Transparent TSN for Agnostic End-hosts via P4-based Traffic Characterization at Switches

Cornelia Brülhart*, Nurefşan Sertbaş Bülbül*, Nils Ole Tippenhauer†, Mathias Fischer*
*Universität Hamburg, Germany
{cornelia.bruelhart, nurefsan.sertbas, mathias.fischer}@uni-hamburg.de
†CISPA Helmholtz Center for Information Security, Germany
tippenhauer@cispa.de

***Abstract*—Mission-critical networks currently face a transition from legacy network protocols to advanced time-sensitive networking (TSN) standards. TSN guarantees reliable and deterministic communication using off-the-shelf Ethernet equipment. However, end-hosts must be TSN-aware and may pose security risks by arbitrarily over-allocating resources. Integrating central instances like a software-defined networking (SDN) controller into TSN networks to streamline network management presents a promising solution. This raises concerns regarding latency in communication between switches and the controller, as well as among switches themselves. To address this, we propose an approach that renders TSN transparent to end-hosts, eliminating the need for their involvement in resource reservations. We embed packet processing logic in P4-enabled TSN switches to characterize network traffic intelligently. This enables switches to allocate network resources autonomously and adjust real-time traffic handling mechanisms. Leveraging P4 storage structures introduces statefulness for traffic characterization computing within the inherently stateless P4 language. Our experiments demonstrate that our P4-enhanced switches require a minimal 0.014 MB of switch memory to distinguish between periodic and non-periodic traffic with an 80% precision while incurring a mere 0.2 ms forwarding latency per packet.**

***Index Terms*—P4 Programming, Traffic Characterization, Software-Defined Networking, Time-Sensitive Networking, Real-Time Traffic Management**

I. Introduction

MISSION-CRITICAL networks, characterized by their deterministic and reliable communication in low-latency settings, face the challenge of adapting to modern system environments' increasing heterogeneity and agility. In response, time-sensitive networking (TSN) has emerged, offering deterministic and reliable communication in low-latency settings that align well with the demands of modern system environments. For example, network infrastructure in smart manufacturing facilities must meet stringent quality of service (QoS) and time-sensitive demands, where integrity in real-time data exchange is crucial for seamless operations. However, as there is a shift towards adopting cost-effective commercial off-the-shelf (COTS) hardware and software over specialized field buses, orchestrating interactions among network components remains complex in real-time applications. Implementing TSN using COTS hardware and software can ensure QoS, but it requires for end-hosts to be TSN-aware, thus restricting the network's flexibility.

In response to this, ongoing research explores the feasibility of central control in time-sensitive networks. Network management can be facilitated, e.g., through the holistic view of a centralized entity, similar to the concept of software-defined networking (SDN) [1]. SDN allows modification and specification of network behaviour through software, effectively separating the control plane from the data plane. A central entity, known as the SDN controller, dynamically manages the traffic and configures the forwarding behaviour of the switches based on the specific application requirements. However, SDN also introduces challenges, especially in heterogeneous network environments, such as TSN. The central (TSN) controller must accommodate diverse device characteristics and network policies. This may result in latencies of up to 0.3 ms per packet [2] and significantly impact network operations' real-time nature. Additionally, the dynamic traffic and resource orchestration in TSN networks involve frequent communication between network components and the TSN controller [3], [4]. As network traffic increases, the controller must process high volumes of messages, which may become a bottleneck in the network.

These challenges show the urgent need for a transparent TSN mechanism that guarantees dynamic, deterministic, and low-latency communication for the desired QoS without mandating TSN-capable end-hosts. In this work, we propose to shift TSN tasks and configuration responsibilities from the network's periphery (the end-hosts) to its internal network components to address this challenge. This enables the network to autonomously manage and adapt to traffic demands and communicate requests, achieving transparent TSN. By identifying various traffic types in real-time, the network can dynamically allocate resources, reduce the burden on end-hosts, and minimize messaging frequency to the controller. Hence, our solution for transparent TSN involves transferring TSN tasks and network management responsibilities from the end-hosts and the TSN controller to the network switches. Empowering switches to manage and identify traffic locally eliminates the need for end-host TSN-awareness. Furthermore, autonomous traffic type characterization at switches reduces messaging frequency between the TSN controller and switches, minimizing network load and reducing latency.

Our approach uses P4[1], a protocol-independent programming language designed to define custom forwarding behaviour and per-packet processing within network devices.

P4 operates in a stateless manner, treating each packet in isolation without relying on past data. We overcome P4's statelessness by leveraging P4 registers – fast-access memory units that can store data elements in arrays-like structures for calculations, metadata storage, and counting purposes. Accordingly, our contributions are:

- Our novel approach eliminates the need for end-host TSN-awareness. We simplify TSN resource reservation process and make it more flexible while enhancing compatibility with diverse devices.
- We enhance switches with local traffic characterization capabilities, reducing the need for extensive message exchange between network components.
  Our approach distinguishes between periodic vs. non-periodic flows with an accuracy of 80%, all while introducing a mere 0.2 ms per flow. We remove the need to announce each flow to the controller as typically seen in conventional networks and substantially enhance overall network performance in time-sensitive networks.

