TSNCard: Bridging the Gap in TSN Diagnostics via Protocol, Algorithm, and Hardware

Xiangwen Zhuge, *Student Member, IEEE*, Zeyu Wang, *Student Member, IEEE*, Xiaowu He, *Member, IEEE*, Shen Xu, Fan Dang, *Senior Member, IEEE*, Jingao Xu, *Member, IEEE*, Zheng Yang, *Fellow, IEEE*, and Qiang Ma, *Member, IEEE*

***Abstract*—Time-Sensitive Networking (TSN) is foreseen as a foundational technology that enables Industry 4.0. It offers deterministic data transmission over Ethernet for critical applications such as industrial control and automotive systems. However, TSN is susceptible to hardware and software errors, necessitating an effective diagnostic system. Traditional network diagnostic tools are inadequate for TSN fault localization and classification due to the tightly coupled traffic and high precision requirements in TSN. In response, this paper presents TSNCard, a cross-cycle postcard-based diagnostic system tailored for Time-Aware Shaper (IEEE 802.1 Qbv) in TSN. TSNCard introduces a novel telemetry protocol that leverages the cyclical nature of TSN networks for data collection at each node. This protocol, coupled with dedicated analytic algorithms and hardware innovations within switches, forms a comprehensive system for TSN monitoring, fault localization and classification. Extensive experiments on both simulation and physical testbeds show that TSNCard can 100% detect fault location and type of the TSN misbehavior while adhering to industrial bandwidth restrictions. TSNCard not only bridges the gap in the TSN protocol stack, but also serves as a versatile toolkit for time-synchronized network analysis, paving the way for future research. The code is available at https://github.com/MobiSense/TSNCard**

***Index Terms*—Time-sensitive networking, network diagnostics, industrial networking.**

I. Introduction

TIME-SENSITIVE Networking (TSN) is recognized as a foundational technology towards Industry 4.0 [1]. It enables deterministic data transmission of time-sensitive traffic alongside best-effort traffic in a time-synchronized Ethernet network, catering to time-critical applications like industrial control and automotive. To achieve this, TSN operates on the principle of time-division multiplexing. A centralized scheduler calculates the precise forwarding time of all critical flows at each hop, preventing any potential conflicts [2].

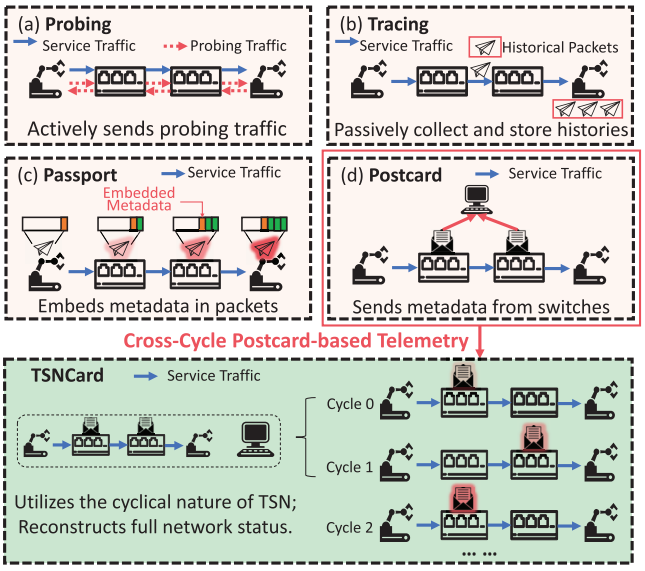


Fig. 1. A comparison of network diagnostic tools.

Despite its precision, TSN is susceptible to hardware and software errors similar to traditional network devices, resulting in time synchronization errors, incorrect forwarding time, or packet disorder. In industrial scenarios where TSN is essential, swiftly localizing and rectifying such failures is imperative to maintain the stable operation of production environments, while identify the fault type is crucial for the long-term maintenance and upgrades of production lines [3], [4]. However, current TSN standards typically rely on static configuration issued by a central controller, lacking a closed-loop control system for monitoring, analysis, and fault diagnosis. Existing preliminary efforts can only detect the misbehavior without fault localization or classification, and they remain largely theoretical without practical implementation [5], [6].

In traditional Ethernet, especially data center networks, there are an abundance of methods for traffic analysis and fault diagnostics. As shown in Figure 1, they can be broadly categorized into *probing* [7], [8], [9], [10], *tracing* [11], [12], [13], *passport-based telemetry* [14], [15], [16], and *postcard-*

Received 23 December 2024; revised 14 April 2025; accepted 28 May 2025; approved by IEEE TRANSACTIONS ON NETWORKING Editor S. Alouf. This work was supported in part by the National Key Research Plan under Grant 2021YFB2900100 and in part by the NSFC under Grant 62302259 and Grant 62302254. A preliminary version of this article appeared in the IEEE/ACM International Symposium on Quality of Service (IEEE/ACM IWQoS 2024) [DOI: 10.1109/IWQoS61813.2024.10682902]. *(Corresponding author: Zheng Yang.)*

Xiangwen Zhuge, Zeyu Wang, Xiaowu He, Shen Xu, and Zheng Yang are with the School of Software and BNRist, Tsinghua University, Beijing 100084, China (e-mail: zgxw18@gmail.com; ycdfwzy@gmail.com; horacehxw@gmail.com; xshenx234@gmail.com; hmilyyz@gmail.com).

Fan Dang is with the School of Software Engineering, Beijing Jiaotong University, Beijing 100084, China (e-mail: dangfan@bjtu.edu.cn).

Jingao Xu is with the Computer Science Department, Carnegie Mellon University, Pittsburgh, PS 15203 USA (e-mail: xujingao13@gmail.com).

Qiang Ma is with the Qiyuan Lab, Beijing 100084, China (e-mail: tsinghuamq@gmail.com).

Digital Object Identifier 10.1109/TON.2025.3575508

TABLE I
PERFORMANCE OF EXISTING METHODS AND TSNCARD ADDRESSING THE CHALLENGES IN TSN DIAGNOSTICS

	Tracing	*Probing*	*Passport*	*Traditional Postcard*	***TSNCard***
Error Localization	N/A	✗	✗	✗	✓
Bandwidth Cost	N/A	14Mbps	15Mbps	59Mbps	**1Mbps**
Measurement Acc.	N/A	$92\mu s$	$178\mu s$	$163\mu s$	$< \mathbf{0.1}\mu s$

based telemetry [17], [18], [19]. These methods typically encompass three major stages to perform network diagnostics: (i) traffic measurement stage that monitors and records real-time statistics for target data streams; (ii) data collection stage that stores or collects the statistics necessary for diagnostics; (iii) and subsequent issue analytics stage that diagnoses the network issues based on collected information.

Nevertheless, different from traditional Ethernet's best-effort approach, TSN necessitates predetermined schedule for the critical traffic transmission. It reserves dedicated bandwidth and nanosecond-level precise time slots for critical data frames. This requirement significantly challenges conventional diagnostic methods across their traffic measurement, data collection, and issue analytics stages, as further explored below:

- **Chain of Errors in Issue Analytics Stage.** TSN's time-division multiplexing principle results in tight interdependence among data stream schedules. This tight coupling means that any variance in one stream's transmission can adversely affect others, leading to a chain of errors. The analytics algorithms in traditional methods are not designed for such fine-grained forwarding time discrepancies, thus fall short in identifying fault localization and classification(§II-C-C1).
- **Bandwidth Constraint for Data Collection Stage.** In industrial TSN networks, the majority of the bandwidth is reserved for high-priority business traffic, limiting the room available for network monitoring and diagnostics. This constraints traditional tools, designed for environments with abundant bandwidth, to collect sufficient data for comprehensive analysis (§II-C-C2).
- **Time Inaccuracy during Traffic Measurement Stage.** TSN requires data to be transmitted synchronously with nanosecond-level precision. This requires diagnostic tools to precisely measure critical traffic or send the probing packets to detect the discrepancies between actual transmission time and the predetermined schedule. However, off-the-shelf tools lack the ability to provide highly accurate synchronized timestamps or measure transmission times from hardware logic. As a result, they are insufficient for reliably TSN misbehavior identification (§II-C-C3).

Table I demonstrates some preliminary results in our case study (details in §II-C). We find that all the existing methods are not able to identify the fault location and type in TSN error chains during the analytics stage, calling for a new traffic analysis paradigm. Regarding the data collection stage, TSN requires a centralized global diagnostic view for comprehensive analysis. However, *tracing* methods inherently collect and store data in a distributed manner, rendering them unsuitable for further comparison. Meanwhile, *probing*, *passport*, and *traditional postcard* methods all exceed the bandwidth limitation in industrial scenarios. In terms of traffic measurement stage, existing methods exhibit more than three orders of magnitude worse accuracy than requirement, making their diagnostic information inapplicable.

Therefore, a brand new traffic analysis paradigm is required to tackle TSN's unique requirements. This poses many challenges that are addressed in this work: (i) how to design a cooperation mechanism between switches and the analytic server to gather adequate diagnostic information under various constraints; (ii) how to ensure swift and precise fault localization and classification in complicated error scenarios; and (iii) how to achieve accurate transmission time measurement with high precision and low overhead.

In this paper, we present **TSNCard**, a cross-cycle post**Card**-based diagnostic system for Time-Aware Shaper (IEEE 802.1 Qbv) in **TSN**. TSNCard is the first system for comprehensive TSN fault localization and classification, which systematically addresses the challenges mentioned above. Overall, TSNCard's design and implementation excel across three layers, i.e., protocol, algorithm, and hardware. To be specific:

- **On the protocol front:** we introduce a cross-cycle postcard-based telemetry protocol for TSN. This protocol utilizes the cyclical nature of TSN networks, collecting data frame transmission information across cycles at each hop. It enables the analytic server to reconstruct the necessary state of critical data flows and comprehensively understand the network performance (§IV-A).
- **On the algorithm front:** we design three algorithms on the analytic server. First, a Flow-Centric Fault Refinement algorithm is proposed to identify fault localization of network problems based on the deviation between data frames' scheduled and actual forwarding time. Second, to save analysis overhead, we introduce a Bandwidth-adaptive Postcard Prioritization algorithm to adjust the amount of postcards to collect according to the application requirements. Third, we propose a Critical-Postcard Flow Scheduling algorithm for the postcard traffic to ensure the timely and accurate collection of diagnostic data (§IV-B).
- **On the hardware front:** we propose a suite of hardware technologies within TSN switches and devices. They enable the measurement of high-precision synchronized transmission time without disrupting the flow of critical data frames and send the pre-scheduled packets precisely, supporting the execution of upper-layer algorithms and protocols integral to TSNCard's functionality (§IV-C).

We fully implement TSNCard based on the Xilinx Zynq platform [20] with software and hardware co-design. Comprehensive experiments are carried out on both simulation and physical testbeds with more than 24,000 test cases. The experiment results demonstrate that TSNCard can 100% detect the fault location and type of the time-related TSN misbehavior in any circumstance, operates smoothly under 1Mbps bandwidth limitation, and takes only about $10ms$ to complete diagnostic data collection.

