\mu s$, and all frames are assumed to have an implicit deadline equal to their period. The GCL slot indicates the times during which that flow's gate is open, i.e., its queue is allowed to access the MAC. The flows have the following parameters: f_1 generates four frames with a period of $40\mu s$, and its gate is opened for a duration of $15\mu s$ in each cycle. The victim's pMAC flow f_v generates four frames with a period of $50\mu s$ (at times 5, 55, 105, and 155 μs in the example). The gate is opened for a duration of $15\mu s$ in each cycle. The attacker's pMAC flow f_a frames are strategically injected into its queue slots, with the gate opened for $20\mu s$.

Suppose that an attacker has observed the network over time and has gained knowledge of the GCL schedule, i.e., the times at which each transmission queue is allowed to connect to the MAC. (We show how in Algorithm 1.) Figure 2 illustrates an attack scenario that exploits the GCL schedule. During the first cycle, $f_{1,1}$ arrives at time 0 and transmits immediately in gate $[0, 15]$; followed by $f_{v,1}$, which transmits with gate slot $[15, 30]$. The attacker strategically waits until time 45 to release frame $f_{a,1}$ within its slot $[30, 50]$. This timing ensures that $f_{a,1}$ gets preempted by $f_{1,2}$, which is a scheduled frame already queued when its gate opens in the second cycle at $[50, 65]$. Thus, when $f_{1,2}$ begins transmitting, it preempts $f_{a,1}$ at time 50. Consequently, although $f_{v,2}$ arrived at time 55 before its gate-open slot, it must wait for the fragmented $f_{a,1}$ to finish, delaying its transmission and causing it to miss its slot. $f_{1,3}$ transmits after within its slot $[100, 115]$, when its gate opened while $f_{v,2}$ further waits until its next opening at $[115, 130]$ in the third cycle. As a result, $f_{v,2}$ transmits at time 115, missing its deadline. This deferral shifts the victim flow by one cycle, causing all subsequent frames of f_v to miss their intended transmission cycle and therefore miss their deadline; for example, $f_{v,3}$ transmits at 165. This example demonstrates how carefully timed low-priority frames can exploit GCL timing to cause adversarial blocking, disrupt the intended TSN schedule, and violate the system's schedulability assumptions.

Table 2 Table of Notation.

Variable	Definition
f_ι	The ι-th flow
O_ι	The ι-th observer flow
$f_{\iota,j}$	jth frame of flow f_ι
C_ι	Transmission time of each frame in f_ι
T_ι	Period of each frame in f_ι
D_ι	Relative deadline of all frames in f_ι
ρ_ι	Priority of frame
S	Payload in bytes
q_i	i-th queue
N	Number of observer frames per queue
d_i	Estimated gate duration for q_i
t	Cycle time across all queues

4 System and Attack Model

4.1 System Model

We consider a network consisting of interconnected nodes, including end devices, egress queues, and multi-port switches with full-duplex links connected to the Ethernet physical layer that facilitate communication across various network segments at a link speed. The network is synchronized using PTP, so all nodes share a common time base. The frames are sorted into $n = 8$ queues at the egress port, denoted as $Q = \{q_0, q_1, \ldots, q_7\}$, where q_i represents the i-th queue, and larger values of i denote higher priority, hence q_7 has the highest priority. The scheduling algorithm selects the frame from the head of the highest priority non-empty queue in a first-in, first-out (FIFO) order for transmission.

A source node generates traffic flows and each flow f_ι, $\forall\iota \geq 0$ is parameterized as $\{C_\iota, T_\iota, D_\iota, Src_\iota, Dest_\iota\}$ where C_ι is the transmission time, T_ι is the period and D_ι is the relative deadline. Every frame in a flow inherits these parameters. The j'th frame of flow f_ι is denoted $f_{\iota,j}$, and it passes through a number of links (switches) from the flow's source to its destination. A typical GCL entry encodes the gate period duration as a 32-bit value. For example, with 8 entries, a cycle time of 1 ms (1000 μs) is divided among the entries.

4.2 Attack Model

The attacker aims to induce additional blocking in preemptable traffic by injecting malicious traffic into pMAC queues while remaining undetected. Stealth is achieved by limiting the injection rate below detection thresholds. The attack assumes coexistence of eMAC and pMAC queues, with attacker and victim flows assigned to pMAC and preemption enabled without strict Hold/Release protection.

We assume that the adversary has compromised a single legitimate node within the network (e.g., Node B in Figure 3), allowing them to generate and manipulate the release of traffic streams from that node [21]. This is a strong assumption, as PSFP can prevent unauthorized transmissions. By injecting flows into transmission windows, the adversary can analyze the timing behavior of its traffic at the switch egress. The adversary can also observe the latency of its frames to the immediate receiver, possibly through echo or timestamped frames.

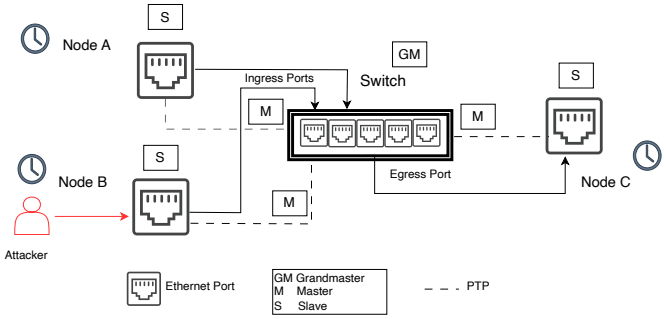


Figure 3 Attack Model.

By analyzing these observations, the adversary can infer traffic timing behavior. While they may not directly manipulate high-priority traffic, they can generate traffic with any priority level. In this paper, we show how this ability to infer traffic information can be exploited by injecting targeted congestion that introduces adversarial blocking, affecting the timely delivery of higher-priority pMAC traffic. The attacker can estimate end-to-end delay of its own frames (e.g., via RTT or timestamping). A single compromised node is sufficient for both probing and injection.

5 TSN Schedule-Based Attack

This section describes how an adversary launches an SBA to induce blocking in TSN. The schedule inference is heuristic and may be sensitive to timing variability. The attacker sends observer frames over multiple cycles to measure end-to-end delays and infer the GCL schedule (Algorithm 1). The periodic nature of the GCL enables reconstruction of transmission windows and identification of critical flows.

Using this knowledge, the attacker injects a malicious frame at a precise time so it partially transmits before the gate closes, blocking the higher-priority victim frame (Algorithm 2). The attack succeeds only if the malicious frame is queued during an open gate slot and transmits long enough to be preempted. The following steps are needed to execute the attack:

1. The attacker initially observes network traffic over multiple cycles using observer flows (O_ι). By analyzing the latency of these flows, the attacker estimates the cycle time and identifies the latest moment a frame from O_ι can be transmitted within a given cycle (Section 5.1).
2. The attacker validates the estimation by probing subsequent cycles to confirm or refine the GCL schedule reconstruction (Section 5.2).
3. The attacker injects malicious flows (f_a) into a low-priority queue to align with critical time slots, triggering preemption (Section 5.3). These flows are preempted by high-priority traffic, resulting in partial transmission and blocking of victim flows (f_v) during their allocated time. The injection is repeated for subsequent instances of f_v.

5.1 Initial Estimation Steps

To begin, the attacker aims to estimate the cycle time and the available transmission slots for a priority queue. We have assumed that an attacker could estimate the end-to-end delay from when the frames were sent. For example, some frames, such as PTP or probe frames,

include timing information that can reveal the delay experienced. However, to determine how long a specific priority queue gate was closed, the attacker needs to isolate and estimate the queuing delay from the end-to-end delay.