Utilizing P4, we aggregate inter-arrival times across three types of traffic (periodic, sporadic, and burst), facilitating direct extraction of their traffic patterns within the switch. We demonstrate how the strategic use of P4 registers enables the application of statistical analysis for per-flow characterization. Our lightweight solution requires only 0.014MB of switch memory for flow characterization in time-sensitive networks. Furthermore, we significantly reduce the message exchange between network participants compared to traditional stream reservation protocol (SRP) and SDN networks [2].

The remainder of this paper is organized as follows. Section II offers background information, while Section III describes related work. In IV, we introduce our system architecture for transparent TSN. A description of our approach for autocorrelation computation follows this. We evaluate our solution in V, share and describe our experiment results, and discuss our findings. Finally, section VI concludes the paper.

II. Background

Traffic characterization and monitoring are crucial in time-sensitive networks. They ensure efficient resource allocation and deterministic delivery of critical traffic. Various approaches exist for resource reservation and network configuration: Figure 1a illustrates a distributed approach akin to SRP, where traffic management responsibilities are decentralized among network participants. Whereas, in Figure 1b an SDN-like mechanism offers enhanced flexibility and adaptability, leveraging a holistic network view. Figure 1c outlines a decentralized approach, where a centralized entity carries out traffic characterization at the data plane. In the subsequent sections, we describe the functionalities and implications of these mechanisms.

[1]https://opennetworking.org/p4/

A. Standard Stream Reservation Protocol

Standard Ethernet operates on a best-effort basis, often resulting in unpredictable latency and packet loss during heavy network traffic load [5]. Ethernet switches forward packets solely based on their destination MAC addresses without considering timing or priority requirements. To address this, the IEEE TSN task group presented SRP to ensure precise resource allocation, low latency, and deterministic communication in TSN. As seen in Figure 1a, the sending end-host (talker) initiates the communication by sending a reservation request called Talker Advertise (TA). This message contains a unique identifier, a stream ID, required resources and QoS parameters. Every network device evaluates this message along the reserved stream's path, assessing whether it can satisfy the specified requirements to maintain QoS standards. Upon successful reservation, the receiving end-host (listener) responds with a Listener Ready (LR) message, and traffic is forwarded along the path. However, these negotiations introduce latency and inefficiencies, specifically in path re-negotiations due to changing traffic patterns or link/switch failures. Additionally, managing reservations across large-scale networks with SRP requires active participation from end-hosts (TSN-awareness).

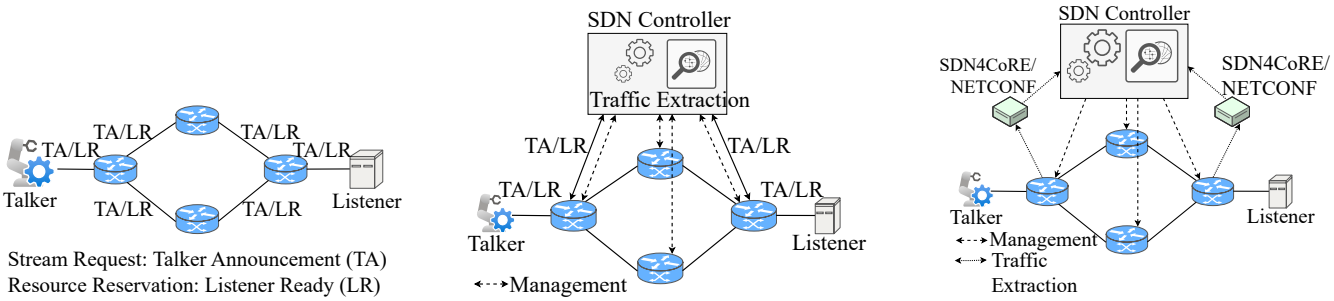
B. Traffic Characterization at the Control Plane

As illustrated in Fig.1b SRP can be enhanced with SDN by shifting resource reservation tasks and the end-host dependency to the network, specifically to the (TSN) controller. The TSN controller enables centralized network management and simplifies the configuration and dynamic provisioning of reservations tailored to traffic demands [4]. The TSN controller's task includes continuously monitoring network devices, available link capacities, and network traffic patterns. A talker initiates communication with a reservation request message to the TSN controller, which contains the Stream ID and encapsulated QoS parameters. The TSN controller then determines the optimal path for the designated stream, allocates resources along the reserved path, and (re-)configures the data plane entities. However, the centralized TSN controller poses a potential single point of failure. Moreover, as the network scales up in size and complexity, the management overhead associated with SDN introduces latency in the reservation process, impacting the network's performance.

C. Traffic Characterization at the Data Plane

The forwarding latency between TSN controller and network devices can be reduced with additional components at the data plane. External components can, e.g. be deployed next to or into the switches to perform traffic characterization or processing.

