Detecting QoS Degradation in Time-Critical Networks due to Misconfiguration or Attacks

Lukas Bechtel*†, Lukas Popperl*, Michael Menth§, Tobias Heer*†
*University of Applied Sciences Esslingen, Germany, {lukas.bechtel,lukas.popperl,tobias.heer}@hs-esslingen.de
§Chair of Communication Networks University of Tuebingen, Germany, menth@uni-tuebingen.de
†Belden Inc., Neckartenzlingen, Germany

***Abstract*—Communication in Industrial Control Systems (ICSs) depends on predictable timing to ensure reliable operation. In converged networks, this timing can be disrupted not only by cyber attacks but also by misconfiguration or benign misbehavior of devices. Such issues degrade Quality of Service (QoS) through delays or jitter, without altering packet content, making it hard to detect them with traditional monitoring systems.**

This paper presents a configuration-agnostic monitoring system that detects QoS degradation using statistical anomaly detection. We evaluate three detection methods, single-value thresholds, exponential moving averages, and distribution-based analysis, and show that distribution-based detection offers the most robust results. The system supports both passive and active timing measurement to balance accuracy and overhead. By operating independently of network configuration, the proposed approach enables early detection of timing anomalies caused by misbehavior, misconfiguration, or attacks, demonstrating applicability in real-world industrial networks.

***Index Terms*—Industry 4.0, TSN, Time-Sensitive Networking, Security, Network Security, Intrusion Detection System**

I. Introduction

Industrial Control Systems (ICSs) are crucial for automating complex manufacturing processes and ensuring operational efficiency in industrial settings. These systems require control messages to arrive at fixed intervals with minimal deviation, i.e., small jitter. The emergence of Time-Sensitive Networking (TSN) standards has enabled the convergence of industrial automation and IT networks, maintaining Quality of Service (QoS) for ICSs while sharing links with bandwidth-consuming IT applications. However, this convergence increases the complexity of network architecture, raising the risk of misconfiguration, unintentional misbehavior, and cyberattacks.

Misconfiguration of TSN networks or misbehaving end devices can degrade QoS for critical traffic, leading to reduced bandwidth and delayed or lost packets. Similarly, attackers can degrade the timing of traffic rather than altering packet contents [1], for example, by delaying packets or manipulating the time synchronization within the network. In addition, injected traffic can displace time-critical messages. While protection mechanisms exist, such as filtering and policing [2], they are often unavailable because the deployed hardware does not support them at scale [3], [4].

This work has been funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID 528745080 - FIP 68 and "FHP: Qualifizierung und Entwicklung des professoralen Personals der Hochschule Esslingen für zukunftsweisende Themen" (FKZ: 03FHP115) as part of the "FH-Personal", funded by the BMBF and MWK Baden-Württemberg. The authors alone are responsible for the content of the paper.

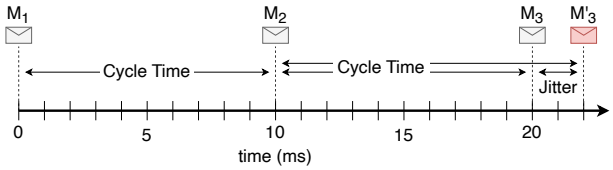


Fig. 1: Exemplary cycle-time and jitter anomaly.

Traditional Intrusion Detection Systems (IDS) rely on packet content inspection and fail to detect QoS degradation, as the degradation does not affect data integrity but alters timing behavior. This work aims to detect QoS degradation independent of the network configuration while ensuring zero false positives, meaning no alerting during normal behavior. We present two mechanisms to *capture* timing information and three algorithms to *detect* degradation based on this data.

Figure 1 shows an example of QoS degradation for cyclic messages. Typically, time-critical traffic follows a known cycle time. In the example, the expected cycle time is 10 ms. Message M_1 occurs at 0 ms, and M_2 at 10 ms. However, the third message M_3 arrives at 22 ms (M_3') instead of the expected 20 ms, resulting in a cycle time of 12 ms and a jitter of 2 ms. Such subtle deviations are difficult to detect for end devices, especially if the time synchronization is manipulated.

Most industrial applications detect increased delays and stop their operation if delays exceed predefined thresholds. However, detection becomes non-trivial when time synchronization is attacked. Additionally, the network management system should detect QoS degradation before critical processes are impacted. Therefore, this work proposes a network monitoring solution that detects QoS degradation independently of network configuration, comprising traffic sampling and degradation detection. The detection aims for zero false positives while reliably identifying early stages of degradation.

In the following, we first analyze the impact and causes of QoS degradation (Section II), before presenting the system model and timing extraction methods (Section III). We then introduce and compare three different detection algorithms (Section IV), and evaluate performance under various degradation scenarios (Section V). Finally, we discuss related work (Section VI) and conclude with future research (Section VII).

II. QoS Degradation

In real-time applications such as motion control, robotics, and industrial automation, consistent and predictable communication timing is essential for correct and reliable system

Accepted for publication in Proceedings of *30th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Porto, Portugal, September 9-12, 2025

behavior. Even when most communication operates within expected limits, intermittent delays can still significantly impact the process quality, depending on how often they occur and how large the timing deviation is.

A. Degradation Impact

Consider a welding robot with a camera that provides visual input for seam tracking [5], [6]. The control system relies on precise transmission timing to coordinate camera data processing, robot motion, and welding torch actuation. An end device can detect delayed packets and treat them as lost if they are correctly synchronized with the network. However, if delays are small, such as 1% of the cycle time, minor misalignments between visual feedback and motion may occur, resulting in slight deviations along the weld path. As delays grow to 5% or 10%, the robot may react to outdated camera frames, leading to visible welding errors, excessive corrections, or welds outside the intended track. Infrequent but significant delays can also cause mistimed weld starts or stops, especially when vision-based adjustments must be made on the fly. These issues compromise weld quality, increase stress on mechanical components, and reduce production efficiency.

