Digital Object Identifier 10.1109/TNSM.2024.3434337

P4-PSFP: P4-Based Per-Stream Filtering and Policing for Time-Sensitive Networking

Fabian Ihle, Steffen Lindner, and Michael Menth
University of Tuebingen, Chair of Communication Networks, 72076 Tuebingen, Germany
Email: {fabian.ihle, steffen.lindner, menth}@uni-tuebingen.de

***Abstract*—Time-Sensitive Networking (TSN) extends Ethernet to enable real-time communication. In TSN, bounded latency and zero congestion-based packet loss are achieved through mechanisms such as the Credit-Based Shaper (CBS) for bandwidth shaping and the Time-Aware Shaper (TAS) for traffic scheduling. Generally, TSN requires streams to be explicitly admitted before being transmitted. To ensure that admitted traffic conforms with the traffic descriptors indicated for admission control, Per-Stream Filtering and Policing (PSFP) has been defined. For credit-based metering, well-known token bucket policers are applied. However, time-based metering requires time-dependent switch behavior and time synchronization with sub-microsecond precision. While TSN-capable switches support various TSN traffic shaping mechanisms, a full implementation of PSFP is still not available. To bridge this gap, we present a P4-based implementation of PSFP on a 100 Gb/s per port hardware switch. We explain the most interesting aspects of the PSFP implementation whose code is available on GitHub[1]. We demonstrate credit-based and time-based policing and synchronization capabilities to validate the functionality and effectiveness of P4-PSFP. The implementation scales up to 35840 streams depending on the stream identification method. P4-PSFP can be used in practice as long as appropriate TSN switches lack this function. Moreover, its implementation may be helpful for other P4-based hardware implementations that require time synchronization.**

***Index Terms*—Software-Defined Networking, Time-Sensitive Networking, P4, Per-Stream Filtering and Policing, Data Plane Programming, Resilience**

I. Introduction

Ethernet-based communication has found its way into industrial applications such as Industry 4.0, the Internet of Things, and in-vehicle networks. In this environment, time-critical communication is prevalent. Failure to meet the strict real-time requirements of mission-critical systems can lead to immediate degradation of factory performance, endanger factory personnel, or put traffic participants at risk. However, communication in Ethernet networks using mechanisms such as best-effort transmission, traffic classification, and VLANs does not provide quality of service (QoS) that meets real-time requirements [1], [2].

Time-Sensitive Networking (TSN) is a collection of standards for enhancing Ethernet communication to meet the stringent real-time requirements of industrial and in-vehicle applications. The collection provides mechanisms to achieve zero congestion-based packet loss and a deterministic upper bound on latency. To that end, senders, so-called talkers, signal their QoS requirements and sending behavior before they start their transmission. Bridges make resource reservations for approximately periodic flows, called streams, initiated by talkers, e.g., via the stream reservation protocol in IEEE Std 802.1Qat [3]. This is referred to as admission control. Many traffic shaping mechanisms exist in the TSN standards to ensure that the previously signaled QoS requirements are met, e.g., the Credit-Based Shaper (CBS) in IEEE Std 802.1Qav [4] or the Time-Aware Shaper (TAS) in IEEE Std 802.1Qbv [5]. However, in pure TSN, no entity enforces a talker's compliance with its announced sending behavior. Malicious or misconfigured participants can therefore consume more resources than announced for admission control. As a result, the network may fail to meet its guarantees of a deterministic upper bound on latency and zero congestion-based packet loss. PSFP [6] defined in IEEE Std 802.1Qci [6] is a mechanism that addresses this problem by limiting or excluding out-of-profile traffic. PSFP consists of three components, called stream filters, stream gates, and flow meters. Stream filters identify and filter streams, stream gates monitor the transmission time, and flow meters monitor the bandwidth. Although there are some TSN-capable switches that support frame prioritization, CBS, and TAS, to the best of our knowledge, there are no switches that support the PSFP mechanism in a comprehensive way and in compliance with IEEE Std 802.1Qci [6]. With the emergence of data plane programmability, users can specify the behavior of switches by themselves allowing for mechanisms, such as PSFP, to be implemented on their own hardware.

The contribution of this paper is as follows. First, we develop P4-PSFP, a novel implementation of the IEEE Std 802.1Qci PSFP mechanism on real hardware in the P4 programming language. The implementation covers all PSFP components, i.e., stream filter, stream gate, and flow meter. Second, we leverage the capabilities of a hardware-based switching ASIC, specifically an Intel Tofino™, to achieve line rate processing of $100\,\mathrm{Gb/s}$ per port and nanosecond precision for time-based operations. We successfully address challenges when using real hardware, such as the periodicity in time-based metering. Furthermore, we propose an approach to account for time inaccuracy in implementations where the application of time synchronization protocols such as the Precision Time Protocol (PTP) is not possible. This approach includes a mechanism to synchronize time-critical stream gates

The authors acknowledge the funding by the Deutsche Forschungsgemeinschaft (DFG) under grant ME2727/3-1. The authors alone are responsible for the content of the paper.

[1]https://github.com/uni-tue-kn/P4-PSFP

with the network time and to account for clock drift.

We conduct comprehensive evaluations regarding the functionality of the implemented PSFP components, i.e., credit-based and time-based policing. Furthermore, we demonstrate the effectiveness of the implementation in protecting a TAS schedule by aiming to eliminate queueing in a congested network environment. Finally, we evaluate the scalability of the implementation regarding the number of supported streams.

The remaining paper is structured as follows. In Section II we provide background information on TSN and review related work. Section III explains the basic concept of the PSFP mechanism followed by an introduction to the P4 programming language in Section IV. The implementation of the PSFP mechanism in P4 is explained in Section V. The implementation is evaluated in Section VI. Finally, we discuss requirements of P4-PSFP and its applicability beyond industrial networks in Section VII and conclude the paper in Section VIII.

II. Background of TSN and Related Work

In this section, we first give background information on TSN. We then review related work.

A. Time-Sensitive Networking (TSN)

TSN is a collection of standards for enhancing Ethernet communication that guarantees QoS requirements, i.e., a deterministic upper bound on latency and zero congestion-based packet loss. It is currently being standardized by the IEEE 802.1 TSN Task Group. A TSN network consists of several bridges and end stations. A sending end station is a talker, a receiving end station is a listener.