External Monitoring and Extraction: Fig.1c illustrates traffic monitoring and characterization at the data plane by deploying external devices. These devices, whether physical appliances or virtual instances are strategically placed at crucial network topology points or alongside switches [6] to perform traffic characterization. However, this also introduces challenges: Synchronizing all network participants to maintain



(a) Distributed: TSN-aware end-hosts (b) Centralized: Controller traffic processing (c) Decentralized: External traffic processing

Fig. 1: Resource reservation and network configuration strategies for TSN.

accurate data transmission timing adds complexity and resource overhead. External network components also introduce security vulnerabilities, privacy concerns, and delays when interacting with other network participants.

Decentralized In-Switch Monitoring and Extraction: Instead of relying on external components or a centralized TSN controller, we propose directly integrating traffic characterization and management capabilities within switches. Our method removes the latency introduced by control-to-data plane communication in centralized systems, leading to more efficient and quicker resource allocation. Furthermore, it eliminates the need for TSN-aware end-hosts, as enforced by the SRP, and minimizes the volume of messages typical in SDN-based networks [2]. By enhancing switches with computing capabilities to identify traffic types locally, our novel approach establishes a transparent TSN environment with significantly fewer message exchanges between network participants.

III. Related Work

This section presents related work on the resource reservation and configuration of time-sensitive networks. SRP proposed by [5] and [7], is essential for deterministic communication and resource allocation management along communication paths in TSN. The authors also observed limitations of TSN, such as the reliance on end hosts to participate actively in the SRP process. Further challenges of SRP have been identified by [8], including consistency problems where listeners expect streams but resources are not reserved. This can be particularly challenging in the field of internet of things (IoT), as investigated in the work of [9], due to IoT devices' large scale, heterogeneity, and resource-constrained nature. To address this issue, Sousa et al. describe modified bridges incorporating Media Access Control (MAC) bridge functionalities based on IEEE 802.1D and 802.1Q protocols. However, the MAC Bridge Ethernet protocol extension introduces additional messages for information exchange between switches. Despite this modification, SRP remained indispensable for resource management, indicating that end hosts must maintain TSN-awareness.

Thi et al. [10] investigated the integration of SDN into TSN networks and discussed the advantages of a centralized and programmable network control plane. A holistic network view allows for a more dynamic and flexible management of network resources. SDN facilitates dynamic resource allocation based on the real-time requirements of different streams and traffic type prioritization. However, the necessary interaction between the TSN Controller and the network devices poses multiple challenges: The messages between the centralized controller and network devices increase as the network scales. This can lead to network congestion and increased latency, particularly in frequent network changes. Furthermore, configuring the TSN controller itself is a complex task, as investigated by [4]. Gutiérrez et al. [11] introduce a self-configuration framework for real-time networks, leveraging a configuration agent to monitor TSN networks and acquire pertinent information. During the learning phase, the agent gathers raw network data from switches. The architecture includes switches with reconfiguration and learning capabilities and a master switch that distributes configurations among switches to minimize downtime during reconfiguration and enhance network availability and reliability. However, as Gutiérrez et al. [12] also concluded, exchanging learning messages introduces additional network overhead, potentially causing congestion and latency. Additionally, interoperability with various network devices adds complexity to network implementation and management [6]. Overall, all discussed approaches have notable shortcomings. Time-sensitive networks utilizing SRP bring forth inconsistent resource allocation, scalability issues, and a require TSN aware end host. Integrating a TSN controller allows for adaptability to network conditions. However, the communication between SDN controller and network devices increases the network congestion and latency. Lastly, the discussed decentralized approach for a self-configurable TSN network requires the deployment of switches with varying levels of complexity. This also requires interoperability with various network devices and additional network overhead due to the amount of communication between network devices.

IV. P4-based Traffic Extraction

This section introduces our P4-based packet parameter extraction approach for traffic characterization per flow. We describe our approach in further detail in the following section.

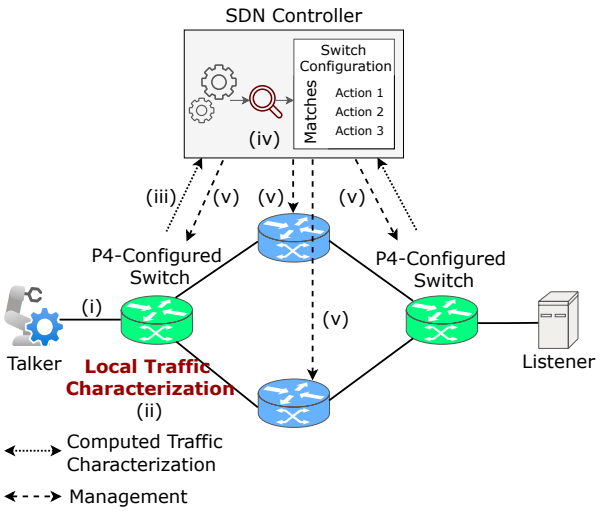


Fig. 2: Transparent TSN with local traffic characterization on edge-switches.