Typically, QoS degradation begins subtly, with only a few packets being delayed by a small amount. Systems may continue operating without timely detection with silent performance loss until the impact becomes severe or disruptive. Early detection of such degradation is critical, allowing systems to react by adapting control logic, buffering input, or alerting operators before quality or safety is compromised.

B. Degradation Sources

Regular Ethernet provides no timing guarantees, as devices send data whenever needed, and delays occur if the network is busy. However, under constant load conditions, delay behavior appears static. A single misbehaving device that sends too much or too often can delay critical messages and disrupt system behavior (cf. *Misbehaving Device* in Figure 2). Time-sensitive applications require lower and guaranteed delays. TSN extends Ethernet to support real-time communication by adding time synchronization, scheduled traffic, and preemption features. These mechanisms enable precise control over data flows but also increase complexity in configuration and deployment (cf. *Misconfigured Switch* in Figure 2), making networks vulnerable to interference by misbehaving devices.

Attackers can intentionally cause similar effects. As a man-in-the-middle, an attacker can inject additional traffic, delay specific packets, or subtly manipulate the time synchronization to degrade QoS without modifying packet contents (cf. *Attacker* in Figure 2). Although the content remains unchanged, application traffic becomes mistimed, disrupting time-critical communication. Detection is particularly challenging when injected traffic is well-formed and time synchronization is manipulated to hide timing anomalies.

Finally, physical failures such as broken links can introduce QoS degradation as well (cf. *Corrupt Link* in Figure 2). Broken equipment typically results in packet loss due to CRC errors, allowing applications to detect these failures through payload counters. However, such failures can also cause subtle degradation before a total link loss occurs.

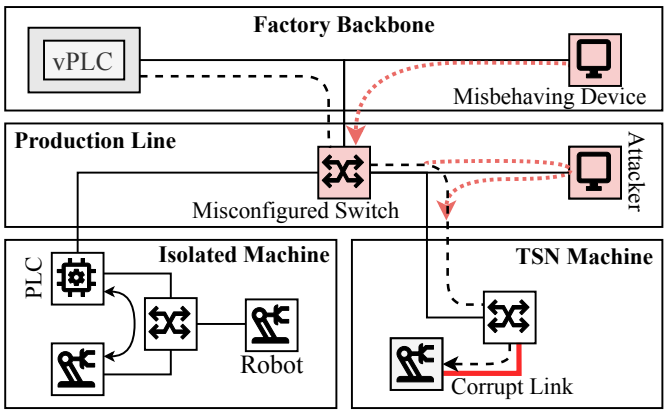


Fig. 2: Potential sources of QoS degradation.

III. System Model

This section describes the four stages of the system model to detect QoS degradation in time-sensitive networks:

1) Collection of timing information,
2) Extraction of timing information,
3) Benchmarking of normal operation, and
4) Detecting QoS degradation

Collection and extraction are continuous tasks during both benchmarking and detection. Benchmarking initializes thresholds based on observed traffic, while the detection compares the measurements to these thresholds to identify anomalies.

In the following sections, we detail the collection and extraction of the data. Afterwards, we highlight the benchmarking and detection operation. We detail three benchmarking and detection algorithms in Section IV. Finally, we conclude this section with requirements for detection performance.

A. Collection

The system captures timing information from the network to evaluate the timing of packets. The closer the measurement point is to the receiver, the more accurately it reflects the observed behavior. Different collection methods offer different timing precision, which influences detection quality. We propose two methods for collecting timing information: passive traffic analysis via port mirroring and enhanced active measurements with in-band timestamping.

1) Passive Traffic Analysis via Port Mirroring: Port mirroring enables non-intrusive observation of existing traffic in both industrial and IT environments. The network card adds a receive timestamp based on a local clock to the metadata of every packet, allowing for time-frequency analysis.

This work uses the Switch Port Analyzer (SPAN) feature, where the monitoring station is directly attached to the mirroring switch. Using SPAN with low-budget network cards, we observed a jitter of 0.5 ms and a standard deviation of 75 µs for traffic with a cycle time of 5 ms. This jitter results from cross traffic on the network, the software stack of the monitoring

device, and port mirroring itself. The detection algorithms must tolerate this imprecision to avoid false positives. More precise capture with hardware timestamping network cards is possible, but it increases deployment costs.

2) Enhanced OAM-based Timing Monitoring: We use active measurements based on Orchestration, Administration, and Maintenance (OAM) protocols to complement passive monitoring. Existing OAM standards such as RFC 7456 [7] and IEEE 802.1ag [8] provide latency measurements. However, they do not capture timing at intermediate points.

We extend OAM-based monitoring with in-band timestamping at the PHY layer of switches, leveraging hardware features originally intended for one-step time synchronization. Specifically, we inject measurement packets that carry transmission timestamps inserted by switches along the path. The injected packets emulate characteristics of critical traffic, such as priority, size, and cyclic behavior. The OAM flows are registered in the network management system to avoid interfering with critical traffic, similar to IEEE 802.1Qdj [9].

In our prototype, the hardware-assisted mechanism achieved a jitter of 5 µs and a standard deviation of 1 µs, caused by other traffic in the network. Although more precise than SPAN-based monitoring, this method requires modifications to network configuration and hardware support for timestamping, making this approach less applicable.

B. Extraction

The monitoring system extracts timing parameters from observed network streams. Each application stream generates a time series based on cycle time and jitter measurements. Depending on the detection algorithm, the system uses either only cycle time, only jitter, or a combination of both.