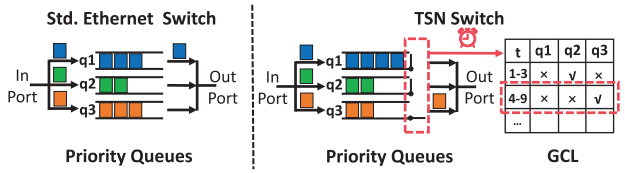


Fig. 2. Standard ethernet switch vs. TSN switch [22].

In summary, this paper makes three contributions.

1) We design and implement TSNCard, as far as we are aware of, the first comprehensive TSN traffic diagnostic system. It is capable of accurately pinpoint fault location and type of forwarding misbehavior. TSNCard effectively resolves the gap in existing TSN standards, enhancing its capabilities for closed-loop traffic analysis.
2) We propose the cross-cycle postcard-based telemetry protocol, along with an array of software and hardware innovations. This comprehensive solution aims to enable precise, robust, and low-overhead fault detection and identification in TSN networks.
3) We extensively evaluate the performance of TSNCard and comparative systems on simulation and physical testbeds. The results demonstrate TSNCard's superior performance.

Contribution to the community. The code is available at https://github.com/MobiSense/TSNCard. The TSNCard enables fault localization classification for any network compatible with 802.1AS [21]. Additionally, we have open-sourced our implementation of the TSN switch and network simulation. We believe these tools will aid future research in TSN networks, particularly in the areas of network monitoring and analysis.

II. Background and Motivation

A. Switch Model of TSN

Figure 2 illustrates the key differences between Standard Ethernet switches and TSN switches. TSN switches and devices share a nanosecond-precision global clock, as specified by IEEE 802.1AS [21]. TSN switches allocate exclusive time slots for each packet of critical traffic to prevent interference from other packets, as defined in IEEE 802.1 Qbv [2] and implemented through Gate Control List (GCL) at packet-level granularity. When a packet is scheduled to send in a time slot, the gate of these frames' queue will be set to open, and the gates of the other queues will be set to close. For instance, the time slot $t = 4 \sim 9$ in Figure 2 is exclusively reserved for the first packet in queue q3. Scheduling ensures end-to-end deterministic delivery by configuring appropriate GCL entries across all switches, where each hop reserves suitable time slots for critical traffic packets.

The period of a flow in TSN is defined as the fixed time interval between the generation of consecutive frames of the same flow, determined by the source and typically dictated by the application's timing requirements. In this paper, the Cycle is defined as the least common multiple of all flow periods, serving as the global scheduling period in TSN. Within each cycle, the behavior of all flows remains consistent.

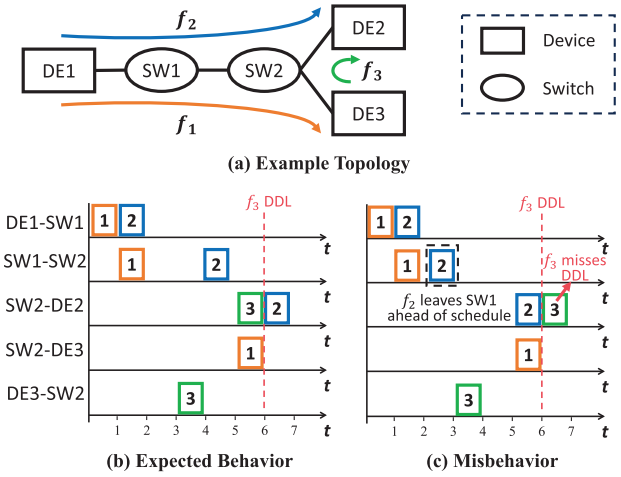


Fig. 3. Forwarding misbehavior and chain of errors.

B. TSN Scheduling and Forwarding Misbehavior

Time-Sensitive Networking (TSN) enables deterministic data transmission, which is achieved by precise time synchronization among network devices and dedicated bandwidth reservations for critical traffic. It relies on a well-designed global schedule to reserve dedicated bandwidth. Figure 3(a) and (b) demonstrate a simple network topology and the predetermined TSN schedule. As can be seen, the schedule dictates the exact transmission times of data packets on each network link. For example, the packet in flow f_1 is supposed to leave the output port of the device DE1, switch SW1, and switch SW2 at timestamps 0, 1, and 5, respectively. Following the schedule, the TSN switches configure their output port's gate control lists to regulate when each packet can depart from their output ports, thus ensuring the smooth operation of a TSN network. Nevertheless, this well-designed schedule also results in strong interdependence among critical flows, introducing unique challenges.

Similar to traditional Ethernet switches, TSN network devices may be susceptible to hardware logic flaws and software errors, resulting in unexpected outcomes. We define TSN forwarding misbehavior as instances in which the actual transmission times of the data packets deviate from the planned schedule. These deviations may manifest as packets that leave a switch earlier or later than scheduled or, in more severe cases, packets being lost entirely. For instance, as depicted in Figure 3(c), the packet of flow f_2 leaves switch SW1 ahead of schedule while the packet of flow f_3 leaves switch SW2 with a delay. Such deviations can lead to end-to-end transmission delays that exceed the application requirements, causing errors in critical industrial processes. In our beta test run, we identified the fault type of common misbehaviors as Incomplete Protocol Support, Network Misconfiguration, and Hardware Logic Flaws (refer to §V-A).

C. Limitation of Current Practice

To study the effectiveness of existing Ethernet diagnostic systems, we deploy a testbed with TSN switches and FPGA-based evaluation toolkits (setup detailed in §V). As mentioned

in §I, *tracing* methods are not applicable in our scenario. Therefore, we implement the *probing* [7], *passport* [14], and *traditional postcard* [17] methods to tackle the TSN fault localization problem. The overall results of our case study are demonstrated in Table I. As seen, neither of the existing methods could successfully localize and classify TSN misbehavior. We dig into the underlying reasons and find that the challenges are three-fold:

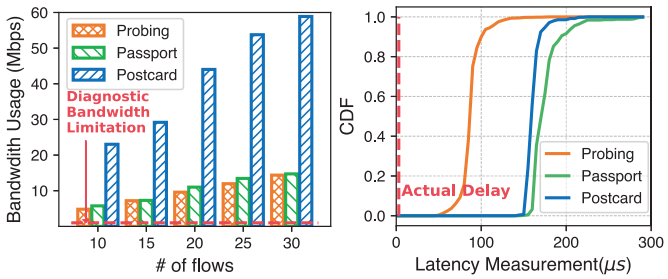
C1: Chain of Errors: In TSN, packets are carefully scheduled for transmission over shared network links, creating complex interdependence among data streams. This time-division multiplexing setup ensures precise timing and deterministic transmission for critical industrial traffic. However, it also implies that a delay or misalignment for one packet can disrupt the transmission of subsequent streams, leading to a chain of errors from the original malfunctioning device. As a result, when the end-to-end delay of a data stream exceeds scheduled deadline, numerous switches in the network may already be experiencing forwarding misbehavior, making it difficult to pinpoint the fault location.

Figure 3 demonstrates a typical chain-of-errors problem we encountered. As the original fault location of misbehavior, the switch SW1 in Figure 3(a) encounters an internal hardware logic malfunction. This increases the packet departure time of flow f_2. As a result, the packet from flow f_2 arrives at switch SW2 earlier than the schedule, subsequently delaying the transmission of flow f_3. From an operational point of view, only flow f_3 appears to miss its deadline, while flow f_1 and f_2's end-to-end latency is acceptable. However, if the network operators or diagnostic tools only inspect the switches and devices along the path of flow f_3, they will never find the fault location of the issue, i.e., switch SW1.

The propagation of errors in TSN differs significantly from conventional Ethernet, specifically in two aspects: (1) Upstream Dependency Propagation: Errors propagate from a faulty node along the flow path. Downstream nodes, even when functioning normally, may exhibit anomalies due to deviations in upstream inputs. For example, in Figure 3, a hardware error in SW1 causes f_2 to arrive prematurely, rendering SW2 unable to process f_2 effectively. (2) Time-Dependent Cascade Propagation: Within a single switch, timing deviations in any flow directly impact subsequent flows by altering the occupancy of their designated time slots, thereby affecting other flows. As illustrated in Figure 3, f_2 preempts f_3's time slot in SW2, resulting in f_3 timing out. In contrast, traditional Ethernet employs best-effort forwarding. Forwarding failures are mitigated through packet loss or buffer queuing mechanisms, avoiding systemic cascading effects.

According to our investigation, all existing tools lack the granularity necessary to identify the forwarding misbehavior at each switch. Therefore, they are unable to effectively trace the error back to its origin.

C2: Limited Available Bandwidth: Typical industrial production networks comprise various types of business traffic, including periodic control, surveillance video, and raw sensor data. Most of the network's total bandwidth is allocated to these high-priority traffic, leaving limited bandwidth for traffic monitoring and diagnostics. According to Li et al. [23] and



(a) Limited available bandwidth. (b) Inaccurate time measurement.

Fig. 4. Limitation of current practice.

our field study in typical manufacturing factories, in a typical 100Mbps industrial network, there are only about 1Mbps free bandwidths left for network analysis. However, traditional Ethernet diagnostic methods, designed for data centers with abundant bandwidth, rely on continuous generation, collection, and storage of extensive historical telemetry data. Therefore, these methods may not be effective under the constrained bandwidth available in TSN.

We evaluate the existing methods in our testbed for traffic analysis. Specifically, we remove background traffic (video, sensor data, etc.), leaving only periodic control traffic in the network. Then, we apply each of these methods to continuously collect analytic information of these critical signals, and record their bandwidth utilization. In theory, *probing* involves emulating the business traffic with test packets, thereby consuming a bandwidth comparable to the periodic control traffic. On the other hand, *passport* and *postcard* methods are collecting the hop-by-hop information. Their bandwidth consumption is related to the number of hops each data stream traverses. As shown in Figure 4a, the bandwidth consumption of these methods increases linearly with the number of critical flows. When the number of flows exceeds 10, all of their bandwidth consumption exceeds the 1Mbps limit, leading to potential traffic congestion and loss of analytic information. Subsequent experiments demonstrate that these methods fail to 1Mbps limit, even when detecting only abnormal flows.

C3: Inaccurate Time Measurement: As mentioned earlier, critical data frames in TSN are required to be transmitted with nanosecond-level precision, adhering to the global synchronized clock and the predetermined schedule. Therefore, to effectively identify potential forwarding misbehavior in TSN, it is essential to acquire high-precision packet-level transmission time at each hop in the network. Existing network diagnostic methods rely on software-based time synchronization, such as the NTP protocol, because its millisecond-level accuracy is sufficient for conventional network monitoring and diagnosis. However, TSN fault diagnosis demands nanosecond-level precision, which can only be achieved through hardware-based synchronization using the PTP protocol. Existing methods either measure only end-to-end or Round-Trip Time (RTT) delays, lacking the necessary hop-level granularity (e.g., tracing, probing), or depend on software-based synchronization (e.g., passport, traditional postcard), which is inadequate for accurately capturing TSN's strict timing requirements. Furthermore, since each critical

data stream conforms to the strict timetable. Our experiments show that embedding 16B timestamp into packets can delay their transmission time about $1us$, leading to discrepancies between observed results and actual performance by packet violating schedule. Therefore, *passport* methods are inapplicable for TSN traffic analysis.