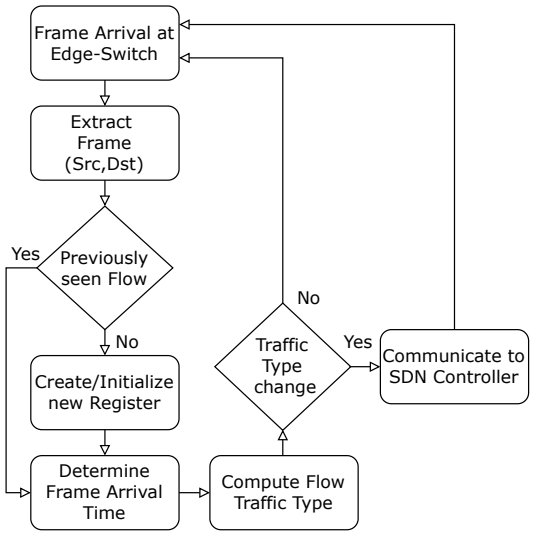


Fig. 3: Local traffic characterization on switches in TSN networks.

A. Overall System

We follow a decentralized network configuration model, as illustrated in Fig.2, for a transparent TSN. Utilizing SDN in TSN, we create a dynamic and adaptive TSN environment that effectively manages traffic flows.

At the network's edge, talkers initiate communication by generating data packets, which encapsulate various parameters, such as priority levels, timing constraints, and source/destination address (i). Edge switches disassemble packets, extract traffic characteristics and determine the traffic type locally Fig.2 (ii). Any identified changes in traffic type or other characterization updates can then be relayed to the TSN controller for further action (iii). Leveraging this information, the TSN controller evaluates the traffic type changes and fine-tunes routing decisions (iv). The TSN controller, equipped with a global network view, can dynamically (re)configure switches to optimize traffic routing or implement security measures in response to suspicious network activity (v).

We divide our traffic characterization into two phases:

Learning Phase The switch receives N packets, determines each packet arrival time and extracts relevant parameters, such as protocol type and source/destination Internet protocol (IP). The switch uses these parameters to characterize the baseline traffic type per flow.

Validation Phase After an initial traffic characterization upon receiving N packets, the switch transitions into the validation phase. The switch verifies the flow traffic type over time by continuously calculating the traffic type of the subsequent N packets.

B. P4-based Traffic Analysis

Implementing packet processing and traffic characterization capabilities in switches usually relies on hardware-based solutions or proprietary software, which can be costly and may entail vendor lock-ins. Flexibility is needed to program data plane forwarding behaviour to enhance switches and ensure efficient packet processing to guarantee low latency in TSN networks. In this context, P4 emerges as an ideal candidate for this task. P4, a domain-specific language, is tailored for programming network data planes, allowing precise behaviour definitions for devices like switches and routers [13]. It grants flexibility to customize packet processing to meet specific network demands. While inherently stateless, P4 operates per-packet and does not retain information or maintain state between packet processing operations. To overcome this limitation, we use registers, which serve as P4 memory elements and can store and manipulate packet data across time. We leverage registers to store intermediate results, metadata and counter values to perform iterative statistical analysis and pattern recognition computations. Using read/write operations on these registers offers a workaround to the absence of, e.g., traditional for-loops, complex data structures such as arrays and recursive function calls in P4 programming.

Fig.3 depicts the flow of characterizing traffic through a P4 configured switch. As soon as a packet arrives at the switch's ingress port, the P4 packet processing pipeline is initiated. The P4 parser extracts relevant fields from the incoming frame to identify a flow, such as the source and destination IP address. A new register is created if the flow has not been observed previously. The current frame's arrival time is then determined using the ingress global timestamp metadata. This data is then utilized to determine packet inter-arrival times and traffic characterization. The switch proceeds to process the next N number of packets for traffic characterization and determines if a change in traffic type occurred.

P4-Register Manipulation and Statistical Analysis: We utilize multiple registers with different purposes to perform traffic characterization calculations. Strategic use of these registers enables us to retain essential information regarding packet flows and execute calculations for traffic character-

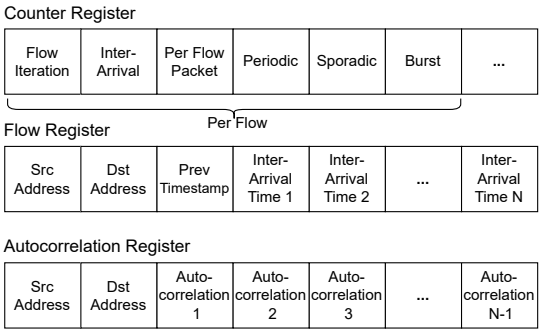


Fig. 4: Registers for reading/writing: Inter-arrival times, autocorrelation results per flow and one main counter register for iterating and storing counters.

ization within the switch. Our approach utilizes one flow, one autocorrelation register per flow, and one general counter register for all flows and traffic characterizations.

a) *Flow Register:* As shown in Fig.4, the first two entries contain the packet's source and destination addresses to identify the flow uniquely. The third entry acts as a temporary buffer, storing the timestamp of the previous packet processed within this flow. Starting from the fourth entry, the register exclusively holds calculated inter-arrival times between successive flow packets.
b) *Autocorrelation Register:* Utilized for storing the flow's autocorrelation values at varying lag intervals.
c) *Counter Register:* Designed for tracking various metrics associated with the flow and facilitating iterative operations as packets are processed.