1) Cycle Time: The cycle time is the time difference between two consecutive packets (cf. Figure 1). The system calculates it based on receive timestamps from mirrored traffic or timestamps embedded in OAM packets. If multiple timestamps are available, such as in OAM-based monitoring, the system can extract cycle times from different locations along the path. Cycle time extraction does not require time synchronization, as the local clock drift between two consecutive packets is negligible. However, packet loss affects extraction, as missing packets lead to measured cycle times being multiples of the expected cycle.

2) Jitter: Jitter is the deviation of the reception time of a packet from its expected reception time within the cycle. During benchmarking, the system establishes the expected reception time by calculating the mean offset of packets within the cycle. Jitter extraction requires time synchronization, as the device must interpret absolute time values. While cycle time extraction is sensitive to packet loss, jitter extraction is not. However, jitter measurements can be manipulated if an attacker compromises the time synchronization. Specifically, an attacker can adjust the observed jitter to hide QoS degradation.

C. Benchmarking

Benchmarking is the initial phase of the detection system. During this phase, we assume no attacker or misbehavior is present in the network. The goal of benchmarking is to establish detection thresholds based on observed timing behavior, ensuring independence from specific network configurations.

We execute the benchmarking algorithm during the first 20 seconds of operation (4,000 packets with a cycle time of 5 ms). During this time, the system continuously collects timing information and extracts cycle time and jitter values. Based on these observations, it calculates statistical measures such as mean, standard deviation, minimum, and maximum values. These values define the thresholds used later for detecting deviations from normal behavior.

Each detection mechanism applies the same calculation methods during benchmarking as it does during detection, but updates the statistical measures only during the benchmarking phase. After benchmarking concludes, the detection phase begins and runs continuously.

D. Detection

During detection, the system applies the same calculations as during the benchmarking but does not update the statistical measures. Instead, it compares the new measurements against the established thresholds. A value that exceeds a threshold is classified as a *positive* detection. Otherwise, it is classified as *negative*. Depending on the correctness, each value is categorized as *True Positive (TP)*, *False Positive (FP)*, *True Negative (TN)*, or *False Negative (FN)*.

E. Requirements

The detection system must achieve a false positive rate of zero, as false alarms require costly manual analysis and reduce practical applicability. At the same time, it must minimize false negatives and reliably detect QoS degradation. Specifically, the detection rate must approach 100% with increased deviations and delays, ensuring early identification of critical issues.

IV. Detecting QoS Degradation

Based on the previous section, we assume we have the timing information from a SPAN measurement. In the following, we introduce three different methods for detecting QoS degradation: A) single-value thresholds, B) moving averages, and C) distribution-based analysis. This section uses reference degradation scenarios to discuss the behavior, pros, and cons of the three mechanisms. We use the same captured traffic for all visualized detections. In Section V, we introduce the testbed and we evaluate the most promising methods with more variation in the degradation and variance in the delay.

A. Single Value Detection

We use the detection based on a single value as a baseline algorithm, as it is intuitive and simple to implement. During the benchmarking, this algorithm combines all extracted cycle times or jitter and calculates the $mean$, standard deviation (std), minimum (min), and maximum (max) value. We calculate the thresholds with these values, as visualized in Figure 3 with the dashed horizontal lines. The subfigure a) visualizes the thresholds $mean \pm (2 \cdot std)$, whereas the subfigure b) uses

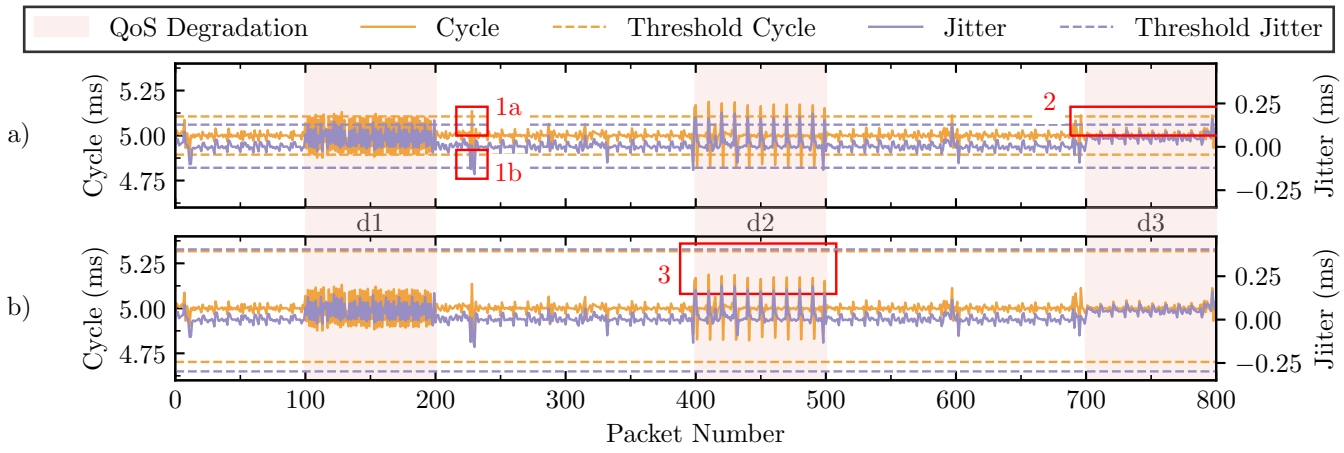


Fig. 3: Detection of time degradations during three scenarios based on a single packet with two different thresholds: a) $mean \pm (2 \cdot std)$ and b) min/max. Degradation delay every 2nd packet by 2% (*d1*), 10th by 5% (*d2*), and every packet by 1% (*d3*).

the min and max for the presented thresholds. The y-axis on the left measures the cycle time, and the y-axis on the right measures the jitter. On the x-axis, Figure 3 presents the packet number. All values between the threshold lines indicate regular behavior, while values above or below the window between the threshold lines indicate QoS degradation. The traffic suffered QoS degradation within the areas with a red background, and the presented monitoring system should detect it. The first degradation *d1* delays every second packet by 2% of the cycle time, which is a very intense degradation of QoS. The second scenario *d2* delays every 10th packet by 5%. The third scenario *d3* delays every packet by only 1%. Outside these areas, the traffic has a similar timing as the benchmarking phase.