We setup the testbed with a switch connecting to two evaluation devices and measure the one hop delay from a device to the switch. To be specific, *probing* collects the RTT along the path of the source device, switch, sink device and then divide it by 4; others rely on the Network Time Protocol (NTP) for clock synchronization and send the receiving time back from the switch to the source device with embedded metadata (*passport*) or customized packet (*traditional postcard*). Theoretically, the one hop delay primarily consists of packet propagation delay on the network cable and the PHY (physical layer) processing delay, which typically does not exceed $1\mu s$. As illustrated in Figure 4b, itprobing method exhibits an average deviation about $92\mu s$, while *passport* and *traditional postcard* methods show deviations over $160\mu s$. The same issue occurs with the transmission of probe packets. Our experiments indicate that using a standard Linux network stack can result in a transmission time error of up to 15 μs. Such a level of inaccuracy in time measurement is problematic for TSN traffic analysis.

Lessons Learned. To advance effective fault localization in TSN, our investigation reveals three aspects could be enhanced:

(i) According to C1, we can design a new network analysis paradigm that is able to pinpoint the root fault location of the complex chain of errors, thus facilitating effective and responsive repair of critical system failures.

(ii) Motivated by C2, the fault localization algorithm should be adaptive to the bandwidth limitation of particular environments and ensure that no vital information is lost in the process.

(iii) Following C3, we require the diagnostic system to have nanosecond-level time measurement accuracy, which includes precisely recording the entry and exit times of packets as well as accurately sending probing packets.

D. TSNCard: System Goals

Goal 1: Full Coverage on Time-related Misbehavior. TSNCard should ensure comprehensive data collection and reliably identify the fault location and type of time-related periodic network issues in all scenarios. This is essential for the effectiveness of any TSN fault localization system (§V-B).

Goal 2: Light Weight. TSNCard should maintain low bandwidth consumption, low data collection time, and avoid disrupting existing critical traffic flows. This allows the diagnostic system to be seamlessly deployed under the constrained industrial environments (§V-C).

III. System Overview

TSNCard systematically tackles the aforementioned challenges to achieve reliable fault localization and classification in TSN. As depicted in Figure 5, it comprises the protocol layer for network telemetry and data collection, the algorithm layer for intelligent data analysis and fault localization, and the hardware layer for high-precision and seamless postcard generation.

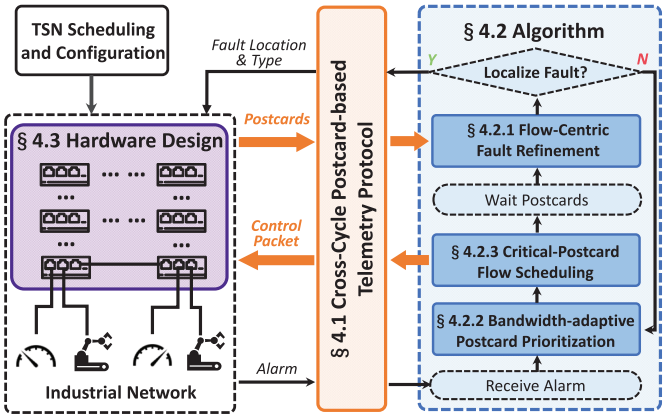
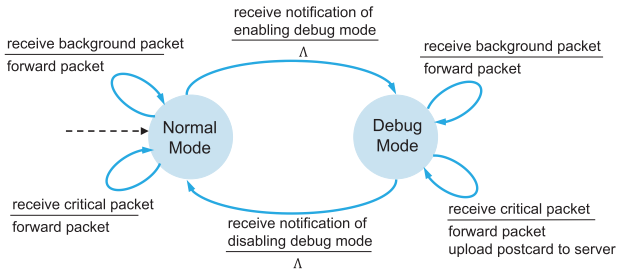


Fig. 5. System architecture of TSNCard.

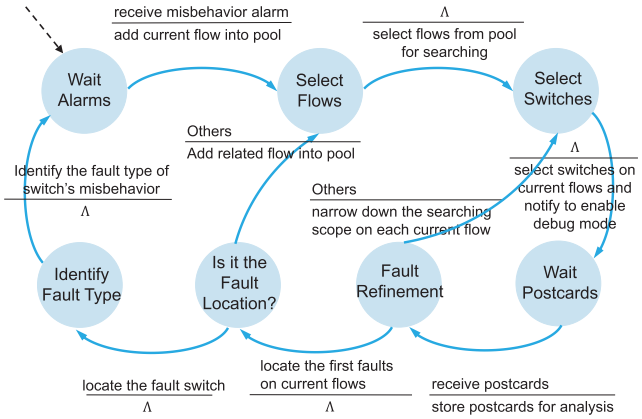
Key Functional Modules.

- *Cross-Cycle Postcard-Based Telemetry* focuses on efficiently collecting network diagnostic information. It leverages TSN's periodic nature to gather and analyze data over multiple cycles, reducing bandwidth consumption while maintaining diagnostic information coverage.
- *Flow-Centric Fault Refinement* identifies and traces the misbehavior within network flows. It utilizes the collected postcards to reconstruct error chains and pinpoint the fault location.
- *Bandwidth-Adaptive Postcard Prioritization* decides whether each switch should generate postcards based on the available network bandwidth. It optimizes the balance between localization efficiency and network load, ensuring efficient and timely fault localization.
- *Critical-Postcard Flow Scheduling* ensures low latency and determinism in postcard transmission. It schedules postcards as critical traffic without affecting the business flow, thereby providing assurance for the convergence of the Fault Refinement algorithm. Then switch creates and sends postcards with high-precision timestamps without interfering with the transmission of critical traffic.

Figure 5 illustrates the overall workflow of the TSNCard. When a critical flow anomaly occurs, end device sends an alarm to the analytic server. The analytic server then utilizes *Bandwidth-Adaptive Postcard Prioritization* algorithm to select switches for postcard collection. Subsequently, *Postcard-Critical Flow Scheduling* allocates dedicated bandwidth for postcards and distributes configurations to the switches. After the switches gather the raw data required for analysis through both hardware and software and upload postcards to the server. The server then employs a *Flow-Centric Fault Refinement* algorithm to pinpoint the fault location and recognize the type. If fails, the server instructs the *Bandwidth-Adaptive Postcard Prioritization* to analyze the next flows or nodes and repeats the process until success.



(a) Automaton on the switch.



(b) Automaton on the analytic server.

Fig. 6. Cross-cycle postcard-based telemetry protocol.

IV. Design and Implementation

A. Protocol: Postcard-Based Telemetry

As explained in §II-C, traditional Ethernet diagnostic tools lack the ability to localize and classify fault in the TSN network. In TSNCard, we propose a Postcard-based Cross-Cycle Telemetry Protocol dedicated to localizing broken root switches or end devices in the TSN network. The automatons in Figure 6 depict the design of this protocol. The first, as shown in Figure 6a, governs how the switches collect and upload postcards. If the switch is notified of enabling debug mode, it timestamps all critical packets' arrival and departure, i.e., *rx* and *tx*. But if not, the switch only conducts pure packet forwarding. The second one, as shown in Figure 6b, shows how to identify the fault location and type on the analysis server. When the server receives an alarm, it selects some flows for analysis from the flow pool and notifies the relevant switches to enter debug mode to collect postcards. Using the Fault Refinement algorithm, we find the first error within the selected flows in an attempt to identify the fault location in TSN; otherwise, we re-select related flows and repeat the steps above. Finally, on top of the fault switch, TSNCard will identify its fault type.

In Cross-Cycle Postcard-based telemetry Protocol, we introduce two modules, i.e., *Cross-cycle data fusion* and *Active postcard-based telemetry*, to overcome the weakness of traditional Ethernet diagnostic protocols.

1) Cross-Cycle Data Fusion: Distinguished from traditional Ethernet, TSN usually serves applications whose traffic has periodic characteristics and requires strict low latency. As a result, if some device is misconfigured or broken down, the misbehavior in this traffic also appears periodically. This inspires us to propose cross-cycle data fusion. Specifically, it is unnecessary to snapshot the whole TSN network. The key characteristics of the time-sensitive flow, i.e., the arrival and departure times at each hop, are consistent in all cycles after misbehavior occurs. As shown in Figure 7, we can selectively inspect the timestamps of the most relevant traffic flows in current cycle and examine other flows in subsequent cycles. By leveraging cross-cycle data fusion techniques, we can significantly reduce the bandwidth consumption required for network telemetry. Furthermore, this approach makes TSNCard efficient and adaptable to varying bandwidth limitations (refer to §IV-B2).

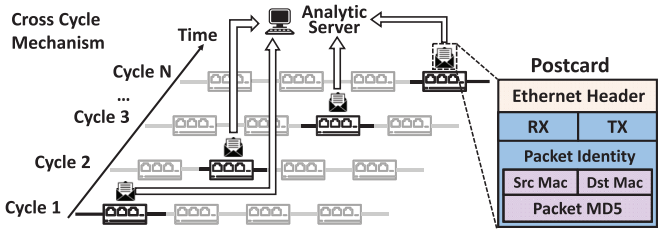


Fig. 7. Illustration of the cross-cycle mechanism.

The proposed cross-cycle data fusion is intuitively effective. In §V-B, we will experimentally demonstrate that we can pinpoint the fault location and type with this method in all cases. In addition, we will also discuss its limitations in §VII.

2) Active Postcard-Based Telemetry: We design the TSN diagnostic protocol based on active postcard telemetry [17]. Benefiting from postcard telemetry, there is no need to modify the packet structure of critical flows, which means TSNCard is compatible with existing end-devices in TSN networks in factories. If the TSNCard switch is enabled in debug mode, the switch will collect data for fault analysis and transmit it to the analysis server. As illustrated in Figure 7, these data include nanosecond-level arrival (*rx*) and departure (*tx*) timestamps actively recorded by the hardware module (refer to §IV-C), as well as the packet identity filtered on the software side (refer to §IV-B). The packet identity consists of source and destination MAC addresses and the MD5 value of the raw packet, which are invariant during packet propagation. The analytic server can utilize these identities to aggregate packets within the same flow.