Performing traffic characterization locally on a switch entails several challenges. Switches typically lack the processing power to handle complex calculations and are constrained by limited memory capacity. Furthermore, packets must be processed within strict time constraints to avoid network disruptions. To address this, we choose autocorrelation analysis to determine traffic patterns over time, as outlined in formula 1. Autocorrelation quantifies the degree of similarity between an event and its delayed version across different lag values, offering a robust means to measure the repetitiveness of traffic patterns. Compared to complex machine learning methods, it is computationally efficient and imposes minimal overhead on switch resources.

$$R(k) = \frac{\sum_{t=1}^{N-k}(x_t - \bar{x})(x_{t+k} - \bar{x})}{\sum_{t=1}^{N}(x_t - \bar{x})^2} \quad \forall N, \bar{x}, k \in \mathbb{N} \qquad (1)$$

$R(k)$: Autocorrelation coefficient at lag k.
N : Total number of frame inter-arrival times.
$x_t \geq 0$: Inter-arrival time of the t-th frame.
$\bar{x}$: Mean inter-arrival time.
k : Lag, inter-arrival times displacement.

After concluding the learning phase and computing the autocorrelation of *N*-1 frames, we analyze specific points at non-adjacent time lags within the autocorrelation register. This localized assessment is conducted across three distinct ranges within the register. We discern patterns indicative of periodic, sporadic, or burst traffic by quantifying the degree of similarity between autocorrelation values at these selected positions. Subsequently, the validation phase initiates as we preserve these outcomes for the next iteration of traffic characterization. The switch repeats these processes until the registers reach capacity *N* to characterize traffic and compare results to the previously obtained data.

V. Evaluation

In this section, we employ various metrics and experiments to evaluate our traffic characterization approach. For that, we briefly explain the evaluation setup and our use of three traffic types to assess the precision of our approach. Additionally, we quantify the forwarding latency introduced by our traffic characterization enhanced switches compared to switches with only basic forwarding capabilities.

A. Evaluation Setup

For our experiments, we set up a network in Mininet 2.3.1ba[2], as it allows the emulation of real-world network conditions and provides a controlled platform for experimentation. In our virtualized environment, we strategically positioned three talkers interconnected to a virtual switch, all linked to a listener. A talker sends a flow of different traffic types: periodic, sporadic, or burst data transmissions. The virtual switch applies P4 programming tasks, especially ingress processing, facilitating network analysis, modification, and management. The listener is the traffic's destination, receiving and observing the forwarded traffic.

To compile our P4 program, we used p4c version 1.2.4.3, which generates target-specific configurations tailored to our network infrastructure. We utilized the Simple Switch Behavior Model version 2 (BMv2), a software-based switch offering a P4 Runtime interface for dynamic control over switch behaviour and configuration for testing and debugging purposes. BMv2 facilitates custom packet processing logic through its API and provides the Simple Switch CLI, a command-line interface for interacting with the switch runtime environment. This CLI enables us to observe runtime states, debug match-action tables, and monitor register changes effectively. By seamlessly communicating with the BMv2 runtime environment, the Simple Switch CLI is an intuitive interface and allows processes to interact with the switch. To evaluate our approach and effectively simulate real-world scenarios, we designed multiple scripts in Python 3.8.10 that generate three types of network traffic in our Mininet topology. These scripts use the Scapy library to construct and send TCP packets, allowing control over the generated traffic's frequency, intensity and duration. Additionally, we integrated a counter in the payload to identify packet loss and compute the switch's forwarding latency per packet.

We generate the following traffic types to evaluate our approach:

[2]https://mininet.org

- Periodic: A continuous stream of data packets transmitted regularly every 30 ms, with a consistent pattern and a random jitter between -0.5 and 0.5 ms to each transmission interval.
- Sporadic: Irregular transmission of packets with unpredictable gaps between packets. We create this type of traffic by initializing a random number generator seed and sending TCP packets within uniformly chosen times ranging from 1 to 100 ms.
- Burst: A series of packets are transmitted in rapid succession, followed by periods of inactivity. Our script randomly selects an idle period between 1 and 3 seconds, e.g., 2.64 seconds, followed by a burst of packets. The burst duration is between 3 and 8 seconds, and the burst size is randomly chosen between 10 and 20 packets.

Upon arrival at the switch's ingress port, incoming traffic undergoes packet header extraction to identify Ethernet frames and IPv4 packets for processing. Following packet processing, the switch executes an IPv4 longest prefix match (LPM) and forwards the packet to the listener. We initialize registers and counter in preparation for the subsequent packet arrival times storage, utilizing the *BMv2 simple_switch* ingress timestamp. This timestamp does not use network time protocol (NTP), providing independence from external NTP servers that could disrupt the experiment's timing measurements and introduce delays through network congestion. We calculate the inter-arrival times between packets and leverage read/write operations on the counter registers and the inter-arrival registers per talker until they are fully populated. Our approach, which involves retaining only a finite number of 500 recent inter-arrival times, implements a **sliding window** to conserve memory. Once the *window* is filled with inter-arrival times and corresponding autocorrelations are calculated, the traffic type is determined, and the registers are flushed for the next iteration.