A data flow initiated by a talker is called a TSN stream and must first be admitted by the network before it can be transmitted. Admission control implies that a talker signals the properties of a new stream to be admitted, e.g., the frame size and QoS requirements, to the network. We call the description of these properties a stream descriptor. The talker agrees to adhere to its signaled stream properties so that the network can rely on this specification for resource management purposes. Based on this information, the network accepts the request and reserves resources for the new stream, or it rejects the new stream for high-priority transmission. TSN streams may coexist with best-effort transmissions which do not require admission control.

A TSN network can use different methods to guarantee latency bounds, such as the Credit-Based Shaper (CBS) in IEEE Std 802.1Qav [4] that uses a token bucket mechanism, and the Time-Aware Shaper (TAS) introduced in IEEE Std 802.1Qbv [5]. The TAS divides the communication into repeating cycles composed of time slices, similar to a Time Division Multiple Access (TDMA)-based approach [1], but for packet-switched communication networks.

Bridges in IEEE Std 802.1Q [7] place frames in a queue according to the priority in the Priority Code Point (PCP) field in the VLAN tag of the frame. Based on the assigned queue, TSN traffic shaping mechanisms can be applied to the frame. By using an IP stream identification function in combination with an active VLAN stream identification from IEEE Std 802.1CB [8], a VLAN tag can be pushed onto frames that match a particular stream. This makes untagged frames eligible for traffic shaping in TSN bridges.

There can be up to eight different priorities in IEEE Std 802.1Q. Within a TSN bridge, each egress port can accommodate up to eight egress queues. Each queue is associated with one of the eight frame priorities specified in the VLAN header of a frame.

In TSN, high-priority streams may be scheduled, i.e., their sending times at talkers are coordinated in such a way that their frames experience hardly any delay in bridges. Scheduled streams are protected from other traffic by the Time-Aware Shaper (TAS). The TAS defines a gating mechanism for each egress queue on an egress port. It can be used to provide transmission resources exclusively to scheduled traffic, which supports ultra-low latency. The gating mechanism defines time slices in which only configured egress queues can send traffic. This is referred to as an egress queue with a gate in a closed or an open state within a time slice. Frames in a queue with an open gate are selected for transmission in a FIFO manner while frames in a queue with a closed gate are not selected and remain in the queue. The time slices and gates of the queues are configured in the so-called gate control list (GCL). It consists of several time slices, each associated with an 8-bit vector indicating the queues with an open gate. This GCL is processed repeatedly, i.e., time slices and gates are periodic. For stream scheduling, the periodic transmission times at talkers are calculated offline together with the GCLs of all bridges. The result is called a schedule [5], [9].

Scheduled streams are sensitive to errors such as a lack of time synchronization between talkers and bridges. If frames are transmitted by non-synchronized talkers outside their assigned time slice, they possibly consume the resources reserved for other streams. As a result, these streams and possibly also others may not meet their latency bounds.

Thus, TSN talkers, listeners, and bridges must be time-synchronized so that all participants in a TSN network share a common understanding of time. High-precision time synchronization is essential for talkers to ensure that they transmit their data at the precisely designated times. At the same time, bridges require accurate time synchronization to determine the arrival time of an incoming frame and to assign it to the correct time slice in the GCL. In TSN networks, protocols such as PTP [10] are used which achieve a precision in the order of tens of nanoseconds [11].

B. Related Work

We first discuss related work on PSFP and similar mechanisms, such as TDMA, in various applications and review an implementation of a stream reservation enforcement mechanism in P4. We then review a hardware traffic generator implemented in P4 called P4TG that is used in Section VI.

1) PSFP in Different Applications: Meyer *et al.* [12] analyze PSFP as a security mechanism to identify misbehaving traffic streams in cars. They show that PSFP reliably detects attacks on car communication in their simulated OMNeT++

[13] environment in several scenarios. Similarly, Luo *et al.* [14] propose to use PSFP for the detection of anomalies caused by DoS attacks and abnormal traffic behavior. Their evaluations indicate that PSFP ensures real-time performance in TSN networks and successfully blocks DoS attacks. Their implementation is based on the OMNeT++ simulation framework and is not implemented on real hardware.

Nsaibi *et al.* [15] provide a migration from the industry standard Sercos III, a TDMA-based real-time Ethernet protocol [16] with a line topology, to TSN networks with a tree topology by extending Sercos III with the TSN standards IEEE Std 802.1AS-Rev [17] and IEEE Std 802.1Qbv [5]. They used a $1\,\mathrm{Gb/s}$ TSN switch for this purpose. TDMA as used in Sercos III is a mechanism very similar to stream gates in PSFP where frames are transmitted only during specific time slices. They conclude that TSN brings many benefits for Sercos III networks, such as multi-gigabit data rate and the support for different network topologies.

Szancer *et al.* [18] aim to improve latency and jitter in TSN-migrated Sercos III networks by integrating not only a single $1\,\mathrm{Gb/s}$ TSN switch, but by enabling TSN for every device in the network. They tested their implementation with a TDMA-based and a CBS-based approach. Their simulation shows that a TDMA-based approach has a significantly lower jitter. Since PSFP essentially consists of both TDMA-based and CBS-based approaches, a Sercos III network migrated to a TSN network could benefit from the additional security features of PSFP.

Bülbül *et al.* implemented a mechanism called TSN gatekeeper to enforce stream reservations in P4 [19]. Their implementation contains elements of the PSFP mechanism, such as stream filtering, stream gates, and flow metering. However, their stream gate implementation remains continuously in the open state, i.e., time-based metering is not applied. Multiple streams are assigned to a single stream gate and only the aggregated bandwidth of these streams is monitored. The transmission times of TSN talkers are not monitored. Unlike the IEEE 802.1Qci standard, their implementation does not use the stream identification functions defined in IEEE Std 802.1CB [8]. They implemented the TSN gatekeeper mechanism in P4 on the BMv2 [20] software target platform and emulated their environment in Mininet. However, BMv2 implementations are not an indicator whether algorithms will also work on real hardware.