Figure 3a shows that this intuitive single-value comparison is too sensitive to outliers (cf. markers 1a and 1b). Specifically, the outliers in the regular behavior have a similar height as the introduced degradation. Independent of the choice of the formula and factor for the standard deviation, the threshold selection will always result in false positives (cf. markers 1a and 1b) and false negatives (cf. marker 2). The algorithm will have even more false negatives if the threshold is adjusted to have no false positives (cf. marker 3). Therefore, the detection based on single values is insufficient for a good IDS. The algorithm must be less sensitive to outliers to fulfill the requirements of zero false positives and low false negatives. In the next section, we introduce the detection based on the exponential moving average to smooth the detection.

B. Exponential Moving Average

The exponential moving average (EMA) is a simple and efficient method for smoothing time series data. It applies exponential decay to past values, giving more weight to recent samples. EMA updates recursively and does not require storing historical data, making it ideal for real-time systems. We use EMA to calculate changes in jitter and cycle times for a single packet based on the previous traffic. Figure 4 visualizes the results for two different EMA configurations.

Equation 1 describes the EMA computation at time step i. X_i is the current sample, A_{i-1} is the previous average, and

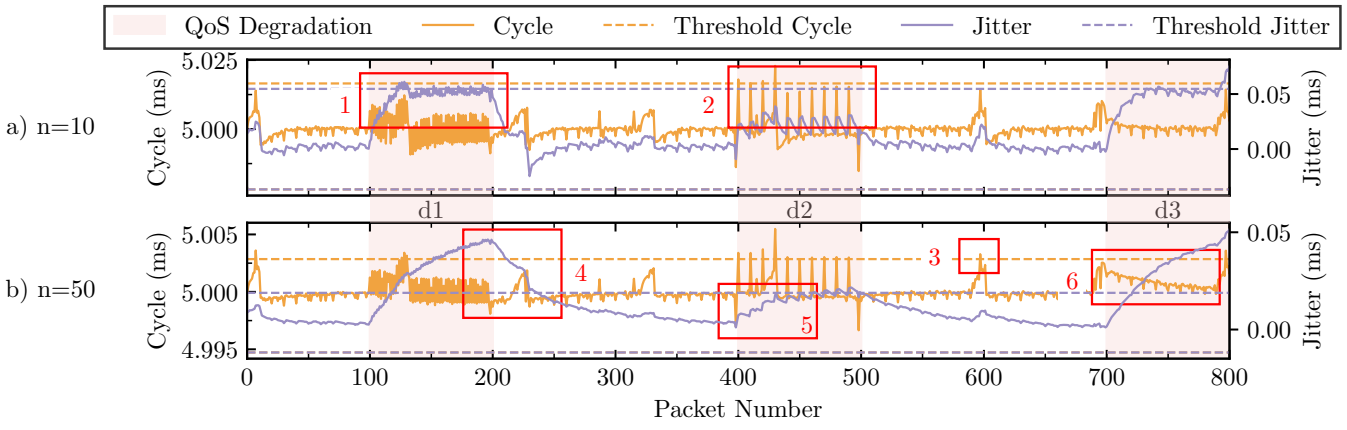


Fig. 4: Detection of time degradation during three scenarios based on exponential moving average (EMA) with two different memory sizes n: a) $n = 10$ and b) $n = 50$ packets. Threshold: min/max. Same degradation scenarios as in Figure 3.

$\alpha \in (0, 1]$ is the smoothing factor that controls the memory. In addition, Menth et al. [10] proposed a common memory definition for moving averages: $\alpha = 1 - 1/memory$.

$$A_i = \alpha \cdot A_{i-1} + (1 - \alpha) \cdot X_i \tag{1}$$

During the benchmarking phase, we apply the EMA to all cycle times and jitter values to calculate the min and max values we use as thresholds. Figure 4 shows the application of the EMA during the detection with different memory sizes: subfigure a) 10 packets and subfigure b) 50 packets. Important to note that the different memory sizes result in different scales for the two subfigures. In both figures, the cycle time has thin spikes during degradation (cf. orange line at marker 2), as one cycle time is larger than the average for a delayed packet and smaller by the same amount for the next packet, which is not delayed. Hence, the average only changes for the first packet and is reset with the second packet. The EMA for the cycle time is always close to the desired cycle time, and only single packets exceed the threshold. It is similar to the previous approach and sensitive to outliers (cf. marker 3 with false positives) and false negatives for the cycle time at marker 1. Additionally, if every packet is delayed, the detection based on the cycle time cannot reveal the degradation (cf. orange line in marker 6). The jitter is a per-packet observation, and a delayed packet does not impact the following jitter. Therefore, the moving average on jitter indicates areas with multiple packets suffering QoS degradation (cf. jitter at marker 1). The larger memory makes the algorithm less sensitive to outliers, resulting in flatter slopes during detection, i.e., requires a longer time until a deviation is alerted (cf. jitter at marker 5) and until detection is reset to normal (cf. jitter at marker 4).

Concluding the introduction of the EMA to detect QoS degradation, we notice a sensitive behavior of cycle time, independent of the memory size, making it insufficient for the goal of having zero false positives. For the EMA based on jitter, sparse degradation (cf. *d2*) results in only a few degradation samples per memory and thus is difficult to detect. Hence, the EMA based on jitter with a larger memory size is less sensitive to outliers and detects sparse degradation more reliably, but has a longer delay until the detection.