5.1.1 Estimation of Cycle time

The effective transmission time over a TAS cycle equals the cycle duration t, i.e., $\sum_{i=0}^{|Q|-1} d_i = t$ when gate intervals do not overlap, where d_i denotes the interval during which a specific queue q_i can transmit based on the GCL, assuming a single transmission window per queue per cycle. When multiple gates overlap, only one transmission occurs and the overlapping interval is counted once toward t. To estimate d_i, for each q_i, observer frames are sent at time $T_{i,j} = T_0 + j \cdot \tau$, where T_0 is the offset, τ represents the transmission interval, and $j = 0 \ldots N-1$ is the index of frames sent to q_i. These frames are sent periodically to measure their end-to-end delay $L_{i,j}$. The absolute residual delay $\delta_{r_{i,j}}$ represents the waiting time at the switch egress queue prior to transmission. If $L_{i,j}$ is measured over a single hop from the switch egress to the receiver, $\delta_{r_{i,j}}$ is given by:

$$\delta_{r_{i,j}} = L_{i,j} - (C_\iota + \delta_{\text{proc}} + \delta_{\text{prop}}) \tag{1}$$

where δ_{prop} and δ_{proc} denote the propagation and processing delays.

$\delta_{r_i,j}$ may include the preemption delay due to interruption by a higher-priority eMAC frame. However, because preemption events occur dynamically at runtime, it is difficult to determine whether and when a frame was preempted solely from the observed delay. Blocking by frames of the same priority is considered a queuing delay. According to IEEE 802.3br, frames smaller than 123 bytes cannot be preempted. Setting the observer frame size below 123 bytes ensures $\delta_{r_{i,j}}$ only reflects time spent in the queue, excluding preemption delays. Frames experience minimal queuing delay when the gate is open and increased delay when the gate is closed. Increased queuing delay may also occur under heavy traffic load, but such variation is not fixed, whereas delays due to gate closures exhibit periodic patterns across cycles.

To detect gate behavior, residual times are grouped into windows per queue, where each window contains consecutive frames with similar queuing delay, corresponding to the same gate opening. Similarity is determined using a threshold $T_{\text{diff},i}$. $T_{\text{diff},i} = \mu_{\{\Delta\delta_{r_i,j}\}} + \sigma_{\{\Delta\delta_{r_i,j}\}}$:

$$\mu_{\{\Delta\delta_{r_i,j}\}} = \frac{1}{N-1} \sum_{j=1}^{N-1} (\delta_{r_i,j+1} - \delta_{r_i,j}) \tag{2}$$

$$\sigma_{\{\Delta\delta_{r_i,j}\}} = \sqrt{\frac{1}{N-2} \sum_{j=1}^{N-1} \left((\delta_{r_i,j+1} - \delta_{r_i,j}) - \mu_{\{\Delta\delta_{r_i,j}\}} \right)^2} \tag{3}$$

where $\mu_{\{\Delta\delta_{r_i,j}\}}$ and $\sigma_{\{\Delta\delta_{r_i,j}\}}$ denote the average successive difference of $\delta_{r_i,j}$ and its deviation, respectively. Once $T_{\text{diff},i}$ is known, the time windows can be formed: the attacker initializes the time window using the first observer frame's residual delay $\delta_{r_i,0}$ as $W_{i,k} = \{\delta_{r_i,0}\}$ where k is the index of the current window being formed. Subsequent frames are added to the current window if the difference between consecutive residual delays is not above $T_{\text{diff},i}$.

$$W_{i,k} \leftarrow W_{i,k} \cup \{\delta_{r_i,j}\}, \quad \text{if } |\delta_{r_i,j} - \delta_{r_i,j-1}| \leq T_{\text{diff},i} \tag{4}$$

When the difference between two consecutive residual delays exceeds the threshold, it indicates a possible gate closure event or a significant change in traffic conditions. A new window $W_{i,k+1}$ for q_i is then created. As a result, $T_{\text{diff},i}$ accounts for the observed $\delta_{r_i,j}$ variability. The value segments the windows and differentiates gate closures from normal traffic variations.

As time windows are formed for each queue, they are stored in $\mathcal{W}[(i,k)]$, representing the observed transmission window for queue q_i at index k. For each window, the earliest and latest observed departures are then identified as $T_{\min,i,k}$ and $T_{\max,i,k}$, respectively.

We note that the attacker does not need to hit the exact moment a gate opens again after a cycle. They only need to time their frame within a range close enough to ensure the frames can transmit during the targeted slot and can be preempted by an express frame.

By computing the differences $\Delta T_{\min,i,k} = T_{\min,i,k+1} - T_{\min,i,k}$ and $\Delta T_{\max,i,k} = T_{\max,i,k+1} - T_{\max,i,k}$, we capture the range of possible cycle durations. Since the difference between consecutive minimum values, $\Delta T_{\min,i,k}$, can sometimes be larger than the difference between maximum values, $\Delta T_{\max,i,k}$, depending on the observed data, we compare both values in the t range. This ensures that we correctly identify the smallest ($t_{\min}$) and largest ($t_{\max}$) observed intervals, providing an estimation of the minimum and maximum possible cycle durations. The minimum interval reflects the shortest recurring cycle period, while the maximum interval accounts for potential variations due to network conditions and bounds the cycle time range. We then select the estimated minimum and maximum cycle time across the available queues:

$$t = [\min\left(t_{min}, \min(\Delta T_{\min,i,k}, \Delta T_{\max,i,k})\right),\ \max((t_{max}, \max(\Delta T_{\min,i,k}, \Delta T_{\max,i,k}))] \tag{5}$$

The gate duration is then calculated as the difference between the maximum and minimum delays observed within each window $W_{i,k}$. The transmission time of the last frame is added to account for transmission within the open slot. To determine the maximum gate duration observed across the cycles, the largest value across all windows is selected:

$$d_i \leftarrow \max(d_i, C_{max,i,k} + (T_{\max,i,k} - T_{\min,i,k})) \tag{6}$$

We identify two main causes of overestimation, firstly due to the remaining transmission time after the gate closes. While d_i is bounded by the configured gate open slot, any frame arriving after the gate closes cannot be transmitted until the next cycle. Although the observer frames are too small to be preempted, if one of them is still transmitting when the gate closes, its remaining transmission time contributes to the observed delay and is added to the gate duration estimate. This addition is expected to have minimal impact. Secondly, when multiple queues share a common slot, observer frame delays may overlap within the same estimation window, and multiple queues may be assigned to the same gate-open window. Let q_m denote another queue that potentially shares the same gate schedule as q_i. We verify this by comparing their estimated gate-close times. If $|\delta_i - \delta_m| \leq T_{\text{diff},i}$, then q_i and q_m are inferred to share the same gate window in cycle k, and their observations are merged into a single window, $W_k = W_{i,k} \cup W_{m,k}$. Note that $T_{\text{diff},i}$ is computed only when δ_{r_i} contains more than two values; otherwise, the corresponding queues are grouped within the same window.

▶ **Example 1.** Here, we illustrate the approach for estimating the GCL cycle time and gate opening duration. Consider three priority queues q_7, q_6, q_2 which handle both normal traffic flows and observer flows (see Figure 4). As a result, observer flows experience queuing delays due to contention. The GCL is the same as the motivating example (Table 1), but q_2 now includes flow f_2 generating a frame every $65\mu s$. The eMAC flow f_1 and pMAC flow

Table 3 Calculated Time Values by the Attacker.

Queue	$T_{\text{diff},i}$	$W_{i,k}$	d_i	$\Delta T_{\text{min},i,k}$	$\Delta T_{\text{max},i,k}$
q_7	36	{8}, {50, 61, 63}	2+(63-50)=15	50-8=42	63-8= **55**
q_6	32	{26, 28}, {65, 78}	2+(78-65)=15	65-26=39	78-28=50
q_2	26	{46, 48}, {80, 82}	2+(80-82)=4	80-46=**34**	82-48=34

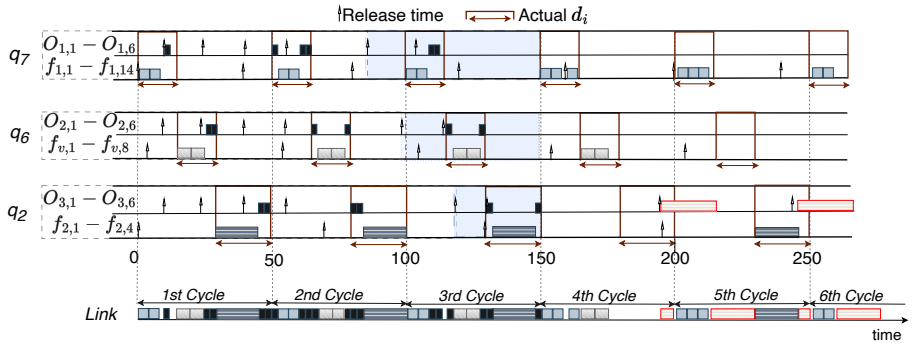


Figure 4 Attacker frames in the schedule.

f_v send two frames every $40\mu s$ and $50\mu s$, respectively. Each O_ι is under 123 bytes with $C_\iota = 2\mu s$ each to avoid being preempted. We assume the frames were released to the switch immediately in this example. The remaining $\delta_{r_i,j}$ observed is the time released to the switch and queued. The attacker observes the following delays using Eq. 1: $\delta_{r_7} = [8, 50, 61, 63]$, $\delta_{r_6} = [26, 28, 65, 78]$, and $\delta_{r_2} = [46, 48, 80, 82]$. As shown in Table 3, using $T_{\text{diff,i}}$, each $W_{i,k}$ is segmented, and using $\Delta T_{\text{min},i,k}$ and $\Delta T_{\text{max},i,k}$ the interval $t = [34, 55]$ is found. d_i is also estimated based on the largest difference in each window.