B. Algorithm: Fault Refinement and Bandwidth Adaptation

Due to the existence of *chain of error*, there may not be direct connections between the breakdown switch and misbehaved flows. Nevertheless, according to C1 in §II-C, the misbehavior is propagated along the flow path and leaves traces. This inspires us to propose *Flow-Centric Fault Refinement* module. It exploits misbehavior traces, which are collected by postcards, to recover the complete error chain and localize the breakdown switch. Furthermore, in order to make TSNCard lightweight enough for the actual product line in factories, we introduce *Bayesian Guided Postcard*

TABLE II
SEVEN CATEGORIES OF MISBEHAVIOR

	Criterion	Misbehavior Category
Ingress-Related Misbehavior	$rx + T < rx^s$	Early Ingress
	$rx > rx^s + T$	Late Ingress
	$rx - rx^s > P$	Periods Late Ingress
Egress-Related Misbehavior	$tx + T < tx^s$	Early Egress
	$tx > tx^s + T$	Late Egress
	$tx - tx^s > P$	Periods Late Egress
Loss Misbehavior	No Postcard	Packet Loss

Prioritization. It automatically adjusts the postcard collection strategy according to the bandwidth limitation, balancing the bandwidth consumption and the fault localization delay.

1) Flow-Centric Fault Refinement: The misbehavior trace refers to the inconsistency of the actual packet arrival/departure time with the schedule. As depicted in Table II, we classify misbehavior into seven categories. The second column presents the criteria for the identification of misbehavior. The superscript s denotes the scheduled timestamp. P is the period of this flow while T is the tolerable deviation between the actual timestamp and the scheduled one. If the server fails to collect some postcard, it is a Packet Loss misbehavior. In order to avoid postcard loss from the switch to the analytic server, TSNCard conducts collection in three consecutive cycles. The Packet Loss misbehavior is ultimately recognized when, and only when none of these three are collected. Notably, we do not need to store the rx^s and tx^s for all packets; these can be computed based on their position within a cycle and the cycle number.

Flow-centric fault refinement adopts two complementary phases to tracking the misbehavior comprehensively. As described in §II-C, TSN propagation has the characteristics of Upstream Dependency Propagation and Time-Dependent Cascade Propagation, for which we design Intra- and Inter-flow refinement algorithms.

In TSN, each packet is allocated a dedicated time slot. So, for diagnostic purposes, we can effectively treat a multi-packet flow as multiple single-packet flows, each with its own timing requirements. This approach allows for more granular analysis of timing violations within the same logical flow. At the end devices, we may observe anomalies in numerous flows, hence, we will analyze the causes of errors for each flow individually. We iteratively apply Intra-flow and nter-flow refinement until the set of flows requiring analysis becomes empty.

Intra-flow refinement phase. To trace the misbehavior upstream along the flow, it is crucial to determine whether any misbehavior originates from upstream. This can be assessed by evaluating whether the flow arrives at the switch on time. If the flow arrives as expected, we can infer that the upstream switch is functioning normally for this specific flow. However, if not, we can only confirm the existence of upstream misbehavior, but can't conclusively determine whether the current switch is functioning properly.

In this phase, we will progressively inspect the switches along the specified flow's upstream direction:

- If the flow arrives as expected at the switch, it indicates that the misbehavior is not caused by upstream elements. Intra-flow refinement phase for this flow is then terminated, and upstream switches need not be further examined.
- If the flow does not meet expectations upon reaching the switch, it suggests an issue with the upstream switch, although the current switch may still be faulty. The issue will then be traced further upstream. Intra-flow refinement phase for this flow continues.

The analyzed switches (SW*) will be marked for further Inter-flow refinement checks, as their status (faulty or not) cannot yet be determined.

Inter-flow refinement phase. The task of the Inter-flow refinement phase is to determine whether the misbehavior of the marked switch (SW*) originates from misalignment in the arrival of flows, a switch failure, or both.

First, we assess whether all flows arriving at the switch reach it in a timely manner, i.e., whether there is any Ingress-Related Misbehavior, as outlined in Table II. Flows exhibiting misbehavior are marked and added to the set requiring Intra-flow refinement.

Next, we determine whether the switch is faulty. By getting the arrival times of all upstream packets, we can infer the expected forwarding behavior of the switch based on scheduling results and compare it with its actual behavior. If the two align, the switch is deemed fault-free; otherwise, faulty.

For example, in Figure 3, we analyze the forwarding behavior of SW2. Obviously, f_1 and f_2 did not arrive as expected and need to be included in the subsequent Intra-flow refinement set. Given the arrival times of $f_1(t = 1 \sim 2)$, $f_2(t = 2 \sim 3)$, and $f_3(t = 3 \sim 4)$, we deduce that SW2 should forward f_2 at $t = 5 \sim 6$ and f_3 at $t = 6 \sim 7$ on link SW2-DE2. Despite a deviation from the expected forwarding schedule, SW2 itself operates without fault.

2) Bandwidth-Adaptive Postcard Prioritization: In the intra-flow refinement phase, if all postcards of current flows are collected in one cycle, the bandwidth constraint is surpassed. On the other hand, if taking advantage of periodic characteristics and collecting one postcard each cycle, the available bandwidth is underutilized and the fault localization delay will be high. As mentioned before, misbehavior is transitive, so the misbehaved switches on a specific flow are continuous. This implies that it is feasible to select several switches to check and narrow down the scope of the first misbehaved switch iteratively. A naive greedy strategy is to prioritize switches with more flows passing through them. This strategy is somehow reasonable because more pass-through flows mean a greater likelihood of being impacted, but it fails to take the topology and flow characteristics into account.

TSNCard introduces *bandwidth-adaptive postcard prioritization* to guide the switch selection. We refer to the Naive Bayes algorithm and design a heuristic selection strategy. As shown in Alg. 1, the K switches with the largest $P(y|x, f)$ values are selected (line 4-5), where $P(y|x, f)$ denotes the probabilities of there is misbehavior on switch y in flow f under the condition switch x in flow f misbehaves. The Alg. 2 shows how to calculate $P(y|x, f)$. According to Bayes'

Algorithm 1 Bandwidth-Adaptive Postcard Prioritization

input : Current flow f, misbehaved switch x, K
output: K switches in f for postcard collection
1 $path \leftarrow$ Get all switches from the source of f to switch x;
2 **foreach** *switch y in path* **do**
3 Calculate $P(y|x,f)$
4 $sorted\_path \leftarrow$ sort $path$ in descending order according to $P(y|x,f)$;
5 $K\_Switches \leftarrow$ the first K switches in $sorted\_path$;
6 **return** $K\_Switches$

Algorithm 2 Calaculate $P(y \mid x, f)$

input : Current flow f, switch y, x
output: $P(y|x,f)$
1 Obtain $P(y,f)$ according to statistics;
2 $P(x|y,f) \leftarrow 1$;
3 **foreach** *link (s,t) in path from y to x in flow f* **do**
4 $P(s|t) \leftarrow 1 - 0.5(1 - e^{-0.5D_{s,t}})$;
5 $P(x|y,f) \leftarrow P(x|y,f) \cdot P(s|t)$;
6 $P(y|x,f) \leftarrow P(x|y,f) \cdot P(y|f)$;
7 **return** $P(y|x,f)$

theorem, there is

$$P(y|x,f) = \frac{P(x|y,f) \cdot P(y,f)}{P(x,f)}, \tag{1}$$

where $P(*,f)$ denotes the switch $*$ in flow f misbehaves. Since $P(x,f)$ is constant, we have (line 6 in Alg. 2)

$$P(y|x,f) \propto P(x|y,f) \cdot P(y,f). \tag{2}$$

In Eq.(2), $P(y,f)$ is the prior probability while $P(x|y,f)$ is the likelihood. In our implementation, we exploit a network simulator to generate a mass of network operation data and obtain $P(y,f)$ according to statistics (line 1 in Alg. 2). $P(x|y,f)$ denotes the probability that the misbehavior on switch y is transmitted to switch x on flow f. As x and y are both in the flow f, we can decompose $P(x|y,f)$ as follows (line 5 in Alg. 2)

$$P(x|y) = P(x_1|y)P(x_2|x_1) \cdots P(x_m|x_{m-1})P(x|x_m), \tag{3}$$

where link $(y,x_1),(x_t,x_{t+1})$and (x_m,x) all belong to flow f.[1] For each $P(x_{t+1}|x_t)$, we notice that it has the following properties:

1) $P(x_{t+1}|x_t)$ is **inversely proportional** to D, where D is the number of flows that pass through link (x_t,x_{t+1}).
2) $P(x_{t+1}|x_t) \in [0,1]$. If $D=1$, $P(x_{t+1}|x_t)=1$. If $D \to +\infty$, $P(x_{t+1}|x_t)$ should be relatively small because it is very possible that the misbehavior could be transmitted to another flow.

In our implementation, we set $P(x_{t+1}|x_t) = 1 - 0.5(1 - e^{-0.5D})$) empirically (line 4 in Alg. 2). As a result, the entire calculation progress of $P(y|x,f)$ are shown in Alg. 2.

[1]For simplicity, the symbol f has been omitted.

3) Critical-Postcard Flow Scheduling: In practice, treating postcards as non-critical traffic led to a high probability of packet loss, especially when network bandwidth was low and saturated with audio and video traffic. Severe packet loss disrupted the periodicity of cross-cycle postcards, causing the entire algorithm to fail. Fortunately, Bandwidth-adaptive Postcard Prioritization reduced postcard bandwidth to within 1Mbps, enabling postcards to be treated as critical traffic and scheduled using fault-free links in most cases.

The overall algorithm flow is as follows: postcard traffic is modeled as critical traffic and jointly scheduled with other service traffic. If joint scheduling fails, the original scheduling of service traffic remains unchanged, and postcard traffic is scheduled incrementally, with unsuccessfully scheduled service traffic sent as ordinary traffic.

Network and Traffic Notation: We model the scheduling problem as follows. In this paper, we use a directed graph $G(V,L)$ to represent the network topology. Here, V denotes the set of network nodes, and $L \subseteq V \times V$ represents the set of data links. All data links are duplex, meaning that if v_i and v_j are connected, both $\langle v_i, v_j \rangle$ and $\langle v_j, v_i \rangle$ belong to L.

Each unidirectional link $\langle v_i, v_j \rangle$ has two attributes: transmission delay $\langle v_i, v_j \rangle.d$ and time slot $\langle v_i, v_j \rangle.ti$. Here, the time slot ti represents the minimum time unit for scheduling. For each critical data flow s_i, this paper defines five attributes: $(s_i.path, s_i.md, s_i.pri, s_i.len, s_i.T)$ where $s_i.path$ represents the links traversed by data flow s_i, $s_i.md$ represents the maximum allowed latency of the data flow, $s_i.pri$ represents the priority of data flow s_i, $s_i.len$ represents the byte length of each data frame in data flow s_i, and $s_i.T$ represents the transmission period of data flow s_i. This paper represents the set of data flows to be scheduled as $S = \{s_0, s_1, \ldots, s_{n-1}\}$.

The data frames contained in data flow s_i on link $\langle v_a, v_b \rangle$ are represented by $f^{s_i,\langle v_a,v_b \rangle}$, and the set of data frames transmitted by data flow s_i on link $\langle v_a, v_b \rangle$ is represented by $F_{s_i,\langle v_a,v_b \rangle}$. For each data frame $f_j^{s_i,\langle v_a,v_b \rangle}$, this paper defines two attributes: $(f_j^{s_i,\langle v_a,v_b \rangle}.start, f_j^{s_i,\langle v_a,v_b \rangle}.lat)$ where $start$ represents the transmission time of the data frame on link relative to the beginning of each period, and lat represents the transmission delay of the data frame from the port to link.