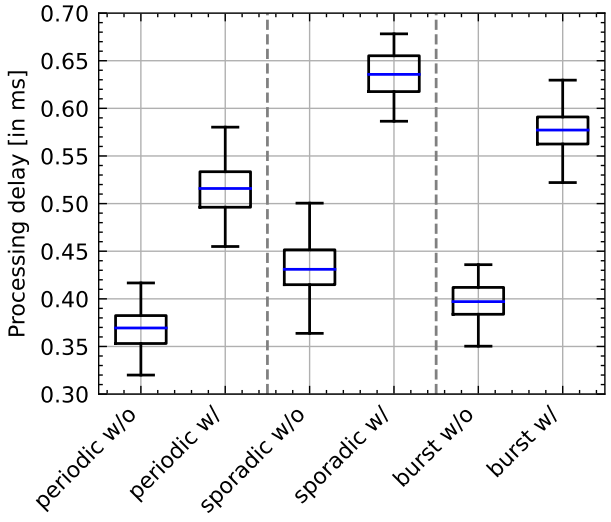


Fig. 5: Switch processing delay of our P4 traffic characterization (w/) compared to basic forwarding (w/o).

B. Evaluation Metrics

This section evaluates the performance of our approach as we use multiple metrics and compare our results to a switch with only basic forwarding capabilities.

1) Classification Accuracy: We set the register size to accommodate N=500 packets per flow for characterization purposes. Following this, we generated 200 periodic and non-periodic traffic flows for each. We subsequently compared the obtained characterization results to the actual sent traffic types. As outlined in the confusion matrix I and table II, our approach accurately identifies periodic and non-periodic traffic in 80% of the flows. Specifically, all 200 periodic flows were correctly identified as such by the switch (true positives). However, the switch incorrectly categorized 51 out of 200 non-periodic flows as periodic (false negatives). Furthermore, we determined a packet loss rate of 0%

TABLE I: Confusion matrix for periodic flow vs. non-periodic flow characterization.

Sent vs. Characterized	**Periodic**	**Non-Periodic**
Periodic (sent)	200 (TP)	0 (FN)
Non-Periodic (sent)	51 (FP)	149 (TN)

We additionally repeated the experiments with the same traffic types and parameters- only varying the register size N from 50 to 500 and observed the corresponding changes in precision. Table II summarizes the precision values obtained for each register size. As shown, precision values range from 0.7 to 0.8, with the highest precision achieved at a register size of 500. This indicates that our approach becomes more accurate in characterizing traffic types as the register size increases. Additionally, the recall values remained relatively stable across different register sizes, indicating consistent performance in correctly identifying periodic and non-periodic traffic.

TABLE II: Precision and Recall for varying register sizes.

Register Size N	**50**	**100**	**250**	**500**
Precision	0.7	0.67	0.73	0.8
Recall	0.74	0.67	0.73	1

2) Performance Measurement: We conducted a performance evaluation to assess the forwarding delays our traffic characterization system incurred on the switch. We first transmit periodic traffic to a virtual switch without traffic characterization functionality to establish a baseline. This switch solely executes basic traffic forwarding to the designated listener. Subsequently, we captured the traffic at the switch's ingress and egress ports and used tshark to match each incoming pcap packet entry with its corresponding outgoing pcap entry. For each matched pair, we calculate the forwarding delay by subtracting the ingress timestamp from the timestamp of the corresponding egress packet. This computation allows us to precisely quantify the time it takes for each packet to traverse the switch and undergo the basic forwarding process. Using identical traffic parameters (interval, duration and seed), we direct periodic, sporadic and burst traffic to our virtual

switch equipped **with** traffic-characterization capabilities and determine its forwarding delay. Figure 5 compares the forwarding delay of our traffic-characterization-enhanced switch and a switch with basic forwarding capabilities – both switches subjected to identical parameterized traffic types. The mean P4 traffic characterization forwarding delay per packet is 0.147 ms higher than in the basic forwarding switch. Additionally, our approach maintains exceptionally low delays, with the mean forwarding delay for non-periodic traffic at 0.193 milliseconds per packet. Given the substantial difference in lines of code and increased complexity of our P4 traffic characterization compared to basic forwarding, a higher delay is expected in all three traffic types.

We scrutinized our approach's RAM consumption in our evaluation, revealing a total utilization of approximately 0.014 MB. This includes allocations for critical components such as tables, header instances, and registers. Notably, tables with their action counter accounted for a significant portion, consuming 8192 bytes, while header instances required around 100 bytes. Additionally, our solution allocated 3000 bytes for the flow's autocorrelation register, 3060 bytes for its interarrival register, and 768 bytes for the overall counter register.