C. Distribution-based Detection

To reduce the sensitivity to single values, e.g., the cycle time in the detection with the EMA, we compare the distribution of values in the benchmarking phase and the detection phase. This work uses the Wasserstein distance (WD), which calculates the distance between two distributions for single and multi-dimensional values. The WD, also known as *Earth Mover's Distance*, measures how much effort is needed to transform one distribution into another, by considering *location*, *spread*, and *shape*. This is especially useful for detecting subtle shifts in timing behavior.

The first-order Wasserstein distance between two sorted samples $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ is given by Equation 2. P and Q are uniform distributions over sorted samples $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$, respectively. The benchmarking phase starts with creating the reference distribution of size n. During the rest of the benchmarking phase, we compare the distribution of the last n received packets to this reference distribution. From this comparison, we store the max distance as the threshold for the detection.

$$W_1(P, Q) = \frac{1}{n} \sum_{i=1}^{n} |x_i - y_i| \tag{2}$$

Figure 5 shows the application of the WD during the detection with different distribution sizes: subfigure a) 10 packets and subfigure b) 50 packets. In addition to the previous plots, we present the WD applied to the combination of cycle time and jitter as a two-dimensional distribution. Important to note is that the subfigures use the same scale, i.e., the detected differences are in the same order, but the thresholds are lower for the larger distribution size. Hence, larger distribution sizes result in more confidence in the detection with WD and remove the impact of single outliers, as presented by

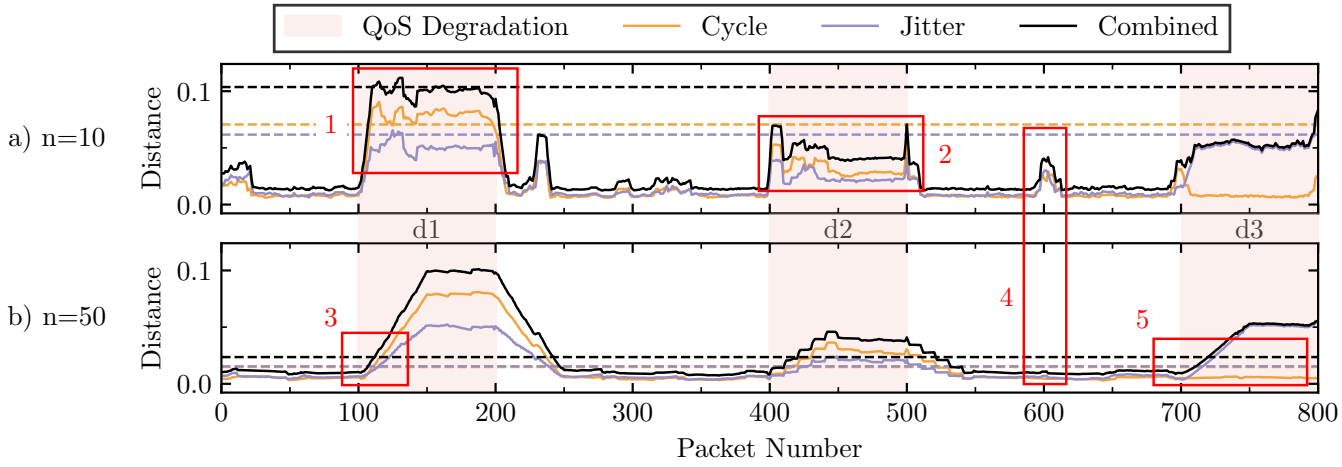


Fig. 5: Detection of time degradation during three scenarios based on Wasserstein distance (WD) with two different distribution sizes n: a) $n = 10$ and b) $n = 50$ packets. Threshold: max. Same degradation scenarios as in Figures 3 and 4.

Figure 5b at marker 4. The confidence in the detection is visible by comparing the detection, i.e., barely reaching the threshold at marker 1 and exceeding the threshold at marker 3. Marker 2 highlights the potential of false negatives in small distribution sizes due to the large threshold. Similar to the detection based on single values and EMA, the detection solely on the cycle time cannot detect the delay of all packets (cf. marker 5). However, for all other degradations, the WD algorithm can operate solely on the cycle time measurements and is independent of the time synchronization.

V. Evaluation

In the previous section, we introduced three methods to identify QoS degradation in the timing information of packet traces. In this section, we evaluate the most promising methods in more detail: the EMA based on jitter and the WD based on jitter, cycle time, and the combination of cycle time and jitter. First, we briefly describe the testbed used to gather all samples for the data presented in this paper. Second, we execute parameter studies to discuss the benefits and downsides of the four approaches. Finally, we use traffic with delay as a normal distribution to evaluate less structured QoS degradation.

A. Testbed

We evaluate the effectiveness of the network monitoring solution in a testbed that represents a small industrial network, as visualized in Figure 7. The testbed comprises four synchronized Hirschmann RSPE35 switches and a real-time OPC UA application. The cyclic real-time traffic stream with a cycle time of 5 ms is based on the open-source open62541 pub-sub application developed by Pfrommer et al. [11]. This transmission is similar to the communication between a controller and an IO system in a discrete manufacturing line using, e.g., PROFINET traffic. We simulate the QoS degradation at

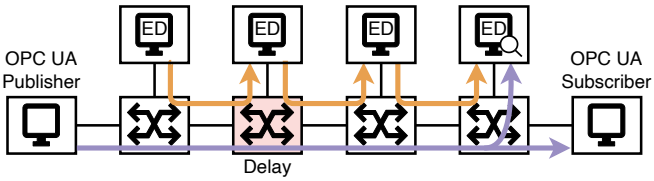
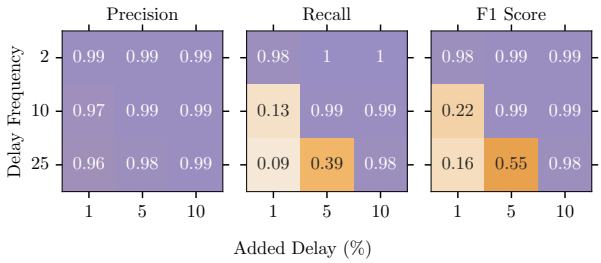


Fig. 7: Testbed for evaluating QoS degradation.