Algorithm 1 estimates a cycle time range $[t_{min}, t_{max}]$ and d_i for each q_i probed by the attacker. Each O_ι are sent at time $T_{i,j}$ and is used the compute $\delta_{r_i,j}$ (Lines 6 and 7). All residual delays collected for q_i are then recorded as δ_{r_i} (Line 8). Next, the time elapsed between pairs of consecutive $\delta_{r_i,j}$ is computed to establish the threshold value $T_{\text{diff},i}$ in Line 11. Using $T_{\text{diff},i}$, residual delays are grouped as transmissions that potentially occur within the same cycle k as $W_{i,k}$ in Line 12. On Line 14, the window set of q_i is stored in $\mathcal{W}$, where each $W_{i,k} \in \mathcal{W}$ represents an observed transmission window for $k = 0, \ldots, \mathcal{K}$. The expressions $T_{\text{min},i,k}$ and $T_{\text{max},i,k}$ on Lines 17 represent the earliest and latest observed departure times within a given window $W_{i,k}$ from a queue, respectively. These values are used to estimate the duration of the gate opening d_i, and the range of possible cycle durations $\Delta T_{\text{min},i,k}$ and $\Delta T_{\text{max},i,k}$ (Lines 20 and 21) for each queue. Finally, t_{min} and t_{max} are computed on Lines 22 and 23 respectively.

5.2 Estimation Validation and Refinement

Algorithm 1 returns the given initial estimate of the TAS cycle-time bounds $[t_{\text{min}}, t_{\text{max}}]$ and gate duration d_i for queue q_i. The refinement phase aims to align probing with a future gate opening and to improve these estimates using the most recent delay observations. Therefore, the attacker must verify that previously observed transmissions recur in future cycles. To achieve this, we use a sliding window to transmit N observer frames in successive rounds within the estimated t_{min} and t_{max} values.

Algorithm 1 GCL Estimation.

Input: N observer frames, τ
Init: $t_{min} \leftarrow \infty$, $t_{max} \leftarrow -\infty$, $d_i \leftarrow -\infty$
Init: $\mathcal{W} \leftarrow \{\}$, $\delta_{r_i} \leftarrow []$
for $q_i \in Q$ **do**
  for $j = 0$ to $N-1$ **do**
    Send observer frame at $T_{i,j} \leftarrow T_0 + j \cdot \tau$
    Compute $\delta_{r_i,j}$ from Eq. 1
    Append $\delta_{r_i,j}$ to δ_{r_i}
  end for
  Init: $k \leftarrow 0$, $W_{i,k} \leftarrow \{\delta_{r_i,0}\}$
  Compute $T_{\text{diff},i} \leftarrow \mu_{\{\Delta\delta_{r_i,j}\}} + \sigma_{\{\Delta\delta_{r_i,j}\}}$
  for each $j = 1, \ldots, N-1$ **do**

$$W_{i,k} = \begin{cases} W_{i,k} \cup \{\delta_{r_i,j}\}, & |\Delta\delta_{r_i,j}| \leq T_{\text{diff},i}, \\ \{\delta_{r_i,j}\}, & \text{otherwise, } k = k+1 \end{cases}$$

  end for
  $\mathcal{W} = \{W_{i,k} \mid k = 0, 1, \ldots, \mathcal{K}\}$
end for
for each (i, k, W) in $\mathcal{W}$ **do**
  $d_i \leftarrow \max(d_i, C_{max,i,k} + (T_{\max,i,k} - T_{\min,i,k}))$
end for
for each (q_i, k, W) in $\mathcal{W}$, $k < |\mathcal{W}_{q_i}| - 1$ **do**
  $\Delta T_{\min,i,k} \leftarrow T_{\min,i,k+1} - T_{\min,i,k}$
  $\Delta T_{\max,i,k} \leftarrow T_{\max,i,k+1} - T_{\max,i,k}$
  $t_{min} \leftarrow \min(t_{min}, \min(\Delta T_{\min,i,k}, \Delta T_{\max,i,k}))$
  $t_{max} \leftarrow \max(t_{max}, \max(\Delta T_{\min,i,k}, \Delta T_{\max,i,k}))$
end for
return $[t_{min}, t_{max}]$, d_i

Let $T_{s,i,\mathcal{K}}$ denote the most recently inferred start time of queue q_i's transmission window, obtained by selecting the minimum observed delay in the last window $W_{i,\mathcal{K}}$ and let T_{curr} be the current time. To avoid probing within an already elapsed window, the next estimated gate-opening time is initialized conservatively as $T_{s,i,\mathcal{K}} + t_{\min}$. Since the true cycle time may still deviate from $t_{\min}$ during refinement, the estimation is advanced in steps of $t_{\min}$ until it lies strictly in the future:

$$t_{\text{next}} = T_{s,i,\mathcal{K}} + \eta\, t_{\min}, \quad \eta = \left\lceil \frac{T_{\text{curr}} - T_{s,i,\mathcal{K}}}{t_{\min}} \right\rceil + 1. \tag{7}$$

η counts how many conservative cycle increments of length $t_{\min}$ must be added to the last inferred window start so that probing does not occur within an already elapsed window. Using the current estimate of the gate duration d_i, the refinement interval is constructed as

$$[\widehat{T}_{\min,i,k},\ \widehat{T}_{\max,i,k}] = [t_{\text{next}},\ t_{\text{next}} + d_i]. \tag{8}$$

Observer frames are transmitted within this interval at a fixed probing granularity τ, starting at $\widehat{T}_{\min,i,k}$ until $\widehat{T}_{\max,i,k}$ is reached. The corresponding $\{\delta_{r_i,j}\}$ is recorded for each queue. From the resulting delay observations, a refined estimate of the gate duration,

Table 4 Refined Time Values from Observed Windows.

Queue	$T_{s,i,\mathcal{K}}$	$\widehat{T}_{\min,i,k}$	$\widehat{T}_{\max,i,k}$	$W_{i,k}$	d_i
q_7	50	50+34=84	50+55=105	$\{108, 110\}$	15
q_6	65	65+34=99	65+55=120	$\{115, 128\}$	15
q_2	80	80+34=114	80+55=135	$\{130, 148\}$	148-130+2=20

denoted $d_{i,\text{new}}$, is computed based on the delay windows. To conservatively account for underestimation, the refinement updates d_i when longer durations are observed. The gate duration estimate is updated as $d_i \leftarrow \max(d_i, d_{i,\text{new}})$.

Let the first observed delay in $W_{i,0}$, denoted as $T_{s,i,0}$, be the newly inferred start time of the first window obtained from the refined observations. Because refinement relies on the conservative step size $t_{\min}$ and the cycle-time estimate may not yet have converged, strict alignment with the predicted interval is not required. Instead, the inferred window is considered valid if

$$T_{s,i,0} \in \left[\widehat{T}_{\min,i,k},\ \widehat{T}_{\max,i,k} + \upsilon_i\right], \qquad \upsilon_i \leq d_i, \tag{9}$$

where υ_i bounds the cycle time underestimation and is conservatively upper-bounded by d_i. This check is necessary to avoid including out-of-window values that may skew the gate duration estimate. Valid windows are retained for further refinement. If the condition is not satisfied, the observed window is discarded. In this case, the most likely cause is blocking by previously queued frames, since the underlying TAS schedule is assumed to repeat across cycles. The reference window start is updated as $T_{s,i,\mathcal{K}} \leftarrow \min(W_{i,\mathcal{K}})$. After processing all queues, the cycle-time estimate is updated to reflect the effective serialized gate durations across queues. The refinement process repeats until at least one valid window is obtained, yielding updated estimates of the cycle time and gate durations.