In summary, our approach demands a mere 0.014 MB of memory on the switch and incurs a maximum forwarding latency of 0.22 ms. These requirements fall well within the acceptable latency bounds in TSN networks [6]. Moreover, our approach introduces less latency compared to conventional SRP or SDN implementations [2]. Additionally, it eliminates the necessity of SDN message exchange, such as Talker advertise or flow setup requests between network participants.

C. Results

Figure 6 shows the aggregated inter-arrival times for the three different traffic classes in dependence on the number of processed packets. The x-axis represents the number of processed packets, while the y-axis denotes the inter-arrival times in milliseconds [ms]. Each dot in the figure represents the difference from the prior packet.

We set the maximum number of processed packets and register size to 500 for consistency across all traffic types.

Fig. 6a illustrates inter-arrival times in a periodic traffic scenario. The plot shows a distinct horizontal cluster, indicating regular packet arrivals within the periodic traffic stream at consistent intervals. In contrast, Fig. 6b depicts the P4-computed inter-arrival times of sporadic traffic, where there is no discernible pattern or shape; instead, data points across the entire range. The burst traffic plot in Fig. 6c illustrates a distinctive distribution of inter-arrival times. Here, most data points form a dense horizontal line near short inter-arrival times, representing packets arriving rapidly during burst periods. The scattered data points at higher inter-arrival times correspond to intervals between bursts.

Using the inter-arrival times, we computed the autocorrelation 1 as shown in Fig.7, in dependence on the lag. In Fig.7a, corresponding to the periodic traffic scenario, a dense concentration of data points is observed around low autocorrelation values, which underlines the strong correlation between adjacent inter-arrival times, as expected for periodic traffic. As the lag increases, the autocorrelation values display a greater variability, scattering towards higher values.

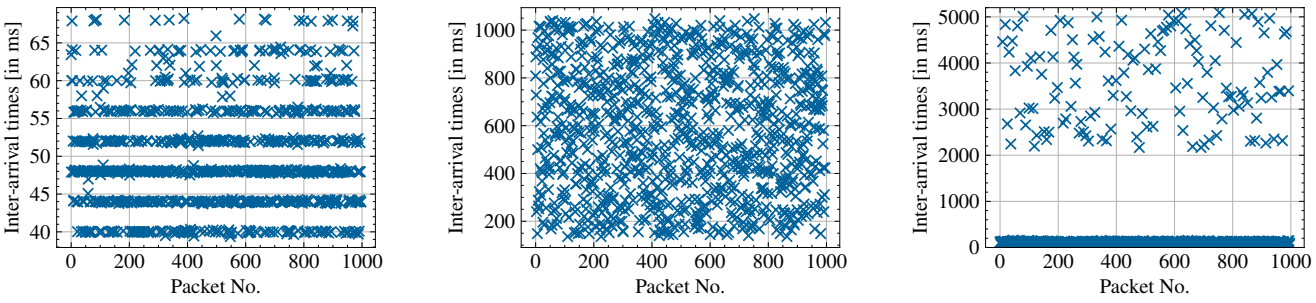
The autocorrelation plot for sporadic traffic, as depicted in Fig.7b presents a more scattered distribution, compared to the periodic traffic scenario. Despite the sporadic nature of the traffic, a significant number of packets arrive in relatively close succession, resulting in short inter-arrival times and, therefore, many low-lag data points. As the lag increases, the correlation between distant time intervals decreases, causing the data points to become more scattered. This illustrates the irregularity and unpredictability inherent in sporadic traffic.

Our results for the P4-computed autocorrelations of burst traffic are depicted in Fig. 7c. Burst traffic is characterized by the distinctive clustering pattern along low autocorrelation values spanning the entire range. As packets are transmitted rapidly during burst periods, the inter-arrival times are notably short, resulting in a pronounced autocorrelation at low lag values. Above the dense horizontal line of closely spaced inter-arrival times within burst periods, scattered data points exhibit a slight linear increase pattern as the lag extends. These data points correspond to inter-arrival times occurring between burst periods, characterized by a decrease in traffic intensity. Consequently, longer inter-arrival times are observed during these intervals.

Our results demonstrate the efficacy of our approach in enabling the switch to differentiate between various traffic types. Periodic traffic displays a consistent and predictable pattern, marked by significant correlations at low lag values, while scattering gradually increases with higher lags. In contrast, sporadic traffic exhibits irregular and unpredictable behaviour, characterized by highly scattered inter-arrival times. Burst traffic displays a unique pattern with dense clustering of inter-arrival times during burst periods, followed by scattered intervals between bursts.