Formulation:

- Time Constraints: The transmission time of data frames cannot be negative and must be completed within one period:

$$\begin{aligned}&\forall s_i \in S, \forall \langle v_a, v_b \rangle \in s_i.path, \forall f_j^{s_i,\langle v_a,v_b \rangle} \in F^{s_i,\langle v_a,v_b \rangle} \\ &\quad \to (f_j^{s_i,\langle v_a,v_b \rangle}.start \geq 0) \\ &\quad\quad \wedge (f_j^{s_i,\langle v_a,v_b \rangle}.start + f_j^{s_i,\langle v_a,v_b \rangle}.lat \leq s_i.T)\end{aligned}$$

Data frames of the same data flow should be transmitted in order on the link, meaning that the next data frame can only be sent after the previous one has been completely transmitted:

$$\begin{aligned}&\forall s_i \in S, \forall \langle v_a, v_b \rangle \in s_i.path, \forall f_j^{s_i,\langle v_a,v_b \rangle} \in F_{s_i,\langle v_a,v_b \rangle} \\ &\quad \to f_j^{s_i,\langle v_a,v_b \rangle}.start \geq f_{j-1}^{s_i,\langle v_a,v_b \rangle}.start \\ &\quad\quad + f_{j-1}^{s_i,\langle v_a,v_b \rangle}.lat\end{aligned}$$

The latency of a data flow from the source node to the destination node cannot exceed the maximum allowed latency of that data flow:

$$\forall s_i \in S \rightarrow f_{last_f}^{s_i,s_i.path[last_p]}.start + f_{last_f}^{s_i,s_i.path[last_p]}.lat + [s_i.path[last_p]].d - f_0^{s_i,s_i.path}.start \leq s_i.md$$

where $last_p$ represents the last link, and $last_f$ represents the last data frame of the data flow.

- Frame Overlap Constraints: Each link can transmit only one data frame at a time, and the scheduling time intervals of any two data frames on the same link must not overlap:

$$\begin{aligned}&\forall s_i, s_j \in S, \forall \langle v_a, v_b\rangle \in s_i.path \cap s_j.path,\\ &\forall F_{s_i,\langle v_a,v_b\rangle}, F_{s_j,\langle v_a,v_b\rangle}, i \neq j,\\ &\forall f^k_{s_i,\langle v_a,v_b\rangle} \in F_{s_i,\langle v_a,v_b\rangle}, f^l_{s_j,\langle v_a,v_b\rangle} \in F_{s_j,\langle v_a,v_b\rangle}\\ &\rightarrow (f^k_{s_i,\langle v_a,v_b\rangle}.start \geq f^l_{s_j,\langle v_a,v_b\rangle}.start + f^l_{s_j,\langle v_a,v_b\rangle}.lat)\\ &\vee (f^l_{s_j,\langle v_a,v_b\rangle}.start \geq f^k_{s_i,\langle v_a,v_b\rangle}.start + f^k_{s_i,\langle v_a,v_b\rangle}.lat)\end{aligned}$$

- Adjacent Link Constraints: The difference in the starting transmission times of a data frame on two adjacent links must be at least equal to the time required for the complete transmission of one data frame on a single link:

$$\begin{aligned}&\forall s_i \in S, \forall \langle v_a, v_b\rangle, \langle v_b, v_c\rangle \in s_i.path,\\ &\forall f^j_{s_i,\langle v_a,v_b\rangle} \in F_{s_i,\langle v_a,v_b\rangle}, \forall f_j^{s_i,\langle v_b,v_c\rangle} \in F_{s_i,\langle v_b,v_c\rangle}\\ &\rightarrow f_j^{s_i,\langle v_b,v_c\rangle}.start - f_j^{s_i,\langle v_a,v_b\rangle}.start\\ &\geq \langle v_a, v_b\rangle.d + f_j^{s_i,\langle v_a,v_b\rangle}.lat\end{aligned}$$

In our implementation, we use the Z3-solver [24] for Satisfiability Modulo Theory (SMT) [25] problems and employ the incremental scheduling algorithm, DeepScheduler [26]. Since scheduling is NP-hard and takes at least several seconds—unacceptable in real operations—we precompute schedules for all possible fault scenarios, then upload the corresponding GCL for use, taking advantage of the typically limited scale of TSN networks.

Taking a topology with 20 switches fully connected as an example, we assume that the number of failed switches in the network is less than or equal to two, so that there are a total of 210 different error scenarios, each of which requires only incremental scheduling of postcard traffic, and DeepScheduler gets the solution in less than 2s, which summarizes in about 7 minutes. This way pre-scheduling adds only a small burden to the network deployment.

C. Hardware: High-Precision In-Switch Postcard Generation

1) Switch Hardware: As mentioned in §II-C, to effectively identify forwarding misbehavior in TSN, it is essential to capture the high-precision data transmission time at the nanosecond level. Consequently, TSNCard should be time synchronized and generate the postcards with accurate packet transmission time to support the upper layer protocol (§IV-A) and algorithm (§IV-B) design.

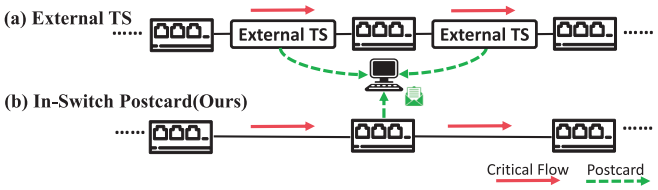


Fig. 8. Illustration of external timestamper and TSNCard's in-switch postcard hardware design.

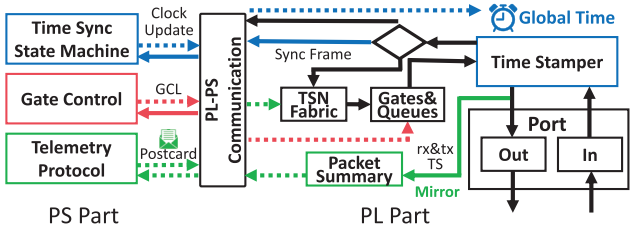


Fig. 9. Architecture of TSN switch hardware design.

As demonstrated in Figure 8(a), an intuitive approach for timestamping is to implement external timestamper devices and insert them into network cables. They forward the data directly between input/output ports and utilize the 802.1AS SYNC messages passing for time synchronization. When critical flows pass through a timestamper, they can record the transmission time of the packet and generate corresponding postcards. However, this method has two downsides:

(1) The PHY (Physical Layer) processing on the timestamper introduces additional packet transmission latency.
(2) It necessitates extra devices attached to each network cable, leading to higher deployment costs.

Accordingly, in TSNCard, we choose the in-switch postcard design, integrating the hardware implementation into existing TSN switch's logic as a plugin as shown in Figure 8(b). In this way, TSNCard can access the local high-precision clock of each switch, and will not introduce additional PHY processing delay. Meanwhile, the design should be more careful to avoid interfering critical packet's transmission. Specifically, TSNCard leverages a NetFPGA-based TSN switch architecture [27], [28] and the Xilinx Zynq platform [20]. As shown in Figure 9, it consists of a hardware PL (Programmable Logic) part and a software PS (Processing System) part. The PL records the transmission and packet identity and uploads them to PS, while the PS packages and sends postcards to the analytic server for necessary packets.

The core design of TSNCard's hardware implementation are as follows:

Time Stamper Modules. The time stamper modules are supposed to measure the precise timing a critical data frame arrives at (*rx*) and departure from (*tx*) the TSN switch. The *rx* time stamper module is positioned right after the MAC (Media Access Control) and before its buffering FIFO. It records the 64bit *rx* nanosecond-level timestamp when the first byte of

each packet is decoded from MAC. Similarly, the *tx* time stamper module continuously monitors the mirrored output data stream and records the timestamp when the first byte of a packet is successfully sent.

Frame Mirroring Mechanism. The frame mirroring mechanism is designed to capture the critical traffic information without hindering its transmission. It is placed after the transmission selection module (containing the TSN cyclical control gates) of each output port. The module mirrors the AXI-stream signal into two paths: the main path directly connects to the port's PHY, while the other path copies the data byte-by-byte into a packet-mode buffer FIFO (First-In-First-Out queue), strictly following the state machine transition process of the main signal. In addition, the module incorporates a FIFO time stamp *rx* and a *tx*, each with 64 bit width. When the first byte of a mirrored data frame is stored in the buffer, the accompanying *rx* signal is pushed into the *rx* FIFO, and the current timestamp is pushed into the *tx* FIFO. After the whole packet is stored in the buffer FIFO, the packet content, *rx* and *tx* timestamps, will pop out from the FIFO and be ready for transmission to PS.

Bandwidth Optimization Techniques. Ideally, we can simply pass the packet content and corresponding timestamps to PS, who can further generate and send a postcard following our protocol design (§IV-A). However, due to hardware constraints, the bandwidth for PL-PS communication is relatively limited, leading to potential packet loss. To mitigate this issue, we introduce two key techniques in TSNCard. First, we implement a packet summary module, transmitting only essential information such as MD5 hash and packet identity (as mentioned in §IV-A). This reduces the PS-PL bandwidth requirement by 95% (assuming that the original packet size is 1500 bytes). Second, instead of transmitting the *rx* timestamp from the corresponding time stamper module, we embed it as an 8 bit side signal (i.e., tuser signal in AXI-Stream protocol [29]) with the packet data stream. Thus, no additional packet identity information is required to match *rx* and *tx* timestamps, further reducing the bandwidth requirement by about 50%.

Following the trend of in-network computing, major network device vendors nowadays are actively embedding their switches with additional programmable logic. For example, Kontron's PCIE-0400-TSN NIC contains an Altera Cyclone v5 FPGA [30], Intel Tofino switch contains a programmable ASIC chip supporting P4 language [31]. These designs enable the end users to get more control over network switch's operation logic. Given the modular design and validated performance of TSNCard, integrating it into these commercial TSN switches would require minimal redevelopment effort.

2) TSNPerf Hardware: As metioned in §II-C, to accurately identify the TSN switch fault type, we can utilize probe packets to observe the state of the switch. So we need a tool capable of integrating network time synchronization and sending specific probing packets with nanosecond precision. This will enable us to test core TSN functionalities such as time synchronization accuracy, gate control capability, and gate control precision.

In TSNCard, we have designed an auxiliary tool called TSNPerf to dig deep into switch failures. TSNPerf is also

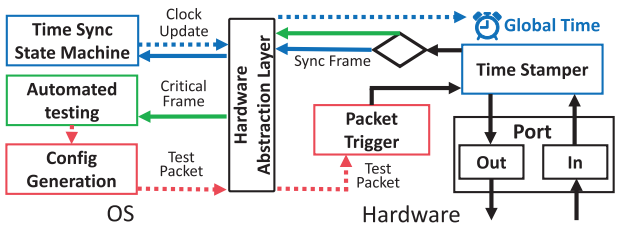


Fig. 10. Architecture of TSNPerf hardware design.