▶ **Example 2.** We illustrate the refinement technique described above using the cycle time obtained in Example 1 (in Section 5.1 and Figure 4). We apply the refinement when the next rounds of frames are sent to the queues in areas marked in blue. The next windows for q_7, q_6, and q_2 are predicted using the minimum values $T_{s,i,\mathcal{K}}$ from the last observed windows. These values are added to $t_{\min}$ and $t_{\max}$, as shown in Table 4. Two observer frames were each sent at times $\widehat{T}_{\min,i,k}$ to $\widehat{T}_{\max,i,k}$. As mentioned earlier, we need to update all d_i if necessary. Here, only d_2 changed to 20. Summing d_i, the total estimated cycle time: $t = 15 + 15 + 20 = 50$.

5.3 Timing Attack For Adversarial Blocking

In this step, the attacker leverages the insights obtained in Sections 5.1 and 5.2 to identify a higher-priority pMAC queue and strategically inject lower-priority malicious frames at critical moments to induce preemption and delay higher-priority frames. eMAC flows exhibit low delay variance [6], whereas pMAC flows show higher variability, with express queues incurring lower delays than preemptable ones.

Algorithm 2 describes the attack execution phase, which leverages the refined timing reference $T_{s,i,\mathcal{K}}$, t, and d_i to (i) identify pMAC queues and (ii) continuously inject malicious frames at specific times. The attacker first iterates over each q_i and initializes the next estimated gate opening time $t_{\text{next}} \leftarrow T_{s,i,\mathcal{K}} + t$. From x up to $R_{\max}$ probing rounds, the attacker forms the predicted open interval (slot) as $[t_{\text{next}},\ t_{\text{next}} + d_i]$ and transmits N observer frames within this slot. From the resulting delays, the attacker constructs windows $W_{i,k}$ and then computes an updated duration estimate $d_{i,\text{prem}}$ that captures the preemption-induced increase in the observed window. If the updated estimate satisfies $d_{i,\text{prem}} > d_i + C_\iota + \delta_{\text{prem}}$,

Algorithm 2 Attack Execution.

Input: Max rounds $R_{\max}$, x, $T_{s,i,\mathcal{K}}, t, d_i$, δ_{prem}, ϵ
$\mathcal{P} \leftarrow \emptyset$; $\mathcal{E} \leftarrow \emptyset$, $x \leftarrow 1$
for each queue q_i **do**
  $t_{next} \leftarrow T_{s,i,\mathcal{K}} + t$
  for $x = 1$ to $R_{\max}$ **do**
    Probe q_i with observer frames in $[t_{\text{next}}, t_{\text{next}} + d_i)$
    Estimate new duration as $d_{i,\text{prem}}$
    if $d_{i,\text{prem}} > d_i + C_\iota + \delta_{\text{prem}}$ **then**
      $\mathcal{P} \leftarrow \mathcal{P} \cup \{q_i\}$ ▷ preemptable
      break
    end if
    $t_{\text{next}} \leftarrow t_{\text{next}} + t$
  end for
  if $q_i \notin \mathcal{P}$ **then**
    $\mathcal{E} \leftarrow \mathcal{E} \cup \{q_i\}$ ▷ express
  end if
end for
$q_a \leftarrow \min(\mathcal{P}), q_v \leftarrow \max(\mathcal{P})$
$T_{\text{inj}} \leftarrow (t_{\text{next}} + d_a) - \epsilon$
while attack is active **do**
  Send malicious frame to q_a at time T_{inj}
  $T_{\text{inj}} \leftarrow T_{\text{inj}} + t$
end while

the queue is classified as preemptable and inserted into the set $\mathcal{P}$; otherwise, the attacker advances to the next cycle by updating $t_{\text{next}} \leftarrow t_{\text{next}} + t$ and continues probing. Queues that do not satisfy the preemption condition are classified as eMAC and added to $\mathcal{E}$.

After classification, the attacker selects the lowest-priority pMAC queue q_a as the attacker and the highest-priority pMAC queue q_v as the victim. The next injection interval is aligned to the first future cycle, and a malicious frame is transmitted near gate closure at $T_{\text{inj}} = t_{\text{next}} + d_a - \epsilon$. $\epsilon > 0$ is a small offset ensuring transmission begins just before gate closure. Finally, to realize a persistent SBA, the adversary repeatedly transmits malicious frames to q_a at T_{inj} and updates $T_{\text{inj}} \leftarrow T_{\text{inj}} + t$ to repeat the injection once every cycle, thereby sustaining delays to the higher-priority victim traffic.

▶ **Example 3.** Consider if $O_{3,6}$ in Figure 4 is at least 123 bytes, making it preemptable. At time 150, it will be preempted by $f_{1,7}$ and resume around time 165. Then, with Algorithm 1, $W_{2,2} = \{130, 165\}$, so the extra delay is due to preemption rather than a full cycle, as it waits only for d_7 plus preemption overhead. By contrast, an eMAC frame like $O_{1,6}$ would complete before 120 μs without preemption as the added duration will be its length of C. Hence, q_2 and q_6 are pMAC, while q_7 is eMAC. The attacker chose the lowest-priority pMAC as its queue and the higher-priority pMAC as the victim's queue. Using $W_{2,2} = \{130, 148\}$ as offset from Table 4, the attacker sends the malicious frames to q_2, $t = 50$ and $d_2 = 20$, the next intervals are computed as: $130 + 50 = 180$, up to $180 + 20 = 200$. Therefore, the attacker targets the intervals $[180, 200]$, $[230, 250]$ and so on, in successive cycles. Since at least two observer frames are transmitted in each cycle, the attacker can send a malicious frame (shown in red box) that will fill at least the transmission of the observer frames, e.g, sending around $5\mu s$ before gate closure 195, 245, and so on, delays the transmission of $f_{v,9}$.

Algorithm 2 summarizes the attack execution, where the attacker identifies preemptable queues by tracking changes in observed gate durations and computes the next transmission slot from $T_{s,i,\mathcal{K}}$ using the estimated cycle time t (Line 4). The slot spans the estimated gate duration d_i and is used to transmit N observer frames that may be preempted (Line 6). The resulting end-to-end delays are grouped into window segments $W_{i,k}$ to isolate delays caused by preemption events. From each window, a refined duration estimate $d_{i,\text{prem}}$ is computed (Line 7). If this estimate exceeds the previous duration by more than the transmission time and preemption overhead (Line 8), the queue is classified as preemptable and added to the set $\mathcal{P}$ (Line 9), and probing for that queue terminates. Queues that do not satisfy this condition within $R_{\max}$ rounds are classified as express and added to $\mathcal{E}$ (Line 15).

After classification, the attacker selects the lowest-priority preemptable queue as the injection queue q_a and the highest-priority preemptable queue as the victim queue q_v (Line 18). The next injection slot is aligned to a valid cycle, and a malicious frame is transmitted near gate closure at T_{inj} (Line 21). Once aligned, the attacker repeats this injection in each subsequent cycle to sustain interference and delay the victim traffic.

To quantify the impact of the SBA, let E be the attacker's pMAC frame spillover into the victim gate,

$$E = \max\Big(0,\ \min(T_{s,v,k} + d_v,\ T_{s,a,k} + C_a + \delta_{\text{prem}}) - \max(T_{s,v,k},\ T_{s,a,k})\Big) \tag{10}$$

i.e., the time within the victim window consumed by the attacker's transmission. Here, C_a is the transmission time of the fragmented frame that resumes into the victim's open slot. The remaining time for the victim queue in that cycle is therefore $d_v - E$. To account for multiple queued victim frames, let B_v denote the total victim queue workload present at the opening of the victim gate. A slot miss occurs when $B_v > d_v - E$, which defers the victim workload to later cycles. A deadline miss for the victim frame under analysis occurs when the resulting extra waiting, approximately $\left(\left\lceil \frac{B_v}{d_v - E} \right\rceil - 1\right) t$, causes C_v to exceed D_v. A single-cycle deferral may be sufficient to cause a deadline miss, whereas flows with larger deadlines can tolerate repeated blocking.