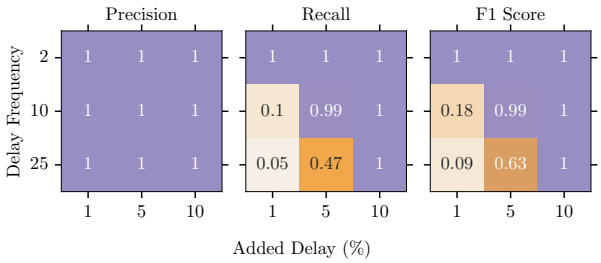
the second switch by delaying every n-th packet. End devices (cf. *ED* in Figure 7) send cross traffic with 10 Mbit/s at every switch with the traffic generation tool iPerf version 2.1.5 to represent background traffic, e.g., SCADA. At the last switch, we mirror the traffic with SPAN and record the reception timestamps with tcpdump version 4.99.2 and libpcap version 1.10.2. The percentage of added delay results in smaller jitter for smaller intervals, making detection more challenging.

B. Parameter Studies

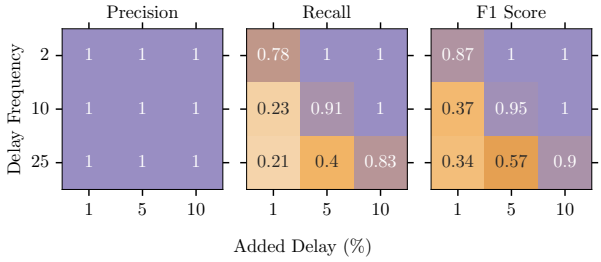
We use a structured evaluation with different parameters on the QoS degradation to analyze the detection performance in different scenarios. Throughout the evaluation, we modified the added delay and the frequency of delayed packets. Figure 6 presents the applied delay to every nth = {2, 10, and 25} packet to determine the impact of different QoS degradation scenarios. Additionally, we apply a delay of varying intensity = {1%, 5%, and 10%}, resulting in nominal values for a 5 ms cycle time of 0.05 ms, 0.25 ms, and 0.5 ms, respectively. We determine the effectiveness of the detection methods by the ratio of correctly and incorrectly classified values (cf. TP, FP, TN, and FN in Section III-D). We evaluate detection performance using precision, recall, and F1-Score (Equations 3 to 5). Precision measures correct positive detections, recall measures the detection rate of actual degradations, and the F1-Score combines both into a single metric. The system requires



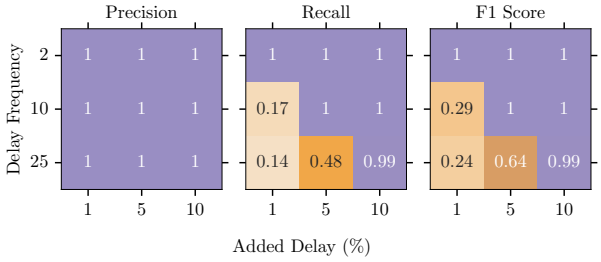
(a) EMA based on jitter.



(b) WD based on cycle time.



(c) WD based on jitter.



(d) WD based on the combination of cycle time and jitter.

Fig. 6: Detecting QoS degradation in different scenarios with the EMA based on jitter and WD based on cycle time, jitter, or the combination of both. Memory/distribution size: 50; Threshold: min/max.

precision to equal 1 and a recall close to 1, specifically for high-frequency, high-delay degradations.

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

$$Recall = \frac{TP}{TP + FN} \tag{4}$$

$$F_1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{5}$$

Figure 6 presents the detection results for the QoS degradation variations. Each subfigure presents the $Precision$, $Recall$, and the $F1$-Score for each evaluation scenario. For each of the added delay and frequency combinations, we used 1.000 degradation samples captured in our testbed, always combined with the same duration of non-degrading traffic to have a cross-check for false positives. We set the detection thresholds during the baseline phase to the max and min detected values. We generally observe that all four detection methods more precisely detect larger delays and higher frequencies. Specifically, all mechanisms identify all degradations for these scenarios (cf. recall heatmaps). As discussed in Section II, the infrequent delay and small delay scenarios do not impact the performance and quality of a machine (cf. left column and bottom row). Still, they typically are the start of a degradation process. Hence, early detection is achieved by detecting the samples at the bottom left of the heatmaps. Based on the jitter, the WD achieves the best results for these small degradations. For applicability in industrial networks, a small false positive rate is important. Someone must analyze the situation and verify the reported state for any false positives causing alert fatigue. In the executed test runs, the WD mechanisms never detected any false positives, while the EMA has a precision of 0.96.

We propose to use the WD based on the combination of jitter and cycle time. If the degradation or an attacker affects the time synchronization, such that the jitter aligns with a wrong clock, the detection system still has the performance of the WD based on the cycle time.