In contrast, the implementation of P4-PSFP in this work supports credit-based metering, cyclic schedules, i.e., time-based metering, an approach for synchronizing the implementation with the network time and accounting for clock drift, multiple stream identification functions, and is implemented on the Intel Tofino™ switching ASIC hardware target platform which offers high-performance packet processing.

2) Deterministic Networking (DetNet): TSN defines mechanisms to achieve bounded latency and zero packet loss at the link layer over standard Ethernet. Deterministic Networking (DetNet) is a similar approach using explicit bandwidth reservation, priority queueing, and service protection at the network layer. DetNet is currently being standardized by the IETF. In RFC 9023 [21], the authors describe an approach for operating a DetNet data plane with IP over a TSN sub-network. They state that the efforts in TSN to achieve bounded latency and zero packet loss are likely compatible with DetNet networks. RFC 9037 [22] describes a similar approach carrying traffic over DetNet-capable MPLS nodes interconnected by a TSN subnet. In this approach, the stream ID in the TSN subnet is derived from the MPLS flow parameters. Therefore, service protection mechanisms such as PSFP can be applied to MPLS flows carried over the TSN subnet.

Choi *et al.* [23] describe an implementation of a DetNet packet forwarding engine operating at $100\,\mathrm{Gb/s}$. They can guarantee a bounded latency with less than $6.25\,\mu\mathrm{s}$ per node. Addanki *et al.* [24] propose a mechanism called Time Aware PIFO Scheduling (TAPS) to address the problem of misbehaving flows in DetNet which is also a problem in TSN networks. The authors of [25], [26], and [27] investigate TSN and DetNet in the context of 5G networks. These networks have ultra-low latency requirements, e.g., to enable industrial automation.

3) Traffic Generation: P4TG is a P4-based traffic generator for Ethernet/IP networks introduced by Lindner *et al.* [28] that runs on an Intel Tofino™ switching ASIC. It is capable of generating both constant bit-rate (CBR) traffic and Poisson traffic at $100\,\mathrm{Gb/s}$ per port with more stable inter-arrival times (IATs) than other traffic generators. The implementation of Lindner *et al.* does not suffer from the limitations of software-based traffic generators running on general-purpose CPUs, i.e., low data rates and being prone to significant fluctuations in traffic generation [29] while being significantly less expensive than hardware-based commercial solutions. The generated traffic can be fed back from the output ports to the input ports through other equipment. In this process, P4TG measures rates, frame types, frame sizes, and packet loss directly in the data plane, and samples RTTs and IATs in the control plane. We leverage P4TG in Section VI for performance evaluation.

III. Per-Stream Filtering and Policing (PSFP)

This section explains PSFP as defined in IEEE Std 802.1Qci [6]. First, an overview of PSFP is given. Then the PSFP components, i.e., stream filter, stream gate, and flow meter, are explained.

A. Overview

PSFP is a mechanism to ensure that streams of participants in a TSN network adhere to the stream descriptors indicated during admission control. Streams that do not adhere to the stream descriptors can be blocked entirely in a PSFP-enabled bridge or have individual violating frames dropped. The PSFP mechanism as defined in IEEE Std 802.1Qci [6] is intended for Ethernet bridges and consists of the three components stream filter, stream gate, and flow meter, that are processed in sequential order. An overview is given in Figure 1.

An incoming frame is first identified by the stream filter instance. The identified stream is then assigned to a stream gate instance and a flow meter instance. The stream gate decides whether to forward or drop the frame depending on its arrival time, i.e., whether it adheres to a cyclic schedule. Afterwards, the flow meter checks if the frame is in-profile in terms of bandwidth.

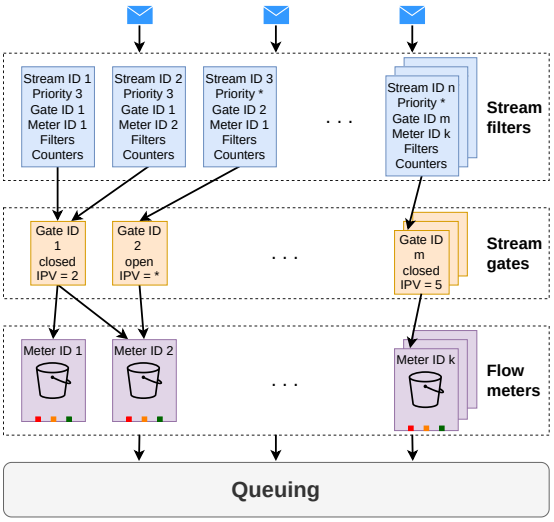


Fig. 1. The three components stream filter, stream gate, and flow meter compose the PSFP mechanism. They are processed in sequential order [6].

B. PSFP Components

This section describes the purpose and functionality of each of the three PSFP components and summarizes the conditions that must be met for a frame to be forwarded.

1) Stream Filter: The stream filter instance implements several stream identification functions according to IEEE Std 802.1CB [8]. A stream identification function maps frames of a particular stream to a parameter called stream handle by identifying the stream according to header fields of a frame. Stream identification functions in IEEE Std 802.1CB match, among others, the source MAC address and the VLAN ID, or IP header fields. A stream gate and a flow meter instance are assigned to a frame based on the stream handle. If a frame is identified by a stream identification function, it is forwarded to the assigned stream gate instance. Otherwise, PSFP is not applied to the frame and it is queued with no further action. For a frame that is not processed in PSFP, the queue is selected according to the frame priority in the PCP field as defined in IEEE Std 802.1Qci [6]. Optional filtering functions can be defined, e.g., a maximum frame size filter that drops frames which exceed a certain frame size. In addition, the stream filter has an additional parameter called *MaxSDUExceeded* which can permanently block a stream if a frame exceeds the configured maximum frame size once. This prevents misbehaving participants from communicating in the TSN network.

2) Stream Gate: A stream gate monitors the arrival time of frames and ensures that frames are only transmitted during their allowed time slices.