FPGA-based, consisting of PS and PL parts. To facilitate the portability of its architecture to third-party TSN devices, we have abstracted its structure into a OS layer and a Hardware layer, corresponding to the PS and PL, respectively, as illustrated in Figure 10. The key component in the OS part is the automated testing module, which generates subsequent test configurations based on the malfunctioned switch. This module allows users to configure packet parameters, like MAC addresses and ethertype, in the OS. The key component in the Hardware part is Packet Trigger Module, which is designed to control the precise timing of packet transmission in the TSNPerf. When the generated packets arrive at the output port, the Packet Trigger Module will hold them until the scheduled transmission time. This release signal is controlled by a global synchronization clock with nanosecond precision, ensuring that the first byte of packets is always transmitted at a pre-scheduled time.

We consider TSNPerf to be a highly practical tool. With TSNPerf, we can not only identify switch's fault type but also leverage its precise packet generation capabilities to simulate real network environments or test the performance of TSN switches. We have open-sourced the entire TSNPerf code, enabling easy porting to any device equipped with hardware timestamp and packet trigger capabilities.

V. Evaluation

A. Experimental Methodology

Testbed setup. We integrate TSNCard's hardware design into the a FPGA-based TSN switch on top of the Xilinx Zynq platform [20], and run the analytic algorithm on a server equipped with a 36-core Intel Core i9-10980XE@3.00GHz and 128GB RAM. As depicted in Figure 11, we set up three testbeds with different topologies, i.e., Ring6, A380, and CEV (6, 9, and 8 TSN switches were deployed respectively i.e., the circles in Figure 11, while the remaining squares denote the TSNPerf). Among them, A380 is a simplified topology of the control network used on Airbus A380 [32], while CEV, short for Crew Exploration Vehicle, is a simplified version of the Orion spacecraft's control network [33]. Ring6 is also popular in industrial networking settings [34]. On each testbed, we stochastically generate several critical flows with a period of $1ms$. Each flow's deadline is between 50 and $200\mu s$.

Simulation setup. In addition to physical testbeds, we also conduct comprehensive simulation evaluations with large-scale network topologies based on OMNeT++ INET Framework [35], [36]. We use the Barabási-Albert model [37] to generate network topologies with 30 TSN devices and 20 TSN switches.

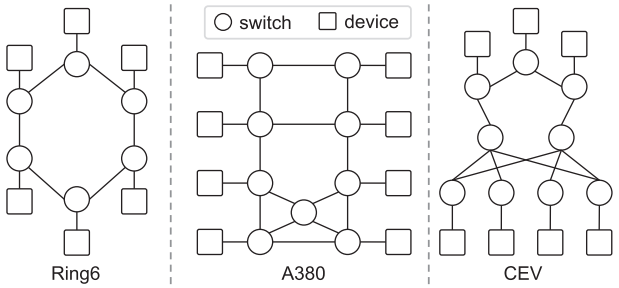


Fig. 11. Topologies in the testbed.

Specifically, new switches are added to the network one at a time. Each new switch is connected to m existing switches with a probability that is proportional to the number of links that existing switches already have. For the devices, each of them is randomly connected to a switch. It's worth noting that all experiments were conducted on the testbed, except for those involving random topology. Simulation experiments serve as a supplement for cases where testbed experimentation is not feasible, because they may underestimate the packet latency [38], [39].

Errors in TSN switches. In our evaluations, we focus on the following three kinds of errors in TSN switches:

- **Incomplete Protocol Support.** Commercial TSN products may not fully implement or comply with the complex and numerous TSN standards, leading to operational discrepancies. To mimic the incomplete support of the guard band mechanism in IEEE 802.1Qbv [2], we intentionally delay the frame transmission time in our experiments.
- **Network Misconfiguration.** It is very error-prone to configure the network switches, especially by hand. We deliberately modify the Gate Control List (GCL) to simulate such network configuration errors.
- **Hardware Logic Flaws.** Hardware designs are not always trustworthy. Sometimes commercial devices may overlook certain corner cases, resulting in hardware logic flows. These bugs are always hidden and hard to detect. We modify the logic of the egress queue on the specific switch's output port to emulate these hardware bugs.

We denote the above three errors as Packet, Gate, and Queue, respectively.

Metrics. We first use success rate, the rate TSNCard localizes the breakdown switches successfully, to evaluate the overall performance. We further measure the bandwidth consumption and localization latency. Since the computation latency of the fault refinement algorithm is relatively small and negligible, we only take the postcard collection latency into account. l_t, the collection latency at cycle t, is measured at 1Mbps bandwidth. The final total postcard collection latency L is calculated as follows

$$L = \sum \max\left(l_t, P\right). \tag{4}$$

Dataset for Bayes algorithm. In order to calculate the prior probability in bandwidth-adaptive postcard prioritization, we generate a large amount of traffic data using the network simulator OMNeT++. Specifically, for every flow schedule in each topology, we generate thousands of sets of network traffic data and use them to statistically compute $P(y, f)$.

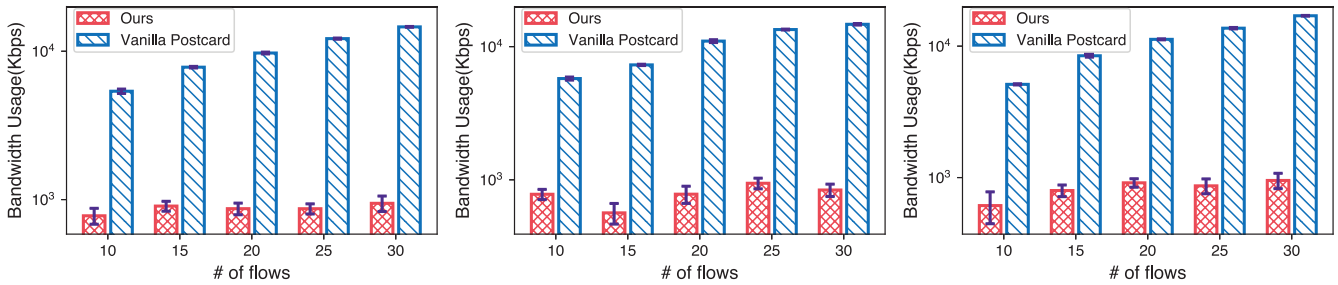
TABLE III
FAULT LOCATION AND TYPE DETECTION SUCCESS RATE. FAULT LOCATIONS AND TYPES WERE CORRECTLY DETERMINED WITH 100% SUCCESS RATE IN ALL CASES

Topology	#Flows	#Cases		
		Queue	Gate	Packet
Ring6	10	147	326	712
	15	197	267	873
	20	131	262	903
	25	146	328	735
	30	164	200	822
A380	10	198	182	868
	15	158	379	748
	20	227	345	801
	25	212	318	719
	30	192	330	684
CEV	10	199	283	774
	15	181	369	640
	20	131	397	750
	25	173	285	822
	30	202	257	851
Random	50	184	255	803
	100	154	376	816
	150	208	421	656
	200	171	359	847

B. Overall Performance

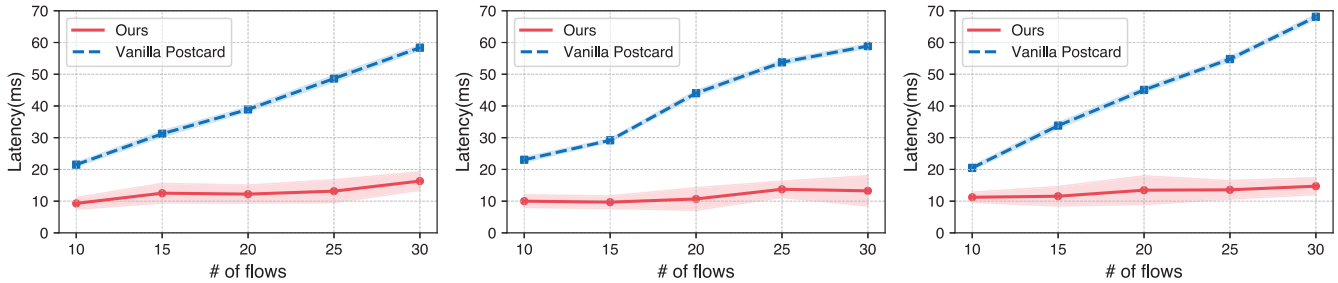
Table III displays TSNCard's success rate of localizing the breakdown switch. We conduct testbed experiments on Ring6, A380, and CEV topologies respectively, and simulation experiments on Random topologies. On each topology, we generate $10 \sim 50$ flows and randomly add errors in one of the switches. We also conduct simulation experiments on Random topology and generate $50 \sim 200$ flows. In every configuration, over 1000 tests are conducted. Theoretically, there might be ring of error situation, where the chain of errors further form a closed loop, making it challenging for TSNCard to pinpoint the exact fault location and fault type of problem. However, we have not observed such phenomenon over thousands of tests. As indicated in the last column of Table III, TSNCard successfully identifies the faulty switches in all test cases, demonstrating its effectiveness. Notably, TSNCard goes beyond merely locating the faulty switches by also identify the specific causes of the errors (Queue Error, Gate Error or Packet Error), providing crucial insights for network administrators to swiftly diagnose and resolve issues.

Figure 12 illustrates the bandwidth consumption of TSNCard over testbed and *vanilla postcard* using a logarithmic scale on the Y-axis. In all test cases, TSNCard utilized bandwidth of less than 1 Mbps with correspondingly lower jitter. Notably, unlike Vanilla Postcard, the bandwidth consumption of TSNCard does not increase linearly with the number of flows, as clearly illustrated in Figure 12(b). This occurs because when a case involves fewer switches and lower interdependencies among flows, TSNCard's algorithm converges



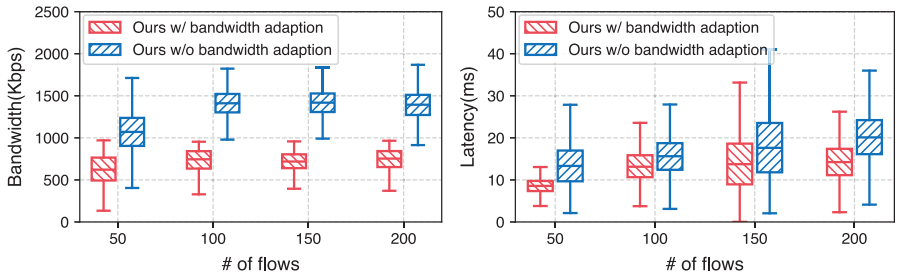
(a) Ring6 (b) A380 (c) CEV

Fig. 12. Bandwidth usage.



(a) Ring6 (b) A380 (c) CEV

Fig. 13. Postcard collection latency.



(a) Bandwidth usage (b) Postcard Collection Latency

Fig. 14. Performance on random topology.

more quickly and requires fewer postcards to identify the faulty switches.

Figure 13 depicts the postcard collection latency of TSNCard and *vanilla postcard* over testbed. TSNCard exhibits a stable latency of around 10ms, largely unaffected by traffic scale. On the contrary, the *vanilla postcard* already experiences a latency of around 20ms with 10 flows, reaching approximately 60ms with 30 flows. TSNCard excels a $50 \sim 83\%$ improvement compared to the *vanilla postcard*.