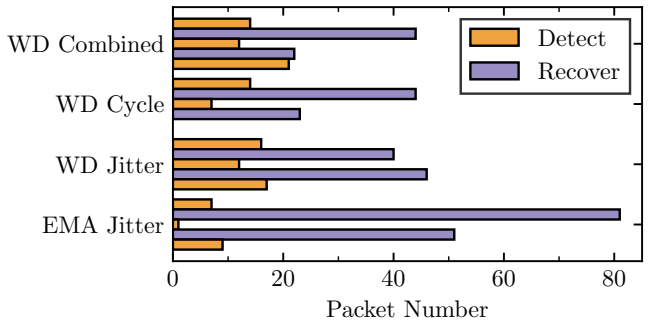


Fig. 9: Delay by packets between degradation start and detection, as well as between degradation end and recovery. Values per method from top to bottom from *d1* to *d3*.

C. Normally Distributed QoS Degradation

As the previous measurements used structured and constant delays, we also execute measurements with non-static QoS degradations. For that, the added delay follows a normal distribution, but has a similar range as the previously discussed scenarios. Figure 8 presents the detection in these scenarios for the EMA and WD in a single graph with the WD scale on the left and EMA scale on the right. The combined figure enables the discussion on the detection speed, where the EMA presents a more sensitive behavior (cf. marker 4). For each method, Figure 9 shows from top to bottom the detection and recovery delay at *d1* and *d2*, as well as the detection delay at *d3*. The EMA jitter is the fastest in detection and the slowest in recovery. All WD detection algorithms have a slightly larger delay for the detection, but recover faster than the EMA. Specifically, the detection speed is important to observe short degradation phases and react as soon as possible. However, the EMA also detects multiple false positives as degradation (cf. marker 3), as the detection for the EMA takes longer to return to normal (cf. marker 2). The three WD detection mechanisms perform similarly, with the jitter being less sensitive in the first degradation scenario (cf. marker 1). Hence, WD proves to have the best applicability in dynamic environments.

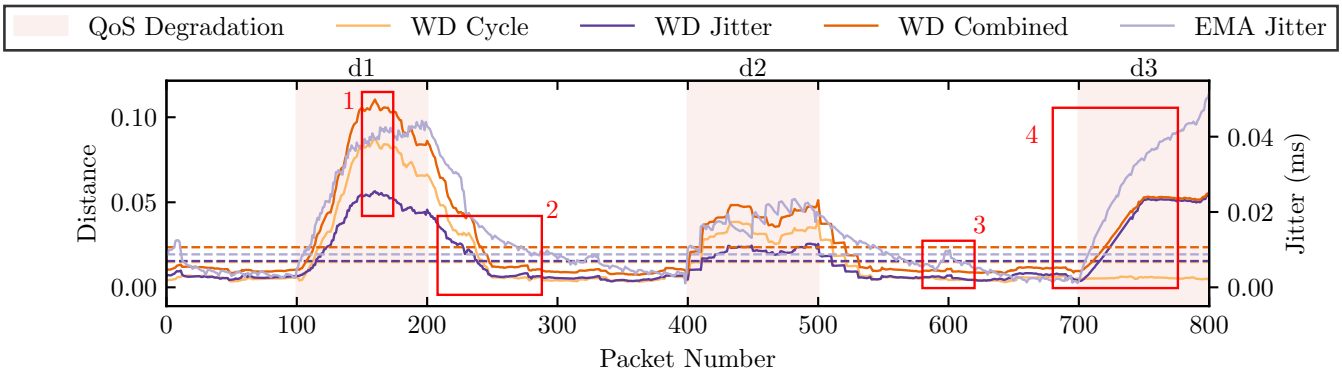


Fig. 8: Detecting QoS degradation in different scenarios with the exponential moving average (EMA) based on jitter and Wasserstein distance (WD) based on cycle time, jitter, or the combination of both. Memory or distribution size: 50; Threshold: min/max. Similar degradation scenarios as in Figure 3, 4, and 5, but with a normally distributed added delay.

VI. Related Work

Time-critical traffic streams are high-priority attack targets, as delayed or lost packages can disrupt the production process. To protect these critical streams, the TSN Task Group developed the IEEE 802.1Qci Per-Stream Filering and Policing (PSFP) standard [2]. Luo et al. [12] and Meyer et al. [13] developed anomaly-based detection systems for Time Sensitive Networking (TSN) that rely on PSFP for attack detection. The authors evaluated their IDS in an OMNeT++ simulation, where the IDS identified and discarded all abnormal traffic events with high accuracy. Their work is limited by the assumption that PSFP is supported on every port of every switch in the network. The same applies to earlier work from Meyer et al. [14], where the authors achieved DoS protection with credit-based metering and PSFP. However, current infrastructure in industrial environments does not support PSFP or has a limited number of filtering rules and thus is not feasible in large networks [4]. Bülbül et al. [15] proposed TSN-Gatekeeper, a Software Defined Networking (SDN) ingress filter approach similar to PSFP. Similarly, Ihle et al. [16] implements P4-PSFP, i.e., a fully IEEE 802.1Qci standard-conform P4-based PSFP implementation. Unfortunately, both implementations are limited to data plane programmability with P4 and thus are not applicable in industrial networks.

Ergenc et al. [17] proposed an open-source IDS for TSN called TSNZeek. Their IDS can accurately detect attacks on TSN protocols like the Stream Reservation Protocol (SRP) and the fault tolerance mechanism Frame Replication and Elimination for Reliability (FRER). However, their IDS is limited to these two protocols and fails to detect delay attacks on time-critical traffic streams. Zhang et al. [18] developed TSN-Peeper, a network monitoring solution that relies on in-band telemetry data to detect forwarding misbehavior of TSN streams. Similar to this work, Chou et al. [19] present the analysis of the cycle time but lack algorithms and evaluation.

None of the existing literature provides a configuration-independent, widely available solution for QoS degradation detection in time-critical traffic. Existing solutions rely on specific standards or are limited to specific attacks or protocols. We close this gap and provide a practical, standard agnostic monitoring solution that detects QoS degradation.