A stream gate instance is controlled by a stream gate control list (GCL). The stream GCL in PSFP is similar to the GCL of the TAS, but they are not the same. Both, the stream GCL and the TAS GCL are composed of cyclically repeated time slices associated with a gate state that can be either open or closed. In contrast to the TAS GCL, the stream GCL is independent of the egress port and its priority queues. It does not contain the 8-bit vector to control queue-specific gates. Instead, the stream GCL only encodes the time slices of a single stream gate instance which corresponds to one or more identified streams. Frames arriving at the switch during a time slice in an open state of the stream GCL are forwarded, frames arriving in the closed state are dropped. In contrast to the TAS, the PSFP stream gate cannot delay frames, it can only drop them. Additionally, time slices in a stream GCL have an optional value called Internal Priority Value (IPV). This value is used to select the egress queue for a frame in this bridge. If the IPV is not set, the PCP value from the 802.1Q header is used.

Stream GCLs must be carefully planned in advance. They are periodic to facilitate continuous operation. The period p_i of a stream GCL g_i is the duration of all individual time slices in the stream GCL g_i. Multiple stream GCLs are usually aggregated into a hyperperiod h which equals the least common multiple (LCM) of all periods. This is needed to model the cyclic behavior [30]–[32] and is illustrated, together with a stream GCL, in Figure 2.

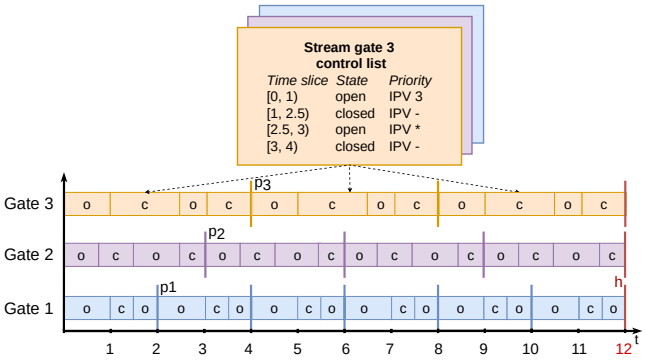


Fig. 2. Example hyperperiod consisting of three stream GCLs with periods $p_1 = 2$, $p_2 = 3$, and $p_3 = 4$, and time slices in the open (o) or closed (c) state. Each stream GCL is extended to the least common multiple of $h = 12$ to form a hyperperiod.

The shown hyperperiod contains three stream gate instances, each with its own stream GCL and its own period. Each stream GCL is extended to the LCM of 12 time steps.

Furthermore, a stream gate holds additional parameters to permanently close the gate if it receives a single frame during a time slice in a closed state (*GateClosedDueToInvalidRX*), or if it receives too much data in a single time slice (*GateClosedDueToOctetsExceeded*).

3) Flow Meter: The flow meter instance monitors the compliance of TSN talkers with their admitted rate in their stream descriptor. A two-rate, three-color marking token bucket policer conforming to RFC 2698 [33] is used for this purpose. The token bucket policer is illustrated in Figure 3.

In the two-rate three-color marking token bucket algorithm, two buckets C and E are filled with tokens at the Committed Information Rate (CIR) and Excess Information Rate (EIR) respectively. The two buckets have a capacity of Committed Burst Size (CBS) tokens, and Excess Burst Size (EBS) tokens. The CIR corresponds to the guaranteed bandwidth, and the EIR represents the additional bandwidth that can be used temporarily if available. A token bucket is initially completely filled. An incoming frame consumes tokens from the buckets according to its frame size. If there are enough tokens in bucket

TABLE I
SUMMARY OF AVAILABLE OPTIONAL PARAMETERS FOR PSFP COMPONENTS IN IEEE STD 802.1QCI [6]. THEY ARE ALL IMPLEMENTED IN P4-PSFP.

Parameter name	PSFP component	Description	Implemented in P4-PSFP
MaxSDUExceeded	Stream filter	The stream is permanently blocked if a single frame exceeds the configured frame size.	✓
GateClosedDueToInvalidRX	Stream gate	The gate is permanently closed if a single frame arrives during a time slice in a closed state.	✓
GateClosedDueToOctetsExceeded	Stream gate	The gate is permanently closed if too many bytes are sent during a single time slice.	✓
DropOnYellow	Flow meter	Yellow-labeled frames, either pre-colored or colored by the flow meter, are marked red and dropped.	✓
MarkAllFramesRed	Flow meter	The flow meter is permanently blocked if a single frame is marked red.	✓
ColorMode	Flow meter	The color mode indicates whether the pre-color of a frame should be respected.	✓

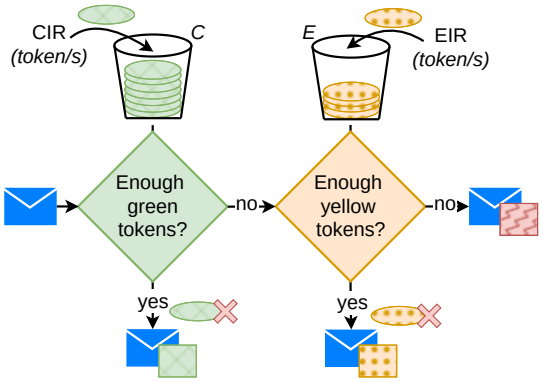


Fig. 3. A two-rate, three-color marking token bucket policer according to RFC 2698 [33].

C, the frame is marked green and the tokens are removed from the bucket. If there are not enough tokens in bucket C but there are enough in bucket E, the frame is colored yellow and consumes tokens from bucket E. Otherwise, the frame is marked red. In RFC 2698, the color of a frame is reflected in the DS field of the IP header. However, this concept can also be used for non-IP traffic by using other header fields to reflect the color.

The RFC only describes how to color frames, but does not associate a color with an action. In PSFP, green-colored frames are forwarded. For yellow frames, the DropEligibleIndicator (DEI) flag[2] is set in the 802.1Q header before they are forwarded, and red frames are dropped [6].

Additional parameters are added to the flow meter instance, e.g., the *DropOnYellow* parameter, which additionally drops yellow frames, or the *MarkAllFramesRed* parameter, which permanently blocks the flow meter once the assured bandwidth has been exceeded. The *ColorMode* parameter indicates whether or not a frame pre-color should be respected.

4) Summary of PSFP Parameters: The optional parameters available for PSFP components are summarized in Table I. A frame that is identified by a stream identification function is assigned to a stream gate instance and a flow meter instance. The frame must meet all of the following conditions to be forwarded.