6 Evaluation

We conducted experiments to measure the accuracy of the GCL estimation (Section 6.1), the impact of SBA on TSN traffic (Section 6.2), and IDS performance at detecting the SBA (Section 6.3). For design space exploration and evaluation of GCL estimation, we created a Python simulator that generates synthetic TSN traffic emulating realistic scenarios. The simulator implements the TSN TAS queues, each with an associated GCL. We also assembled a realistic hardware-based TSN environment in a lab, as shown in Figure 5. It comprises two TSN-capable NXP switches (LS1028A-1 and LS1028A-2) and three end devices acting as traffic generators. Two BeagleBone Black (BBB) devices and one LS1021ATSN generate TSN traffic to LS1028A-1, which serves as the central switch and connects downflow to LS1028A-2. The LS1028A switches provide four Ethernet ports with up to 1-Gbps link speed, hardware timestamping, and run OpenIL for deterministic TSN operation. All ports are bridged and synchronized via PTP to a common time base. The BBBs and LS1021ATSN synchronize to the PTP master and generate flows from different TSN traffic classes, enabling precise evaluation of SBA-induced blocking effects.

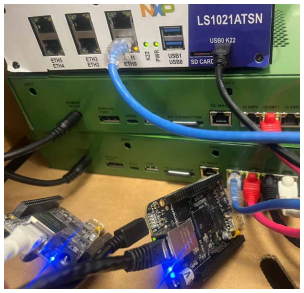


Figure 5 Hardware Testbed Setup.

6.1 GCL Estimation Experiments

To evaluate the accuracy of the GCL estimation we conducted experiments using both the software simulator and hardware testbed.

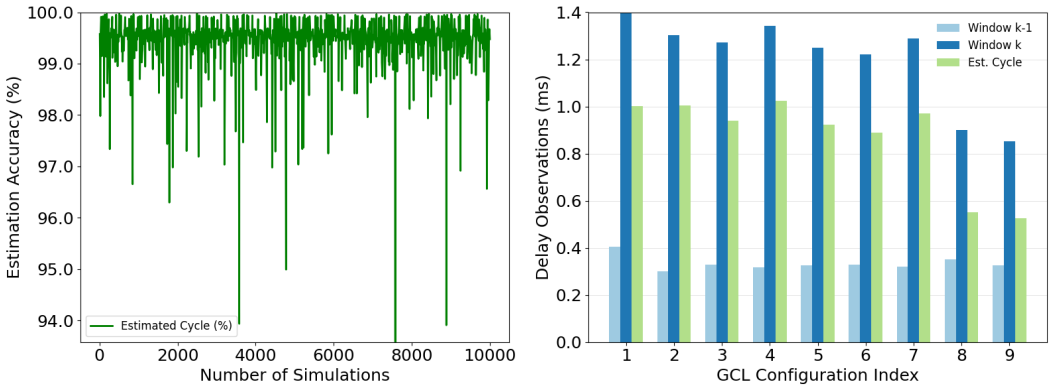
6.1.1 GCL Schedule Estimation in Simulation

Using the software simulator we generated 10,000 simulation runs lasting approximately 5 minutes of simulated time. In each run, the number of queues was randomized between 3–8, with a fixed link speed of 100 Mbps. The cycle time was randomly chosen from [3,000,15,000], and gate durations were assigned via a weighted random distribution. Two types of traffic were modeled per queue: observer frames below 123 bytes were generated periodically at $300\mu s$ intervals to probe the gate-opening patterns. Additional background frames with randomized sizes from 123-1500 bytes were also generated to simulate normal network load, to be between $50 - 85\%$ load ratio compared to observer frames. These background frames shared the same transmission queues. To model preemption in the simulation, the number of express queues is set to at least 33% of the total number of queues, rounded up to the nearest whole number. When an express queue is transmitting, the gates of other queues remain closed. However, preemptable frames may still be partially transmitted if their gates were previously open.

Using Algorithm 1 and the refinement analysis, we analyzed the observer frames delays to obtain the transmission windows, the cycle time range and gate durations. During refinement, observer frames were aligned with the estimated cycle time to improve gate duration accuracy. For each simulation run, the estimated cycle time was compared against the true cycle time. Estimation accuracy was computed as $\left(1 - \frac{|\text{Estimated} - \text{Actual}|}{\text{Actual}}\right)$. Figure 6a shows the actual cycle time in blue (i.e., 100% accuracy) and the estimated cycle time in green. Our approach achieves over 99% accuracy in most cases, with the lowest observed accuracy of about 94%. The average accuracy is 99.74%.

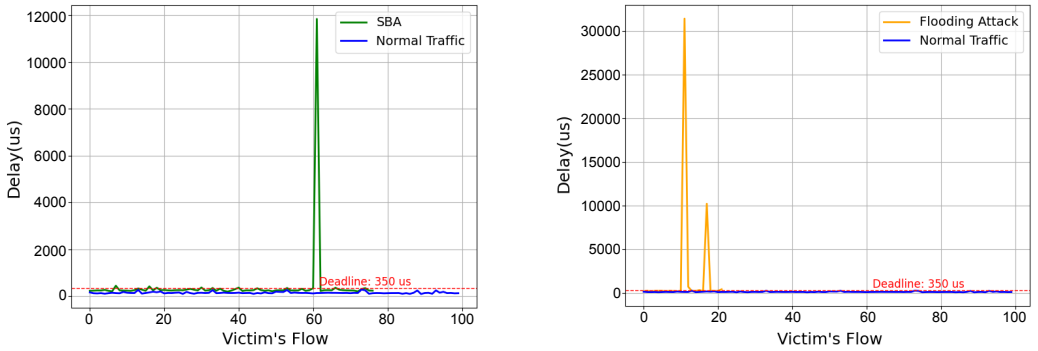
6.1.2 GCL Schedule Estimation on Hardware

To evaluate gate observability and window segmentation, we generated periodic ICMP observer traffic (assigned to priority 2) from an LS1021ATSN host to an LS1028-2 receiver through a TAS-configured switch. All egress queues were defined in the GCL, but only the ICMP queue was allowed to transmit during its gate window. The TAS cycle time was fixed at 1 ms, and the ICMP gate allocation was swept across nine configurations ranging



(a) Simulation. **(b)** Hardware.

Figure 6 (a) Estimation accuracy of cycle times across multiple simulations as a percentage. (b) The first two bars show the earliest observed delays after the gate opens in window $k-1$ and k, while the third bar shows the estimated cycle time, the actual cycle time is 1 ms.



(a) Delay under the SBA compared to normal traffic.

(b) Delay under the flooding attack compared to normal traffic.

Figure 7 Hardware comparison of SBA and flooding attacks under normal traffic.

from $[900\,\mu s, 100\,\mu s]$ to $[100\,\mu s, 900\,\mu s]$ open/closed durations. ICMP packets were sent every $50\,\mu s$, and end-to-end delays were extracted from PCAP traces. Algorithm 1 was applied to segment windows $W_{q_i,k}$ and infer window start times. Figure 6b shows the inferred window boundaries and estimated cycle times. When the open window is short (Configs 1–7), window boundaries are stable, with $T_{s,q_i,k-1} \in [0.30, 0.40]$ ms and $T_{s,q_i,k} \in [1.20, 1.40]$ ms, clearly revealing the 1 ms cycle. As the open window increases (Configs 8–9, $\geq 80\%$ open), temporal contrast decreases, leading to less stable segmentation and a reduced cycle estimate (0.50–0.55 ms). This confirms that segmentation is most reliable when closed intervals provide strong temporal separation or when multiple queues are observed.

6.2 SBA Impact

We examined two case studies using TSN configurations inspired by prior art [18, 25] to characterize the impact of SBA.

Table 5 TSN Traffic Classes for Scenario 1.