C. Component Study

We conduct several experiments to understand the effectiveness of each module in TSNCard.

Bandwidth Adaptive postcard prioritization. Experimental data is from simulation. Figure 14a validates the importance of the bandwidth adaptation module by comparing it against a baseline that excludes this module. Our experimental results indicate that, in most cases, TSNCard's bandwidth consumption is lower than that of the baseline. Moreover, due to the cross-cycle mechanism, the bandwidth consumption of both methods does not increase linearly with the number of flows. Figure 14b presents the time taken to collect postcards for both methods, showing that TSNCard requires slightly less average time and jitter compared to the baseline. We observed that when the number of flows exceeds 100, the latency remains nearly constant, as the error chain length generated by the randomly produced traffic does not vary with the flow volume.

Second, we dive into the Bayes-guided postcard prioritization module. We compare it with three postcard selection strategies

- **Random**: Select K switches on the current flows at random.
- **Uniform**: Select K switches on the current flows uniformly.
- **Greedy**: Select K switches where most flows pass on the current flows.

As shown in Figure 15, we measured the data collection latency of these methods at different bandwidths by simulation. The latency of Bayes-guided postcard prioritization is always the lowest compared to the other three strategies.

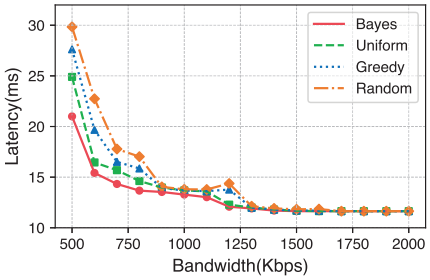


Fig. 15. Performance on different searching methods.

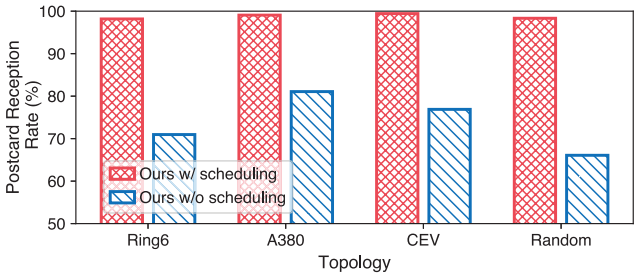
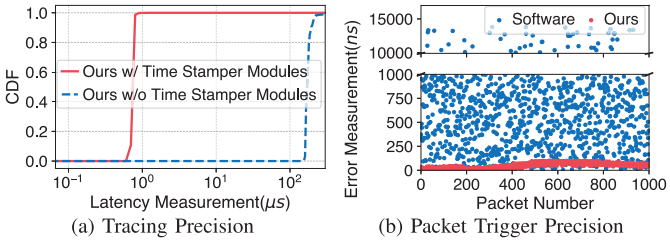


Fig. 16. Postcard reception rate on different topologies.



(a) Tracing Precision (b) Packet Trigger Precision

Fig. 17. Time measurement accuracy.

Moreover, it is notable that the latency of the four methods becomes almost the same when the bandwidth is larger than 1.5Mbps, this is because the bandwidth is large enough to collect all postcards of the current flows in one cycle.

Critical-Postcard Flow Scheduling. We explored the impact of Critical-Postcard Flow Scheduling on TSNCard over testbed. Figure 16 shows the postcard reception rate across different topologies. With Critical-Postcard Flow Scheduling, TSNCard maintains nearly a 100% reception rate across all topologies. However, in some cases, insufficient bandwidth prevents postcards from being scheduled as high-priority traffic, leading to packet loss. Without this mechanism, TSNCard's reception rate typically drops below 80%, and in the Random topology, it falls below 70% due to its complexity and the impact of critical traffic on postcard transmission.

Time Measurement Accuracy. On the testbed, we investigated the impact of removing the hardware Time Stamper module from TSNCard switch on measurement accuracy, as shown in Figure 17a. The comparative approach involved measuring packet propagation delay through software-based *rx* and *tx* timestamps. Experimental results indicate that TSNCard achieves an accuracy two orders of magnitude higher than the software method, with software measurements exhibiting

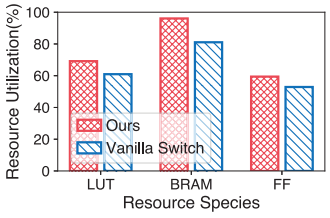


Fig. 18. Resource utilization.

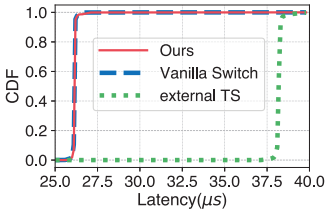


Fig. 19. Packet forwarding.

errors of up to hundreds of microseconds. Hence, the hardware Time Stamper Module is critical for tracing.

We validated the necessity of the Packet Trigger Module by measuring the errors in periodically (1ms) transmitted packets. As shown in Figure 17b, the accuracy error with the Packet Trigger Module remains within $100ns$, while the error with standard software packet transmission can exceed 14 microseconds, rendering the latter unsuitable.

D. Overhead Study

We evaluate the overhead that TSNCard introduces to the original TSN switches over our testbed.

Hardware resource utilization. Figure 18 illustrates TSNCard's utilization of Look Up Table (LUT), Block RAM (BRAM) and Flip Flops (FF) on FPGA. Compared to the original TSN switch, TSNCard uses around 8% more LUT, 15% more BRAM and 7% more FF. The result demonstrates that TSNCard demands few hardware resources and could be well integrated into traditional TSN switches with little or no additional hardware cost.

Packet Forwarding Delay. We compare the packet forwarding delay of the TSNCard switch to that of a vanilla TSN switch and External TS (see Figure 8(a)). With constant propagation delay, packets are sent between two TSN devices via a switch to measure end-to-end delay (Figure 19). The hardware modules introduced by TSNCard have minimal impact on packet forwarding delay, offering an almost imperceptible method for monitoring critical TSN traffic. In contrast, the delay of External TS increases significantly, making it unsuitable for TSN monitoring.

VI. Related Work

We will review the network diagnostic techniques in this section.

Probing involves sending test packets to assess network conditions like latency, packet loss, and throughput. Classic tools like Ping [7] and Traceroute [8] diagnose connectivity

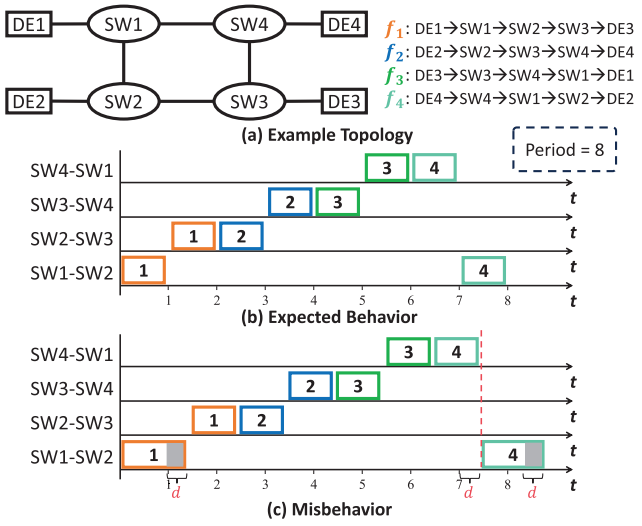


Fig. 20. Ring of errors.

issues and map network topology. Advanced methods, such as Pingmesh [9], measure latency in large-scale data centers, while TSLP [10] detects congestion on inter-domain links.

Tracing tracks packet paths to identify bottlenecks and performance issues. Tools like Tcpdump [11] and Wireshark [12] are vital for troubleshooting. Everflow [40] traces specific packets for fault resolution in data centers, while dShark [13] focuses on distributed packet capture for diagnosing network problems.

In-band network telemetry (INT) [14] is an emerging technique for collecting and analyzing network statistics. To reduce telemetry data per packet, PINT [15] encodes data across multiple packets, while TCP-INT [16] makes telemetry data available to end-hosts for a more integrated view of network and application performance.

Postcard telemetry [17] sends packet metadata to an analytic server, enabling network telemetry without affecting packet forwarding time. HyperSight [18] uses a query language and Bloom Filter Queue for behavior-level monitoring. NetSight [19], based on the postcard mechanism, captures packet histories, offering a detailed view of network traffic for efficient troubleshooting.

Besides traditional Ethernet diagnostics, some preliminary work focuses on Time-Sensitive Networking. [41] proposes a method to measure the end-to-end latency of TSN networks on commercial-grade TSN-ready devices. TSN-Insight [5] addresses time-synchronization issues, computing clock offsets between nodes and the leader node, but it requires known topology and transmission delays. TSNPeeper [6] introduces a model that stores TSN forwarding misbehavior on switches and uses active probing to collect this information. However, it cannot identify fault location and type in the potential chain of errors.

VII. Discussion

In this section, we discuss the limitations of TSNCard and future research directions.

Ring of Errors. In some extreme cases (refer to Figure 20), the proposed cross-cycle data fusion may fail. In the ring topology, there are four data flows f_1, f_2, f_3 and f_4 (refer to Figure 20a), all with a period of 8. We focus on the traffic between four switches. The expected schedules of four flows are shown in Figure 20b. Assuming a hardware flaw occurs on SW1, resulting in each packet being forwarded with a delay of d longer than normal (refer to Figure 20c). So, f_1 will pass through (SW1, SW2) at time 0 and through (SW2, SW3) at time $1+d$, and the forwarding of f_2, f_3, and f_4 will also be delayed by d. However, note that the forwarding of f_4 at SW1 is also delayed by d, f_1 in the second period can only be transmitted at time $8+2d$. Similarly, in the k-th period, f_1 will pass through (SW1, SW2) at time $8(k-1)+2(k-1)d$, causing all other flows to be transmitted $(2k-1)d$ time units later than expected. Therefore, in this case, the packet behavior within different periods varies, which means cross-cycle data fusion is not applicable. We name these cases **Ring of Errors**. Overall, the Ring of Errors is rare (refer to §V-B), and we can easily evade it by adjusting the scheduling table. We believe that the TSNCard is effective in almost all cases, and leave Ring of Errors as a direction for future research.

Error Scope and Limitations: TSNCard is designed to diagnose persistent switch errors in TSN environments effectively. However, transient faults may also occur in real network environments due to various hardware failures like brief link interruptions. It is currently limited in its ability to detect and analyze these errors, as these require a prolonged time span to collect sufficient diagnostic data across cycles. This limitation stems from the system's reliance on cross-cycle telemetry and its focus on reconstructing consistent misbehavior patterns. If bandwidth constraints were removed and all switches in the network recorded every packet's information, TSNCard could diagnose the fault. However, this approach would be costly, as it would require tens of Mbps of bandwidth to monitor a difficult-to-predict transient fault. Additionally, TSNCard is limited to diagnosing faults within switches and does not extend its diagnostic capabilities to end devices in the network. Addressing transient errors remains a challenge and an avenue for future research.