1) The frame size is below the maximum frame size.
2) The identified stream is not permanently blocked due to a previous frame exceeding the maximum frame size if the MaxSDUExceeded parameter is enabled.
3) The frame is assigned to a time slice of the stream GCL in an open state.
4) The assigned stream gate is not permanently closed due to a previous frame if the GateClosedDueToInvalidRX parameter is enabled.
5) The number of bytes transmitted in the current time slice of the stream GCL does not exceed the maximum configured number of bytes per time slice if the GateClosedDueToOctectsExceeded parameter is enabled.
6) The frame is not marked red by the flow meter.
7) The assigned flow meter is not permanently blocked due to a previous frame if the MarkAllFramesRed parameter is enabled.
8) The frame is neither pre-colored yellow[3] nor marked yellow by the flow meter if the DropOnYellow parameter is enabled.

IV. INTRODUCTION TO P4

This section provides fundamentals of the P4 programming language. We start with an overview of P4. Then we explain match+action tables (MATs). Finally, we explain the P4 pipeline in the context of the Tofino Native Architecture (TNA).

A. Overview

Programming Protocol-independent Packet Processors (P4) is a high-level programming language for describing a data plane and an architecture-based programming model. The P4 language can be used to manipulate packets and make forwarding decisions in the data plane to implement user-defined

[2]The DEI flag indicates frames that may be dropped in the presence of congestion.

[3]A frame is pre-colored yellow if it has the DEI flag already set upon entering the switch.

algorithms. Devices that implement a specific architecture are called targets. Targets can be either software-based, e.g., the `simple_switch` as used in the BMv2 [20], or hardware-based, e.g., the Intel Tofino™ switching ASIC [34]. A P4 program is designed for a specific architecture and may be used with all targets that implement this specific architecture [35]. Externs extend the functionality of a P4 program with target-specific operations such as registers or counters. The actual implementation of the extern functionality depends on the target. One hardware-based architecture is the TNA which will be discussed further in this section. It builds upon the more basic Portable Switch Architecture (PSA) [36].

Implementing a complex, time-dependent algorithm on real hardware poses many challenges. To meet the line rate of the switch, hardware targets often have limitations on the number of operations that can be applied per packet, so writing a P4 program for them is more difficult than for software targets. Implementations on software switches or simulation frameworks, like in CoRE4INET [37], do not suffer from these constraints and cannot guarantee line rate. Therefore they are not suitable for a real-world setting. Further information on P4 can be found in a detailed survey by Hauser et al. [38].

B. Match+Action Tables (MATs)

A core feature of P4 programs are match+action tables (MATs). They describe tables that associate key fields, i.e., header or metadata fields, with user-defined actions. Metadata in P4 can be intrinsic or user-defined. Intrinsic metadata contains data provided by the architecture, e.g., the ingress port of a frame. The concept of MATs is illustrated in Figure 4.

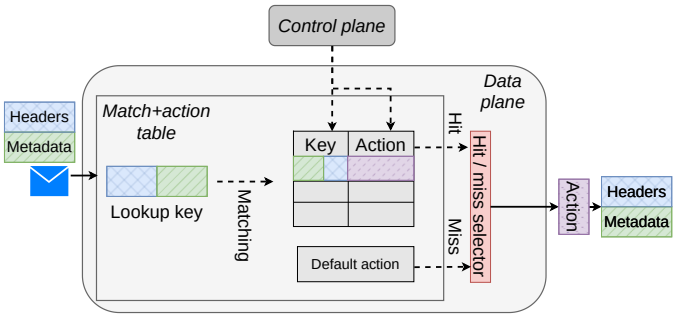


Fig. 4. Illustrated match+action table (MAT) [38]. The header and metadata fields of a frame form a composite key to match entries in the table. In the data plane, either the assigned action is executed on a match, or the default action is executed on a miss. The table entries are populated by the control plane.

An incoming packet is matched against the key fields in a MAT and an associated action is executed upon matching an entry. Actions can be described as small code fragments and may affect packet forwarding or manipulate fields. MATs and actions are defined in the data plane while the entries of a MAT and the parameters of an action must be filled in by the control plane. The core P4 language defines three match types for MATs: ternary, longest-prefix-match (LPM), and exact. A ternary match applies a predefined bitmask to the key fields. An LPM match type implements longest prefix matching which is well known from IPv4. The exact match type can be seen as a special case of LPM with the maximum prefix length.

C. Tofino Native Architecture (TNA)

In general a P4 pipeline consists of a programmable packet parser and control blocks, followed by a packet deparser. The packet parser extracts header information from packets and stores the information in an adequate data structure. Control blocks define packet processing operations and describe the algorithm to be applied to packets [39]. P4 control blocks may use branching constructs as well as logical and simple arithmetic expressions. Loop constructs cannot be used in P4. Recirculation is a mechanism that models loop behavior by sending a packet from the output port back to the input port where it traverses the pipeline again.

In particular, the pipeline of the TNA, the architecture used by the Intel Tofino™, consists of both ingress control blocks and egress control blocks [40], each of which has its own parser and deparser. The TNA pipeline is illustrated in Figure 5.

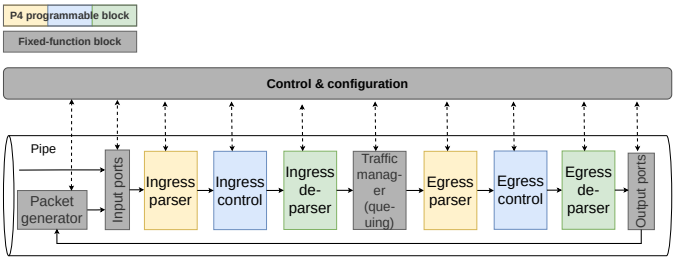


Fig. 5. Illustrated TNA pipeline [40]. It consists of several P4 programmable blocks in the ingress and egress and some fixed-function blocks, such as the packet generator.

Parsers, deparsers, and both control blocks are programmable in P4 and can be used to control packet forwarding and manipulation. A TNA-specific packet generator can be configured to assemble packets and send them through the Intel Tofino™ output ports, e.g., in a periodic manner. Furthermore, the TNA defines an additional type for matching in MATs: range. This type can be used to match a key field to an interval range and execute the action accordingly.