VII. Conclusion and Future Work

This work presents a configuration-independent detection system for identifying QoS degradation in time-sensitive networks. We introduce passive and active timing measurements to capture traffic behavior and evaluate three statistical detection algorithms. Among them, the Wasserstein distance achieves the most reliable results, detecting degradation early while remaining robust against minor jitter fluctuations.

Operating independently of network configuration and application semantics, the system remains applicable even in heterogeneous industrial environments. Our evaluation confirms that early-stage degradations, often precursors to critical failures, can be reliably detected. For the majority of degradations, detection does not require time synchronization. Future work will refine threshold selection and validate the system across diverse industrial environments to bridge the gap between experimental and operational deployments.

References

[1] D. Ergenç, C. Brülhart, J. Neumann, L. Krüger, and M. Fischer, "On the Security of IEEE 802.1 Time-Sensitive Networking," in *IEEE International Conference on Communications Workshops (ICC Workshops)*, Montreal, QC, Canada, 2021.

[2] "IEEE Standard for Local and metropolitan area networks–Bridges and Bridged Networks–Amendment 28: Per-Stream Filtering and Policing," *IEEE Std 802.1Qci-2017*, pp. 1–65, 2017.

[3] A. Grigorjew, S. Geißler, P. Diederich, T. Hoßfeld, and W. Kellerer, "Resilience in Time-Sensitive Networking: An Overview and Open Research Directions," in *International Workshop on Resilient Networks Design and Modeling (RNDM)*, Hamburg, Germany, Sep. 2023.

[4] X. Jiang, X. Yang, T. Zhou, W. Fu, W. Quan, Y. Jiao, Y. Sun, and Z. Sun, "Towards Memory-Efficient Traffic Policing in Time-Sensitive Networking," *arXiv preprint arXiv:2403.01652*, 2024.

[5] T. Striffler and H. D. Schotten, "The 5G Transparent Clock: Synchronization Errors in Integrated 5G-TSN Industrial Networks," in *International Conference on Industrial Informatics (INDIN)*, Palma de Mallorca, Spain, Jul. 2021.

[6] J. Haxhibeqiri, P. Avila-Campos, I. Moerman, and J. Hoebeke, "Optimizing Scheduling in Wireless TSN Utilizing Genetic Algorithms," in *International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Paris, France, Oct. 2024.

[7] T. Mizrahi, T. Senevirathne, S. Salam, D. Kumar, and D. E. E. 3rd, "Loss and Delay Measurement in Transparent Interconnection of Lots of Links (TRILL)," RFC 7456, Mar. 2015. [Online]. Available: https://www.rfc-editor.org/info/rfc7456

[8] "IEEE Standard for Local and Metropolitan Area Networks Virtual Bridged Local Area Networks Amendment 5: Connectivity Fault Management," *IEEE Std 802.1ag - 2007*, pp. 1–260, 2007.

[9] "IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks Amendment 38: Configuration Enhancements for Time-Sensitive Networking," *IEEE Std 802.1Qdj-2024*, pp. 1–56, 2024.

[10] M. Menth and F. Hauser, "On Moving Averages, Histograms and Time-DependentRates for Online Measurement," in *Proceedings of ACM/SPEC on International Conference on Performance Engineering*, L'Aquila, Italy, Apr. 2017.

[11] J. Pfrommer, A. Ebner, S. Ravikumar, and B. Karunakaran, "Open source OPC UA PubSub over TSN for realtime industrial communication," in *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Turin, Italy, Sep. 2018.

[12] F. Luo, B. Wang, Z. Fang, Z. Yang, and Y. Jiang, "Security Analysis of the TSN Backbone Architecture and Anomaly Detection System Design Based on IEEE 802.1Qci," *Security and Communication Networks*, vol. 2021, no. 1, 2021.

[13] P. Meyer, T. Häckel, F. Korf, and T. C. Schmidt, "Network Anomaly Detection in Cars based on Time-Sensitive Ingress Control," in *IEEE Vehicular Technology Conference (VTC)*, Victoria, Canada, Nov. 2020.

[14] P. Meyer, T. Häckel, F. Korf, and T. C. Schmidt, "DoS Protection through Credit Based Metering-Simulation-Based Evaluation for Time-Sensitive Networking in Cars," *Proceedings of 6th International OMNet++ Community Summit*, vol. 66, 2019.

[15] N. S. Bülbül, J. J. Krüger, and M. Fischer, "TSN Gatekeeper: Enforcing stream reservations via P4-based in-network filtering," in *IFIP Networking Conference*, Barcelona, Spain, Jun. 2023.

[16] F. Ihle, S. Lindner, and M. Menth, "P4-PSFP: P4-Based Per-Stream Filtering and Policing for Time-Sensitive Networking," *IEEE Transactions on Network and Service Management*, vol. 21, no. 5, 2024.

[17] D. Ergenç, R. Schenderlein, and M. Fischer, "TSNZeek: An Open-source Intrusion Detection System for IEEE 802.1 Time-sensitive Networking," in *IFIP Networking Conference*, Barcelona, Spain, Jun. 2023.

[18] C. Zhang, B. Zhou, Z. Tian, L. Cheng, Y. Liu, H. Zhang, S. Chen, Y. Wan, W. Xu, T. Pan, Y. Xu, Y. Wang, H. Zhu, and B. Liu, "TSN-Peeper: an Efficient Traffic Monitor in Time-Sensitive Networking," in *IEEE International Conference on Network Protocols (ICNP)*, Lexington, KY, USA, Oct. 2022.

[19] Y.-K. Chou and B.-C. Cheng, "Intrusion Detection for Time-Sensitive Industrial Control Systems," *Communications of the CCISA*, vol. 30, no. 1, 2024.