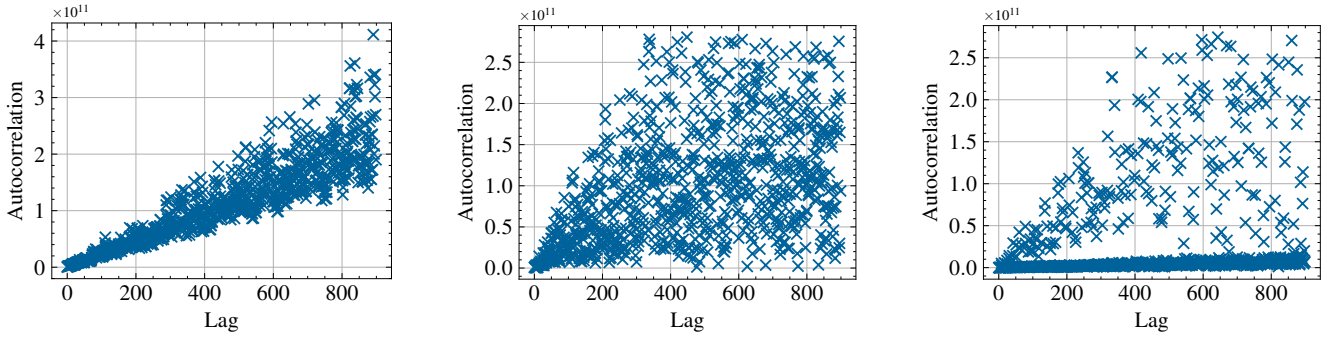
VI. Conclusion

This paper presents a novel paradigm: transparent TSN through local traffic characterization on the data plane. Our system allows network participants to reap the benefits of TSN, including deterministic and low-latency communication with QoS guarantees, without requiring TSN-awareness on the end-hosts. By shifting traffic analysis and management tasks from the TSN controller to the control plane, we significantly reduce communication overhead between network participants, effectively minimizing latency while ensuring adherence to QoS policies. We empower resource-constrained network components, specifically switches, to perform intelligent packet processing and statistical analysis autonomously, eliminating end-host dependencies for reservation requests. Our system leverages strategic usage of P4 registers to enforce stateful Ethernet packet processing. This enables precise determination of inter-arrival times and the calculation of autocorrelations for traffic type characterization. Performance measurements of



(a) periodic (b) sporadic (c) burst

Fig. 6: Inter-arrival times of three traffic types calculated by our P4-configured switch.



(a) periodic (b) sporadic (c) burst

Fig. 7: Autocorrelation of three traffic types calculated by our P4-configured switch.

our enhanced switch show a forwarding latency of 0.2 ms per periodic packet and an 80% precision rate for our approach. We eliminate the need for message exchanges between end-hosts and TSN controller, significantly reducing network load. Future work involves testing our implementation on hardware to assess its impact and performance in real-world scenarios. Additionally, further research could focus on comparing the performance and impact of our local traffic characterization in TSN networks to traditional TSN traffic management strategies.

References

[1] M. Seliem and D. Pesch, "Software-Defined Time Sensitive Networks (SD-TSN) for Industrial Automation," in *14th International Conference on Computational Intelligence and Communication Networks (CICN)*, 2022, pp. 1–7.

[2] T. Hackel, P. Meyer, F. Korf, and T. C. Schmidt, "Software-Defined Networks Supporting Time-Sensitive In-Vehicular Communication," in *IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*, 2019, pp. 1–5.

[3] G. N. Kumar, K. Katsalis, and P. Papadimitriou, "Coupling Source Routing with Time-Sensitive Networking," in *IFIP Networking Conference (Networking)*, 2020, pp. 797–802.

[4] H. Chahed and A. J. Kassler, "Software-Defined Time Sensitive Networks Configuration and Management," in *IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2021, pp. 124–128.

[5] I. T. Group, "IEEE Draft Standard for Local and metropolitan area networks–Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks Amendment: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements," *IEEE P802.1Qcc/D2.1*, pp. 1–236, 2018.

[6] N. Sertbaş Bülbül, D. Ergenç, and M. Fischer, "SDN-based Self-Configuration for Time-Sensitive IoT Networks," in *2021 IEEE 46th Conference on Local Computer Networks (LCN)*, 2021, pp. 73–80.

[7] S. Nam, H. Kim, and S.-G. Min, "Simplified Stream Reservation Protocol Over Software-Defined Networks for In-Vehicle Time-Sensitive Networking," *IEEE Access*, vol. 9, pp. 84 700–84 711, 2021.

[8] D. Bujosa, I. Álvarez, and J. Proenza, "CSRP: An Enhanced Protocol for Consistent Reservation of Resources in AVB/TSN," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 5, pp. 3640–3650, 2021.

[9] R. Sousa, P. Pedreiras, and P. Gonçalves, "Enabling IIoT IP backbones with real-time guarantees," in *IEEE 20th Conference on Emerging Technologies and Factory Automation (ETFA)*, 2015, pp. 1–6.

[10] M.-T. Thi, S. Ben Hadj Said, and M. Boc, "SDN-Based Management Solution for Time Synchronization in TSN Networks," in *25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2020, pp. 361–368.

[11] M. Gutiérrez, A. Ademaj, W. Steiner, R. Dobrin, and S. Punnekkat, "Self-configuration of IEEE 802.1 TSN networks," in *22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2017, pp. 1–8.

[12] M. Gutiérrez, W. Steiner, R. Dobrin, and S. Punnekkat, "A configuration agent based on the time-triggered paradigm for real-time networks," in *IEEE World Conference on Factory Communication Systems (WFCS)*, 2015, pp. 1–4.

[13] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.