VIII. Conclusion

In this work, we design and implement TSNCard, a pioneering network diagnostic system for Time-Sensitive Networking. TSNCard represents a significant leap forward in TSN diagnostic capabilities, seamlessly integrating advanced telemetry protocols, fault refinement algorithms, and cutting-edge hardware design. On this basis, TSNCard enables us to localize the network fault in bandwidth-constrained TSN applications quickly and accurately. Extensive evaluations on simulation and testbeds demonstrate TSNCard's superior performance. We envision TSNCard as an important step in advancing TSN development.

References

[1] H. Lasi, P. Fettke, H. G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," *Bus. Inf. Syst. Eng.*, vol. 6, no. 4, pp. 239–242, 2014.
[2] Enhancements for Scheduled Traffic, IEEE Standard 802.1Qbv, 2015.

[3] S. Ahmad et al., "Analyzing critical failures in a production process: Is industrial IoT the solution?," *Wireless Commun. Mobile Comput.*, vol. 2018, no. 1, pp. 1–12, Jan. 2018.

[4] S. Sulaiman and N. Ismail, "Analysis of failure in manufacturing machinery," *Proc. IOP Conf. Ser., Mater. Sci. Eng.*, vol. 50, p. 11, Dec. 2013.

[5] T. Bu, Y. Yang, X. Yang, W. Quan, and Z. Sun, "TSN-Insight: An efficient network monitor for TSN networks," in *Proc. Sigcomm Poster*, 2019.

[6] C. Zhang et al., "TSN-peeper: An efficient traffic monitor in time-sensitive networking," in *Proc. IEEE 30th Int. Conf. Netw. Protocols (ICNP)*, Sep. 2022, pp. 1–11.

[7] J. Postel, *Internet Control Message Protocol*, document RFC 792, Internet Eng. Task Force, Request for Comments, Sep. 1981.

[8] G. S. Malkin, *Traceroute Using an IP Option*, document RFC 1393, Internet Eng. Task Force, Request Comments, Jan. 1993.

[9] C. Guo et al., "Pingmesh: A large-scale system for data center network latency measurement and analysis," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 139–152, Aug. 2015.

[10] A. Dhamdhere et al., "Inferring persistent interdomain congestion," in *Proc. Conf. ACM Special Interest Group Data Commun.*, NY, NY, USA, Aug. 2018, pp. 1–15.

[11] (Nov. 2023). *Tcpdump and Libpcap*. [Online]. Available: https://www.tcpdump.org/index.html

[12] Wireshark. (Nov. 2023). *The World's Most Popular Network Protocol Analyzer*. [Online]. Available: https://www.wireshark.org/

[13] D. Yu et al., "DShark: A general, easy to program and scalable framework for analyzing in-network packet traces," in *Proc. 16th USENIX Symp. Networked Syst. Design Implement. (NSDI 19)*, Jul. 2019, pp. 207–220.

[14] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, and L. J. Wobker, "In-band network telemetry via programmable dataplanes," in *Proc. Conf. ACM Special Interest Group Data Commun. (SIGCOMM)*, New York, NY, USA, 2015, pp. 1–2.

[15] R. Ben Basat, S. Ramanathan, Y. Li, G. Antichi, M. Yu, and M. Mitzenmacher, "PINT: Probabilistic in-band network telemetry," in *Proc. Annu. Conf. ACM Special Interest Group Data Commun. (SIGCOMM)*, New York, NY, USA, 2020, pp. 662–680.

[16] G. Jereczek, T. Jepsen, S. Wass, B. Pujari, J. Zhen, and J. Lee, "TCP-INT: Lightweight network telemetry with TCP transport," in *Proc. SIGCOMM Poster Demo Sessions*, Aug. 2022, pp. 58–60.

[17] H. Song et al., *On-Path Telemetry Using Packet Marking to Trigger Dedicated OAM Packets*, document Internet Draft draft-song-ippm-postcard-based-telemetry-16, Internet Engineering Task Force, 2023.

[18] Y. Zhou et al., "HyperSight: Towards scalable, high-coverage, and dynamic network monitoring queries," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 1147–1160, Jun. 2020.

[19] N. Handigol, B. Heller, V. Jeyakumar, D. Mazières, and N. McKeown, "I know what your packet did last hop: Using packet histories to troubleshoot networks," in *Proc. 11th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, Seattle, WA, USA, Apr. 2014, pp. 71–85.

[20] Xilinx. (Jul. 2021). *Socs With Hardware and Software Programmability*. [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html

[21] Timing Synchronization for Time-Sensitive Appl., IEEE Standard 802.1AS, 2020.

[22] X. Zhuge et al., "InNetScheduler: In-network scheduling for time- and event-triggered critical traffic in TSN," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, May 2024, pp. 421–430.

[23] E. Li, F. He, Q. Li, and H. Xiong, "Bandwidth allocation of stream-reservation traffic in TSN," *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 1, pp. 741–755, Mar. 2022.

[24] Z3Prover. (2021). *Github Repository of Z3prover*. [Online]. Available: https://github.com/Z3Prover/z3

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

[26] X. He, X. Zhuge, F. Dang, W. Xu, and Z. Yang, "DeepScheduler: Enabling flow-aware scheduling in time-sensitive networking," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, May 2023, pp. 1–10.

[27] N. Zilberman, Y. Audzevich, G. A. Covington, and A. W. Moore, "NetFPGA SUME: Toward 100 Gbps as research commodity," *IEEE Micro*, vol. 34, no. 5, pp. 32–41, Sep./Oct. 2014.

[28] Z. Yang et al., "CaaS: Enabling control-as-a-service for time-sensitive networking," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, New York, NY, USA, May 2023, pp. 1–10.

[29] A. Limited. (2010). *Amba AXI-stream Protocol Specification*. [Online]. Available: https://developer.arm.com/documentation/ihi0022/latest

[30] K. Group. (2023). *Pcie-0400-tsn Network Interface Card*. [Online]. Available: https://www.kontron.com/en/products/pcie-0400-tsn-network-interface-card/p151637

[31] I. Corporation. (2023). *Intel TofinoT 2*. [Online]. Available: https://www.intel.com/content/www/us/en/products/details/network-io/intelligent-fabric-processors/tofino-2.html

[32] F. Boulanger, D. Marcadet, M. Rayrole, S. Taha, and B. Valiron, "A time synchronization protocol for A664-P7," in *Proc. IEEE/AIAA 37th Digit. Avionics Syst. Conf. (DASC)*, Sep. 2018, pp. 1–9.

[33] D. Tămaş-Selicean and P. Pop, "Optimization of TTEthernet networks to support best-effort traffic," in *Proc. IEEE Emerg. Technol. Fact. Autom. (ETFA)*, Jul. 2014, pp. 1–4.

[34] P. N. America. (2019). *Industrial Topology Options and Profinet*. [Online]. Available: https://us.profinet.com/wp-content/uploads/2019/08/Topology.pdf

[35] A. Varga, "OMNeT++," in *Modeling and Tools for Network Simulation*. Berlin, Germany: Springer, 2010, pp. 35–59.

[36] (Nov. 2023). *INET Framework for OMNEST/OMNeT++*. [Online]. Available: https://github.com/inet-framework/inet

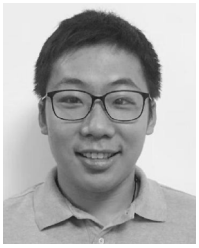
[37] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[38] M. Bosk et al., "Methodology and infrastructure for TSN-based reproducible network experiments," *IEEE Access*, vol. 10, pp. 109203–109239, 2022.

[39] H.-H. Liu et al., "Improving TSN simulation accuracy in OMNeT++: A hardware-aligned approach," *IEEE Access*, vol. 12, pp. 79937–79956, 2024.

[40] Y. Zhu et al., "Packet-level telemetry in large datacenter networks," in *Proc. 2015 ACM Conf. Special Interest Group Data Commun.* London, U.K.: ACM, Aug. 2015, pp. 479–491.

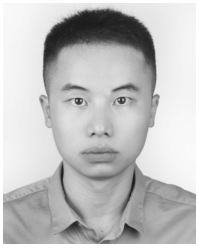
[41] S. Sudhakaran, C. Hall, D. Cavalcanti, A. Morato, C. Zunino, and F. Tramarin, "Measurement method for end-to-end time synchronization of wired and wireless TSN," in *Proc. IEEE Int. Instrum. Meas. Technol. Conf. (I2MTC)*, May 2023, pp. 1–6.



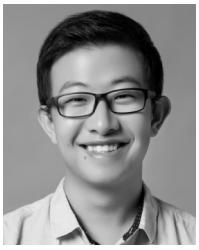
Xiangwen Zhuge (Student Member, IEEE) received the B.E. degree from the School of Software, Tsinghua University, Beijing, China, in 2023, where he is currently pursuing the Ph.D. degree. His research interests include time-sensitive networking and large language model inference acceleration.

Zeyu Wang (Student Member, IEEE) received the B.E. degree from the School of Software, Tsinghua University, in 2020, where he is currently pursuing the Ph.D. degree. His research interests include time-sensitive networking and mobile computing.

Xiaowu He (Member, IEEE) received the B.E. degree from the School of Computer Science and Engineering, University of Electronic Science and Technology of China, in 2019, and the Ph.D. degree from the School of Software, Tsinghua University, in 2024. His research interests include time-sensitive networking, edge computing, the Internet of Things, and datacenter networking.

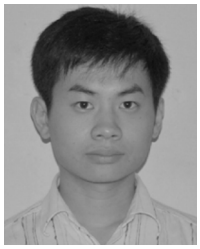


Shen Xu is currently pursuing the B.E. degree with the School of Software, Tsinghua University. His research interests include machine learning and large language models.



Fan Dang (Senior Member, IEEE) received the B.E. and Ph.D. degrees in software engineering from Tsinghua University, Beijing, in 2013 and 2018, respectively. He is currently an Assistant Professor at the School of Software Engineering, Beijing Jiaotong University, Beijing. His research interests include industrial intelligence and edge computing. He is a member of ACM.

Jingao Xu (Member, IEEE) received the B.E. and Ph.D. degrees from the School of Software, Tsinghua University, in 2017 and 2022, respectively. He is currently a Post-Doctoral Researcher at the Living Edge Lab, Computer Science Department, Carnegie Mellon University. His research interests include the Internet of Things and mobile computing.



Zheng Yang (Fellow, IEEE) received the B.E. degree in computer science from Tsinghua University and the Ph.D. degree in computer science from the Hong Kong University of Science and Technology. He is currently an Associate Professor with Tsinghua University. His main research interests include the Internet of things, industrial network, and AI computing system.

Qiang Ma (Member, IEEE) received the B.S. degree from the Department of Computer Science and Technology, Tsinghua University, China, in 2009, and the Ph.D. degree from the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, in 2013. He is currently an Assistant Researcher at the Qiyuan Lab. His research interests include wireless sensor networks, mobile computing, and privacy.