Flow	Source	ρ_i	Period (µs)	d_i (µs)	Size (Bytes)
eMAC	Controller	7	250	50	64
eMAC	IOD1	6	300	50	128
pMAC	IOD1	5	350	50	1472
pMAC	IOD2 (Compromised)	2-0	500–1000	250	1472

6.2.1 Case Study 1

Table 5 summarizes the TSN flow configuration used in this case study, which is inspired by [18]. The objective is to evaluate the impact of a compromised I/O device that intentionally aligns its transmissions with TAS gate boundaries to induce adversarial blocking. Two BBB devices serve as the controller and I/O Device 1 (IOD1), respectively. An LS1021A device acts as I/O Device 2 (IOD2) and is assumed to be compromised. All traffic is forwarded through a TSN switch (LS1028-1) toward a receiving switch (LS1028-2), which supports frame preemption. The controller and IOD1 generate express traffic mapped to queues 7 and 6, respectively. In addition, IOD1 generates an independent AVB flow mapped to queue 5, which serves as the victim flow f_v, while IOD2 generates BE traffic mapped to queue 2-0; both flows are configured as preemptable traffic.

TAS is enabled on interface `swp1` of LS1028-1 with a three-entry GCL and a cycle time of 400 μs. Queues 7, 6, and 5 are each allocated 50 μs, while queues 0–2 share a 250 μs window. All bridge ports are synchronized via PTP. Using crafted ICMP observer frames, we inferred gate timing and measured end-to-end delay. The measured propagation delay was approximately 35 ns. Gate closures of up to 148 μs were observed during queue transitions, and RTTs converged to approximately 252 μs, confirming the expected 400 μs cycle. Frames larger than twice the `MIN-FRAG-SIZE` (60 bytes) exhibited fragmentation effects, verifying that queues 2 and 5 were preemptable. For the SBA, the compromised node aligns its transmissions with the end of its assigned gate window ($\rho_{\iota,j} = 2$). Release times are computed using the inferred window boundaries and cycle time as defined in Algorithm 2. Using the GCL base time T_b, the current TAS cycle index is computed as $k = \left\lfloor \frac{T_{\text{curr}} - T_b}{t} \right\rfloor$. IOD2 transmits with a period approximately $800\,\mu s$ with an offset $\epsilon = 5\,\mu s$, aligning near gate boundaries to increase preemption events.

PCAP traces were collected over a five-minute interval. Under normal operation, the average delay of f_v remained near 150 μs with no deadline violations (Figure 7a). Under SBA, approximately 448,800 preemption events were observed, reflected by a sharp increase in the `MACMergeFragCountTx` counter. Under our configuration on the switch, the resumption of preempted pMAC frames results in spillover. Analysis of 100,000 victim frames (averaged over bursts of 100) showed that roughly 10% of the first 60 bursts violated deadlines, with delays peaking at 365 μs. This behavior indicates that, in some cycles, f_a was itself preempted. However, when sufficient transmission time remained within the victim's gate window, these events did not consistently result in deadline violations for f_v. As preemption accumulated, worst-case delays increased sharply, reaching 11.8 ms. Beyond the 78th burst, persistent deadline misses caused frame drops, with delivery falling below 80%. For comparison, we evaluated a flooding attack without GCL awareness, transmitting frames every 105 μs (approximately $9,500$ packets/s). Victim frames initially met deadlines, but delays increased rapidly due to excessive queuing, reaching 31.3 ms by the 22nd burst, after which sustained frame loss was observed (Figure 7b).

Table 6 Rate-Based IDS Detection Results.

Attack	Pkt Rate	Byte Rate	IDS Thresholds	IDS Alert	(%) of Drops
Flooding	$>$ 9000 pps	$\sim$ 108 Mb/s	8000 pps & 20 Mb/s	Yes (R1)	22
SBA	$<$ 1300 pps	$\sim$ 15 Mb/s		No	78

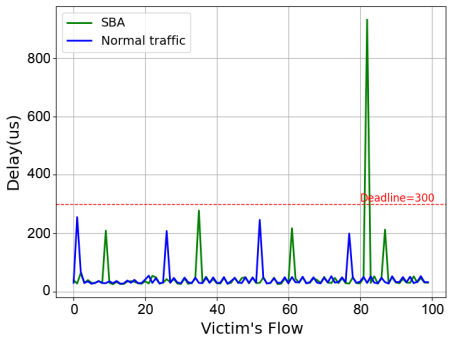
Table 7 Flow characteristics with slot time per queue.

Configuration	Flow	Type	ρ_i	Period (ms)	Size (Bytes)	d_i (μs)
(a)	eMAC	Generic	6	0.2	512	15
	pMAC	Generic	5	0.3	840	15
	pMAC	Generic	4	0.5	1472	15
(b)	eMAC	Tactile	6	1	82	200
	pMAC	Audio VBR	5	24	480	200
	pMAC	Video VBR	4	16.67	1472	200

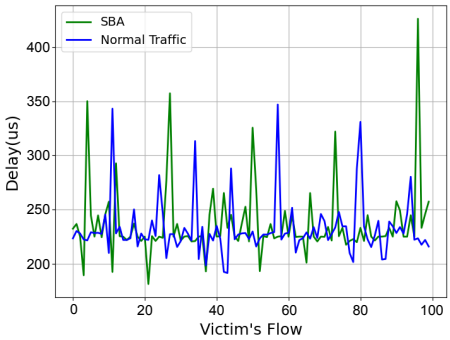
6.2.2 Case Study 2

All configurations employ a fixed gate duration equal to one-third of the cycle across all queues. Configuration (a) represents a generic traffic scenario in which all flows generate the same traffic type. The flow with $\rho_{\iota,j} = 6$ is mapped to an eMAC queue, while the remaining flows are assigned to pMAC queues. The attacker generates pMAC traffic with $\rho_{\iota,j} = 4$. All traffic classes are configured with short $d_i = 15$ μs, with a total cycle time of 45 μs [25]. Configuration (b) represents a tactile control scenario in which the flow with $\rho_{\iota,j} = 6$ is mapped to an eMAC queue, while the remaining flows are AVB traffic mapped to pMAC queues. Larger $d_i = 200$ μs with longer periods are used to emulate multimedia traffic.

Under configuration (a), uniformly short gate durations result in frequent preemption events affecting both victim and attacker frames. Although the attacker's frames are preempted, the victim's fragment also repeatedly defers the attacker's transmissions. This repeated preemption disrupts the attacker's alignment with the victim gate, causing attacker fragments to be shifted outside the victim transmission window. Consequently, worst-case blocking does not occur immediately. In our measurements, a pronounced delay event occurs around the 80th burst, where the victim experiences delays of up to 800 μs due to accumulated cycle misses, as shown in Figure 8a. This result shows that once alignment

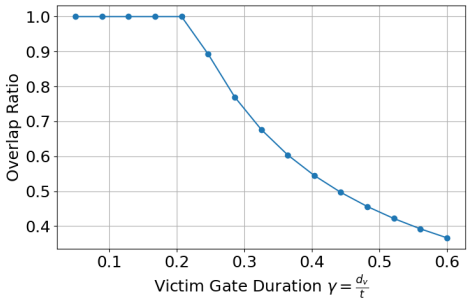


(a) Case Study 2a: Shows the effect when short gate duration and frequent interruptions affect the attackers' spillover.

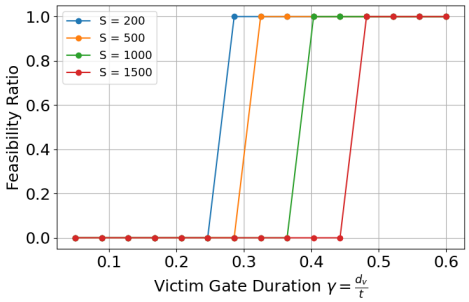


(b) Case Study 2b: Shows the effect when the victim's long gate duration absorbs the attacker's spillover.

Figure 8 Comparison of SBAs for different delay values compared with normal traffic.



(a) Attacker spillover overlap versus victim gate duration (as a fraction of a fixed t).



(b) Slot completion feasibility versus victim gate duration fraction under attacker spillover.