V. P4 Implementation of PSFP for Intel Tofino™

In this section, we give an overview of the implementation of PSFP in P4 on an Intel Tofino™ switching ASIC integrated within an Edgecore Wedge 100BF-32X [34] switch. P4 language-specific challenges such as the periodicity of stream GCLs are addressed and solutions are presented. An approach to account for time inaccuracy in implementations without time synchronization capabilities in the data plane is proposed. The source code is publicly available on GitHub [41].

A. Overview

A high-level description of the P4-PSFP implementation is given in Figure 6. A frame entering the P4-PSFP bridge is matched against a MAT in the stream filter component.

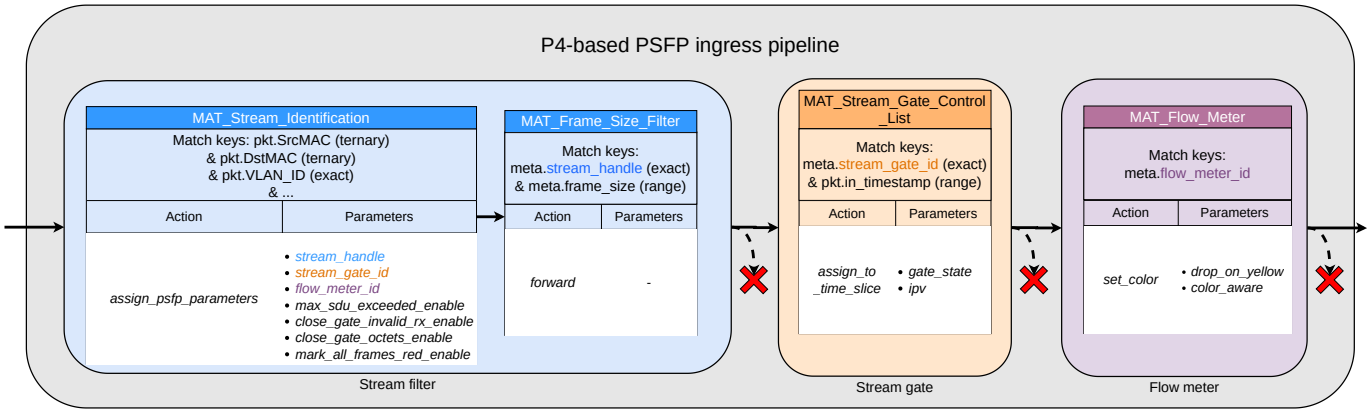


Fig. 6. Overview of the P4-based PSFP ingress pipeline. A frame is identified by the stream filter via a MAT, filtered according to its frame size, assigned to a stream gate time slice by its ingress timestamp, and finally flow metered. Any of the components may drop the frame.

For simplicity, we only consider VLAN-tagged frames in our implementation. The key in the stream filter MAT consists of selected header fields to identify the stream, e.g., the VLAN ID. The header fields used for this purpose depend on the used stream identification function. If the frame is identified by the stream filter, it belongs to a time-sensitive stream and is eligible for PSFP processing. As a result, the stream gate instance and flow meter instance are assigned to the frame using metadata. Furthermore, the frame size is checked in an additional MAT.

In the next step, the frame is matched depending on the frame ingress timestamp against a MAT in the stream gate instance. This MAT represents the stream GCL and a frame is assigned to a time slice. The associated action retrieves the gate state in the current time slice and assigns the IPV to the frame. Each entry in the stream gate instance MAT corresponds to a time slice in the open state of a stream GCL. Time slices in the closed state do not need to be listed in the stream gate instance MAT. A gap between two open time slices in the stream gate instance MAT implicitly models closed time slices. Packets arriving in such a gap do not match in the MAT, resulting in a miss. On a miss in the MAT, the packet is dropped by the default action. If the frame was assigned to a time slice in an open state, it is further matched against a MAT with its previously assigned flow meter ID in the flow meter instance.

In the flow meter instance, a token bucket algorithm according to RFC2698 [33] is applied to the frame. To that end, a P4-specific meter extern is assigned to the flow meter MAT. This meter extern can be configured with the appropriate rates for the token bucket algorithm. Depending on the frame color, the frame is then dropped, marked, or forwarded. Any of the components may drop a frame if it violates conditions announced during admission control, i.e., it exceeds a certain frame size, arrives in a closed state time slice, or exceeds the admitted bandwidth.

B. Details of the PSFP Implementation

This section gives insights into the details of the implementation. First, some challenges and their solutions are elaborated. Subsequently, the interaction of all implemented mechanisms is explained.

1) Maximum Frame Size Filter: The stream filter component is designed to optionally discard frames that exceed a configured maximum frame size. The TNA provides the frame size in the intrinsic egress metadata for further processing. Since PSFP has the ability to modify frame priorities and thus their queue, the PSFP mechanism must be applied prior to the queuing process. In the TNA, queuing takes place subsequent to the ingress control block but preceding the egress control block, as illustrated in Figure 5. As a result, the implementation of PSFP must reside within the ingress control block. However, the frame size is only available after egress parsing in the TNA [40], but is needed for filtering in the ingress control block in the stream filter.

We use the concept of recirculation in P4 to solve this problem. A frame that is eligible for PSFP processing traverses the P4 pipeline twice. On the first pass, the frame gets a recirculation header prepended to its header stack in the egress control block. This header contains the frame size available in the egress control block. The frame is then recirculated and sent back to an ingress port. After recirculation, the frame size in the recirculation header can be used to apply the stream filter in the ingress control block. Packet processing in the Intel Tofino™ takes a constant amount of time on sub-microsecond scale, so a recirculation only adds a constant, known amount of time to the processing delay. The effect of recirculation in P4-PSFP is discussed in Section VII-A.

2) Periodicity of Stream GCLs: This section focuses on the implementation of configured stream GCLs and achieving periodicity. Finding a valid schedule and scheduling algorithms for a TSN environment are outside the scope of this work. Further information on scheduling algorithms in TSN networks can be found in a detailed survey by Stüber *et al.* [42].

A PSFP stream GCL is defined by time slices with relative time steps that are periodically repeated. The stream GCL must be able to operate continuously for an indefinite amount of time. To that end, the absolute timestamp of an incoming frame must be mapped to the correct relative time slice in the stream GCL. This is illustrated in Figure 7.