Figure 9 Impact of attacker spillover on victim gate utilization and slot completion feasibility.

occurs, feasibility is governed by the resulting spillover overlap E, which captures how much attacker transmission extends into the victim window rather than by the period. However, the period sets the available deadline margin.

In configuration (b), the victim flow has a period and deadline = 24ms. While the d_i of the victim remains fixed at one third of the cycle, increasing d_v enlarges the victim's transmission opportunities $d_v - E$. Consequently, the additional waiting induced by the attack, $\left(\left\lceil \frac{B_v}{d_v - E} \right\rceil - 1\right) t$, remains within the available deadline margin. Figure 8b shows the worst-case scenario for the attacker: long gate durations absorb spillover effects, allowing the victim (Audio VBR (pMAC)) flow with D_v of 24 ms to transmit within its slot despite the SBA, as the maximum delay (450μs) is not enough to cause deadline miss. Together, configurations (a) and (b) show that SBA effectiveness is governed by the interaction between the spillover overlap E and the effective service window $d_v - E$ relative to the period or deadline. While varying the transmission period influences alignment and delay tolerance, it does not alter spillover formation itself once it occurs. The same overlap may therefore cause deadline violations under short deadlines, whereas under longer deadlines it can be absorbed. This motivates our subsequent evaluation of the feasibility of victim slot completion in a cycle across varying gate durations and frame sizes, while ensuring that both the gates and the periods align, allowing attacker traffic to overlap with the victim flow.

6.2.3 Spillover Impact on SBA Effectiveness

To analyze the overall effectiveness of the SBA under alignment, we next generalize beyond the single traces by analyzing the structural vulnerability of TAS schedules to SBA-induced spillover through a simulated dataset. We consider $t = 500\mu s$ and parameterize the victim gate duration as a fraction of the t such that $\gamma = \frac{d_v}{t}$. The observed behavior is driven by the absolute duration of the victim gate due to the fixed t. The attacker frame size is fixed to 1500, and the period is fixed to one injection each cycle on average to align with the victim's gate. We vary the victim frame size $\in \{200, 500, 1000, 1500\}$ to control the extent to which its flow utilizes its allocated gate duration d_v. In Figure 9a, the y-axis also shows the normalized overlap ratio $\frac{E}{d_v}$, i.e., the fraction of the victim d_v consumed by attacker transmission and preemption overhead. A value of $\frac{E}{d_v} = 1$ indicates complete overlap of the victim gate ($d_v - E = 0$), whereas $\frac{E}{d_v} = 0$ denotes no overlap. For small γ (up to approximately 0.2), the overlap reaches $\frac{E}{d_v} = 1$, fully occupying the victim gate slot. As γ increases, the victim gate duration grows faster than the attacker spillover, resulting in a monotonic decrease in

the normalized overlap $\frac{E}{d_v}$. The overlap drops below one (e.g., $\frac{E}{d_v} = 0.9$ at $\gamma = 0.25$) and falls below one-half at $\gamma = 0.60$. These results show that when the victim flows have more transmission opportunities, attacker spillover can be absorbed.

We next show how attacker overlap affects slot completion feasibility within a cycle, as illustrated in Figure 9b. The y-axis indicates in-cycle feasibility: 1 indicates that the victim frame completes entirely within its slot, and 0 denotes insufficient remaining time under SBA. Since only $d_v - E$ remains usable, feasibility requires $d_v - E \geq C_v$ (Eq. 10). As γ increases, a sharp transition to feasibility occurs once the remaining open time exceeds the victim transmission demand. For example, feasibility is restored at $\gamma = 0.28$ ($d_v = 140$ μs) for a 200 bytes frame, whereas a 1500 bytes requires nearly half of the cycle to transmit fully.

6.3 IDS Evaluation

Existing IDS systems, such as [9], are designed to detect anomalies like excessive resource requests, duplicate flows, and flooding requests. However, these rules are ineffective against the SBA presented in this paper. SBA enables an attacker to subtly manipulate transmission windows to increase the frequency of preemption, a behavior that is undetected by traditional IDS systems. Although flooding attacks cause immediate impact, they are easier to detect.

We implemented an IDS using Zeek to analyze incoming packets on the switch in Case Study 1 (Section 6.2) using three main rules: R1: rate of incoming frames (any source/protocol), R2: duplicate flow, and R3: per-host request monitoring. We included a sliding threshold-based detector to identify excessive requests and high-rate traffic. The system maintains a smoothed baseline packet rate per host and raises an alert when the current rate exceeds a dynamic threshold (e.g., 2× the baseline of 4000 packets/s), thereby minimizing false positives. In the IDS context, a packet denotes an Ethernet frame being monitored. The thresholds are set to 8000 packets/s (pps) and 20 Mb/s. Additionally, duplicate flows sent at short intervals are flagged, capturing transmissions that deviate from normal host behavior. The rule ensures that even high probing requests can be detected. The SBA remains undetected because these rules are rate and volume-based, while the attack exploits precise timing at low traffic rates. On the other hand, the system detected more than 9,000 pps and a low-priority frame byte rate of 108 Mb/s during the flooding attack and flagged them. Table 6 summarizes the IDS performance on attack types.

7 Related Work

TSN Attacks. Topsakal and Cevher [24] analyze the impact of denial-of-service (DoS) attacks on TSN, focusing on how delays caused by DoS attacks across different traffic classes affect network performance. Similarly, Bülbül et al. [6] conducted a traffic analysis to predict and exploit specific traffic classes to initiate preemptive DoS attacks. Security vulnerabilities related to the Ethertype field in gPTP packets are highlighted by Fotouhi et al. [11], where improper checks by switches allow adversaries to bypass gPTP security measures and conduct high-risk spoofing attacks. Fischer and Merli [10] discuss the broader security implications for TSN within industrial control systems, including potential DoS, Man-in-the-Middle, and other PTP-based attacks. In contrast, none of these works examined the potential for an SBA attack in TSN.

Schedule-Based Attacks. SBAs pose a significant threat, particularly in real-time systems, where attackers exploit task timing to extract sensitive information [14]. The work by Nasri et al. [17] introduces a taxonomy for SBA, categorizing them based on the timing relationships

critical to their success. Much of the prior work has focused on SBA and defense within real-time processor scheduling [2, 3, 7, 15, 16, 26]. Notably, Hounsinou et al. [12] discuss the vulnerability of the automotive Controller Area Network (CAN) bus to SBA, emphasizing the need for robust scheduling defenses in vehicular systems. The authors show how to use SBA techniques to conduct classical CAN injection and bus-off attacks. Our work instead targets SBAs in TSN, where deterministic frame transmission is enforced by IEEE 802.1Qbv GCLs in network switches, exposing a distinct timing attack surface with a unique impact on the timing predictability of the victim frame.

8 Mitigations

The SBA analyzed in this work is configuration-dependent and can be mitigated using standard TSN mechanisms, although trade-offs remain. A direct mitigation is to assign TAS traffic to eMAC. Since eMAC frames are not subject to preemption, the fragmentation and resumption behavior that enables spillover does not arise. However, this limits the efficiency gains of frame preemption for mixed-criticality workloads.

PSFP, along with admission control and stream policing, can restrict unauthorized or non-conformant traffic. These mechanisms are more effective against injection than probing: policing can detect or drop non-conformant traffic, while admission control limits which flows are permitted. However, probing traffic from a compromised but legitimate end node may remain undetected. These approaches also introduce increased configuration complexity and reduced runtime flexibility.

Preemption-with-hold introduces guard bands that prevent new pMAC transmissions near gate boundaries, reducing spillover likelihood. However, guard bands do not constrain the resumption of preempted fragments; thus, spillover may still occur, while the attack surface is reduced. Another trade-off is reduced link utilization, as guard band intervals remain unused regardless of whether express traffic is present. Collectively, these mitigations highlight the role of careful system configuration in ensuring secure TSN deployments.

9 Conclusion

SBAs exploit fine-grained timing manipulations in a way that is hard to detect with a traditional IDS that focuses on traffic anomalies. This work highlights an SBA that aligns malicious flows to cause targeted adversarial blocking of a victim frame. Unlike flooding attacks, SBAs use subtle techniques that evade rate-based IDS detection. Evaluating the effectiveness of the mitigation approaches against SBA is left to future work. Future work could also explore IDS designs capable of monitoring internal TSN switch schedules or access control mechanisms at the switch level to ensure only authorized nodes transmit specific traffic classes.

References

1 Mohammad Ashjaei, Mikael Sjödin, and Saad Mubeen. A novel frame preemption model in tsn networks. *Journal of Systems Architecture*, 116:102037, 2021. doi:10.1016/J.SYSARC.2021.102037.

2 Vijay Banerjee, Sena Hounsinou, Yanyan Zhuang, Monowar Hasan, and Gedare Bloom. On evading randomization-based defense in hierarchical real-time systems. *ACM Trans. Cyber-Phys. Syst.*, 10(2), 2026. doi:10.1145/3783983.

3 Sanjoy Baruah, Pontus Ekberg, Mehdi Hosseinzadeh, Ao Li, Bryan Ward, and Ning Zhang. Who's Afraid of Butterflies? A Close Examination of the Butterfly Attack. In *2023 IEEE Real-Time Systems Symposium (RTSS)*, pages 53–63, 2023. ISSN: 2576-3172. doi:10.1109/RTSS59052.2023.00015.

4 Lucia Lo Bello, Mohammad Ashjaei, Gaetano Patti, and Moris Behnam. Schedulability analysis of time-sensitive networks with scheduled traffic and preemption support. *Journal of Parallel and Distributed Computing*, 144:153–171, 2020. doi:10.1016/J.JPDC.2020.06.001.

5 Daniel Bujosa, Julian Proenza, Alessandro V Papadopoulos, Thomas Nolte, and Mohammad Ashjaei. An improved worst-case response time analysis for avb traffic in time-sensitive networks. In *2024 IEEE Real-Time Systems Symposium (RTSS)*, pages 135–147. IEEE, 2024. doi:10.1109/RTSS62706.2024.00021.

6 Nurefşan Sertbaş Bülbül and Mathias Fischer. Preemptive DoS attacks on Time Sensitive Networks. In *GLOBECOM 2023-2023 IEEE Global Communications Conference*, pages 4289–4294. IEEE, 2023. doi:10.1109/GLOBECOM54140.2023.10437837.

7 Chien-Ying Chen, Amiremad Ghassami, Stefan Nagy, Man-Ki Yoon, Sibin Mohan, Negar Kiyavash, Rakesh B Bobba, and Rodolfo Pellizzoni. Schedule-based side-channel attack in fixed-priority real-time systems. Technical report, University of Illinois Urbana-Champaign, 2015.

8 Doğanalp Ergenç, Cornelia Brülhart, Jens Neumann, Leo Krüger, and Mathias Fischer. On the security of ieee 802.1 time-sensitive networking. In *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6. IEEE, 2021. doi:10.1109/ICCWORKSHOPS50388.2021.9473542.

9 Doğanalp Ergenç, Robin Schenderlein, and Mathias Fischer. Tsnzeek: An open-source intrusion detection system for ieee 802.1 time-sensitive networking. In *2023 IFIP Networking Conference (IFIP Networking)*, pages 1–6. IEEE, 2023. doi:10.23919/IFIPNETWORKING57963.2023.10186421.

10 Florian Fischer and Dominik Merli. Security considerations for ieee 802.1 time-sensitive networking in converged industrial networks. In *2022 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*, pages 1–7. IEEE, 2022. URL: https://ieeexplore.ieee.org/abstract/document/9988000/.

11 Mahdi Fotouhi, Alessio Buscemi, Abdelwahab Boualouache, Florian Jomrich, Christian Koebel, and Thomas Engel. Assessing the impact of attacks on an automotive ethernet time synchronization testbed. In *2023 IEEE Vehicular Networking Conference (VNC)*, pages 223–230. IEEE, 2023. doi:10.1109/VNC57357.2023.10136275.

12 Sena Hounsinou, Mark Stidd, Uchenna Ezeobi, Habeeb Olufowobi, Mitra Nasri, and Gedare Bloom. Vulnerability of controller area network to schedule-based attacks. In *2021 IEEE Real-Time Systems Symposium (RTSS)*, pages 495–507, 2021. doi:10.1109/RTSS52674.2021.00051.

13 Omolade Ikumapayi, Paul Agbaje, Yanyan Zhuang, Habeeb Olufowobi, and Gedare Bloom. Deadline-based class assignment for time-sensitive network frame preemption. In *Proceedings of the IEEE International Conference on Industrial Technology (ICIT)*. IEEE, 2024.

14 Omolade Ikumapayi, Vijay Banerjee, Sena Hounsinou, and Gedare Bloom. Work-in-progress: Vulnerability of tsn tas with frame preemption to schedule-based attack. In *2025 IEEE Real-Time Systems Symposium (RTSS)*, pages 632–635. IEEE, 2025. doi:10.1109/RTSS66672.2025.00065.

15 Sina Yari Karin, Hakan Aydin, Dakai Zhu, Steven Drager, and Matthew Anderson. Impact of priority assignment on schedule-based attacks in real-time embedded systems. *Journal of Systems Architecture*, 145:103021, 2023. doi:10.1016/J.SYSARC.2023.103021.

16 Ao Li, Marion Sudvarg, Han Liu, Zhiyuan Yu, Chris Gill, and Ning Zhang. PolyRhythm: Adaptive Tuning of a Multi-Channel Attack Template for Timing Interference. In *2022 IEEE Real-Time Systems Symposium (RTSS)*, pages 225–239, 2022. ISSN: 2576-3172. doi:10.1109/RTSS55097.2022.00028.

17 Mitra Nasri, Thidapat Chantem, Gedare Bloom, and Ryan M. Gerdes. On the pitfalls and vulnerabilities of schedule randomization against schedule-based attacks. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 103–116, 2019. doi:10.1109/RTAS.2019.00017.

18 NXP Semiconductors. *Real-Time Edge User Guide*, 2024. URL: https://www.nxp.com/docs/en/user-guide/REALTIMEEDGEUG.pdf.

19 R. Ramaswamy, Ning Weng, and T. Wolf. Characterizing network processing delay. In *IEEE Global Telecommunications Conference, 2004. GLOBECOM '04.*, volume 3, pages 1629–1634 Vol.3, 2004. doi:10.1109/GLOCOM.2004.1378257.

20 Deep Shrestha, Zhibo Pang, and Dacfey Dzung. Precise clock synchronization in high performance wireless communication for time sensitive networking. *IEEE Access*, PP:1–1, February 2018. doi:10.1109/ACCESS.2018.2805378.

21 Soc-e. Time-sensitive networking (tsn) in the electric sector: Security, 2018. Accessed: 2024-02-12. URL: https://soc-e.com/wp-content/uploads/2018/04/security_SAS_TSN-180405.pdf.

22 Daniel Thiele and Rolf Ernst. Formal worst-case performance analysis of time-sensitive ethernet with frame preemption. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–9. IEEE, 2016. doi:10.1109/ETFA.2016.7733740.

23 Daniel Thiele, Rolf Ernst, and Jonas Diemer. Formal worst-case timing analysis of ethernet tsn's time-aware and peristaltic shapers. In *2015 IEEE Vehicular Networking Conference (VNC)*, pages 251–258. IEEE, 2015. doi:10.1109/VNC.2015.7385584.

24 Mustafa Topsakal and Selcuk Cevher. Impact analysis of denial of service attacks in ieee 802.1 time sensitive networking. In *2022 30th Signal Processing and Communications Applications Conference (SIU)*, pages 1–4. IEEE, 2022. doi:10.1109/SIU55565.2022.9864840.

25 Marian Ulbricht, Stefan Senk, Hosein K Nazari, How-Hang Liu, Martin Reisslein, Giang T Nguyen, and Frank HP Fitzek. Tsn-flextest: Flexible tsn measurement testbed. *IEEE Transactions on Network and Service Management*, 21(2):1387–1402, 2023. doi:10.1109/TNSM.2023.3327108.

26 Man-Ki Yoon, Sibin Mohan, Chien-Ying Chen, and Lui Sha. Taskshuffler: A schedule randomization protocol for obfuscation against timing inference attacks in real-time systems. In *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 1–12. IEEE, 2016.