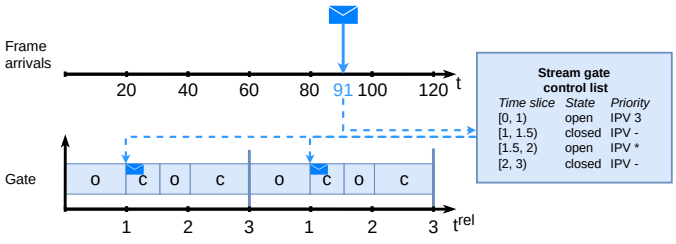


Fig. 7. A frame received at an arbitrary absolute timestamp, e.g., at $t_i = 91$, is mapped to its relative position in the stream GCL, e.g., to the time slice $[1, 1.5)$.

Multiple stream GCLs are aggregated to a hyperperiod to model cyclic behavior as described in Section III-B2. The i-th frame is received at time t_i. For further processing, i.e., matching a frame to a time slice in the stream gate instance, a mapping to a relative frame arrival time t_i^{rel} in a hyperperiod with the duration h is required. Let t_0^h be the start time of the first hyperperiod. Then its relative arrival time can be determined as shown in Equation 1.

$$t_i^{rel} = (t_i - t_0^h) \bmod h \tag{1}$$

The calculation in Equation 1 cannot be implemented on the Intel Tofino™ due to the lack of a modulo operator.

We leverage the internal packet generator of the TNA to imitate the modulo operator. The packet generator is configured to periodically generate a packet j every h time steps and send it to an ingress port of the P4-PSFP switch. On reception of such a generated packet, its ingress timestamp t_j^h is stored in a data plane register r^h that references the last completed hyperperiod. The relative position t_i^{rel} of a frame i within a hyperperiod of the duration h can then be calculated as described in Equation 2.

$$t_i^{rel} = t_i - t_j^h \tag{2}$$

Equation 2 is semantically equivalent to the modulo operation in Equation 1 with the requirement that the timestamp of the generated hyperperiod packet arrives perfectly after an elapsed hyperperiod, i.e., it holds that $t_j^h = t_1^h + (j-1) \cdot h$. Since this may not always be the case, we introduce a mechanism to account for such inaccuracies in Section V-C.

The Intel Tofino™ uses integers to represent timestamps with a granularity of 1 ns [40]. Moreover, timestamps on the Intel Tofino™ are limited to 48 bits. Thus, they overflow after $t^{max} = 2^{48} - 1$ ns ≈ 3.25 days. This leads to an underflow in the calculation of the relative frame arrival time. Figure 8 illustrates the problem of underflows in the calculation of the relative timestamp if t^{max} is reached.

In Figure 8, the absolute arrival time t_i of frame i is smaller than the stored timestamp of the last completed hyperperiod t_j^h because t overflowed at t^{max}. Therefore, the calculated relative position t_i^{rel} results in a negative value according to Equation 2, i.e., it underflows. As the data type of the timestamp is unsigned, the negative value is represented as a very large value. An underflow is therefore detected if the calculated relative timestamp t_i^{rel} results in a very large value, i.e, larger than the hyperperiod h. If an underflow is detected,

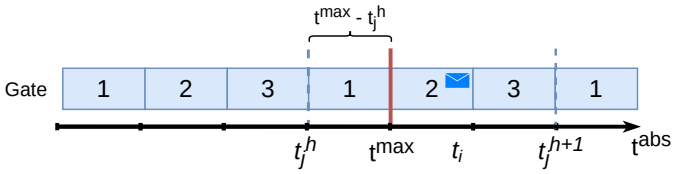


Fig. 8. The relative timestamp t_i^{rel} of a frame i arriving at t_i after t^{max} was reached results in an underflow if the timestamp of the last elapsed hyperperiod t_j^h was set before t^{max}.

the elapsed time of the hyperperiod before t^{max} was reached is calculated and added to the arrival time t_i of the frame as shown in Equation 3.

$$t_i^{rel} = t^{max} - t_j^h + t_i \tag{3}$$

The packet generator of the TNA can be configured with eight different periodic triggers [40], i.e., eight different hyperperiods can be implemented on a P4-PSFP switch. Each hyperperiod comprises multiple stream gate instances with individual stream GCLs that accommodate multiple streams. The packet generator configures the periodic trigger for a hyperperiod on a per-ingress-port basis. Therefore, P4-PSFP supports eight ingress ports with distinct hyperperiods. On each ingress port, the individual stream GCLs of the stream gate instances yield the hyperperiod of this ingress port.

3) Timestamp Matching Precision: The assignment of a frame to a time slice in a stream GCL is based on its ingress timestamp. The implementation selects 20 bits from the 48 bit wide timestamp of an incoming frame, e.g., bit 12 to 31, to enable matching in the stream gate MAT using the range match type. The MAT timestamp key is truncated to these selected bits. This is necessary to save computing resources in a hardware-constrained switch. The truncation process reduces the potential temporal resolution of time slices in a stream GCL. P4-PSFP supports a temporal resolution of 2 µs to about 2.1 s, i.e., a time slice can be assigned down to a precision of 2 µs and a hyperperiod can be at most 2.1 s long. Although this is technically a limitation of the implementation, it is not a problem. According to a survey on scheduling algorithms for the TAS by Stüber *et al.* [42], stream hyperperiods range from 32 µs in [43] to 500 ms in [44]. The range of selected bits from the timestamp can be adjusted to the actual requirements of the network environment to achieve nanosecond accuracy.

4) Permanent Stream Blocking: Each PSFP component must be able to permanently block traversing streams that do not conform to the stream descriptors indicated during admission control. The stream filter can block a stream if the stream exceeds the maximum frame size. The stream gate can be closed to block streams if frames are transmitted during a time slice in a closed state or if too many bytes arrive in a single time slice. The flow meter can be blocked if the streams using it exceed the announced bandwidth.

Permanently blocking a stream in the stream filter, the closing of a stream gate, and the blocking of a flow meter are all handled entirely in the data plane. As a result, there is no delay between a stream not conforming to its stream descriptor and the P4-PSFP switch blocking the corresponding

PSFP component. The data plane keeps track of the blocking of each stream, each stream gate, and each flow meter in multiple registers.

Additionally, to block the stream gate if too many bytes are transmitted in a single time slice, the remaining number of available bytes per time slice per hyperperiod is maintained in a separate register. This register is decremented by the frame size for each frame received and reset upon receiving a generated hyperperiod frame. Frames are dropped according to the register states. The control plane configures which of the permanent stream blocking mechanisms is enabled.

5) Interaction of the Implemented Mechanisms: There are four different paths a frame can pass in the data plane processing pipeline, combining all previously explained mechanisms. The paths differ for different frame header stacks. They are illustrated in Figure 9.

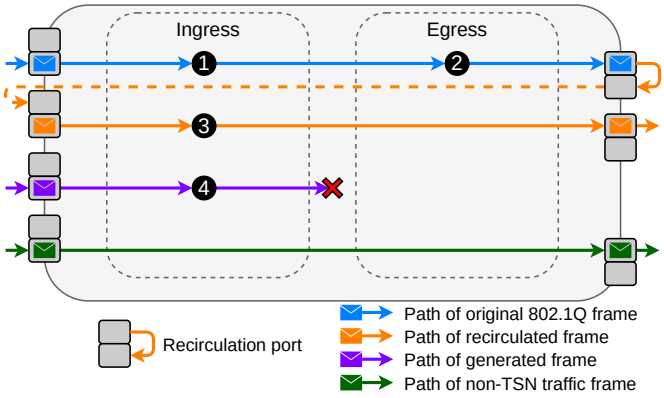


Fig. 9. A frame containing a VLAN tag first takes the blue path and traverses the pipeline once. It is then recirculated and PSFP is applied on the orange path before the frame is forwarded. Generated packets are dropped after ingress processing in the purple path. Non-TSN traffic is forwarded without any further action in the green path.

A frame that has a VLAN tag is eligible for PSFP processing. For such frames, a timestamp of the last completed hyperperiod is retrieved from a register in step ❶. The relative position in the stream GCL is calculated from this value and appended to the frame. The frame is then forwarded to the egress control block.

In egress processing in step ❷, the frame size is added to a recirculation header, together with the relative position in the stream GCL. This increases the size of each frame by additional seven bytes. Furthermore, the detection of overflows when calculating the relative position in a hyperperiod is handled. The frame is recirculated and sent back to an ingress port. The recirculation header can then be used to perform the PSFP mechanism in the ingress control block after recirculation.

Finally, in step ❸, the PSFP mechanism is executed and the stream filter, stream gate, and flow meter instances are applied. Each component is represented as a separate control block consisting of multiple MATs to load configuration parameters and apply the appropriate actions. A frame needs to pass all of the conditions mentioned in Section III-B4 to be forwarded.

Frames in the purple path in Figure 9 correspond to packets generated by the TNA packet generator. They are used to store a hyperperiod timestamp in a register r^h in step ❹ to account for the periodicity of stream GCLs. After that, they are no longer needed and are therefore discarded.

Non-TSN traffic, i.e., frames that do not have a VLAN header, or are not identified by a stream filter entry, are treated as best-effort traffic and are forwarded without any further action.

C. Time Synchronization

High-precision time synchronization is essential to correctly assign a frame to its time slice in the stream GCL. Although the Intel Tofino™ is capable of synchronizing to the network time [34] with the Precision Time Protocol (PTP) [10], not all hardware platforms are capable of doing so. One alternative to PTP is the implementation of the Data Plane Time Synchronization Protocol (DPTP) [11]. However, this requires additional space in the P4 pipeline and is not feasible due to the complexity of the P4-PSFP implementation. This section first describes the problems of lacking time synchronization and then introduces a proposal for achieving time synchronization of stream GCLs in the data plane where the application of protocols like PTP is not possible.

1) Problem Statement: The problems resulting from a lack of time synchronization are manifold. We identify and present three root causes and their implications on the time synchronity of stream GCLs.

First, the data plane of P4-PSFP and its control plane may run on different hardware and will therefore have a different hardware time. We assume that the control plane is part of a time-synchronized network, e.g., the control plane is synchronized using a protocol such as PTP. Talkers and the control plane in the TSN network are synchronized to the network time, but the data plane is not. This is illustrated in Figure 10.

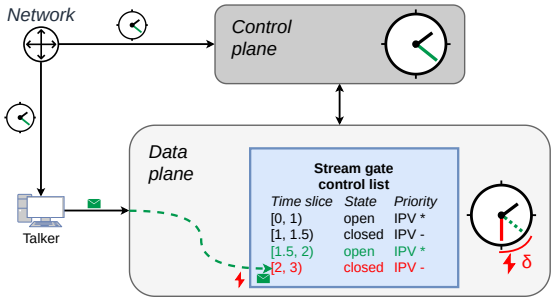


Fig. 10. The control plane and the talker are synchronized to the network time, but the data plane is not. The time differs by the value δ.

In Figure 10, the data plane has an offset of δ relative to the network time. This offset results in the network time currently being in the green time slice while the data plane time is in the red time slice. A frame that should arrive in an open state time slice may arrive in a closed state time slice due to lack of time synchronization. The offset must be corrected so that the data plane correctly assigns incoming frames to time slices in the stream GCL according to the network time.

The second root cause lies within small time delays ε_1 originating from clock drifts or inaccuracies. Craciunas *et al.* [45] identify the problem of a temporary loss of time

synchronization for time-triggered schedules, leading to clock drift. Clock drifts in P4-PSFP can occur due to delays or inaccuracies in the reception or the generation of hyperperiod packets. They may be very small, i.e., in the order of nanoseconds, but can accumulate over time. The value of ε_1 is determined by comparing the measured timestamp t_j^h with the expected duration of $j-1$ hyperperiods since the first completed hyperperiod t_1^h. This is shown in Equation 4. Since ε_1 is always less than or equal to the duration[4] h, we can apply the modulo operation in Equation 4 to simplify the formula to Equation 5.

$$\varepsilon_1 = (\underbrace{(t_j^h - t_1^h)}_{\text{measured}} - \underbrace{(j-1)\cdot h}_{\text{expected}}) \bmod h \tag{4}$$

$$\Leftrightarrow \varepsilon_1 = (t_j^h - t_1^h) \bmod h \tag{5}$$

Finally, the third root cause for time inaccuracy results from the configuration of the packet generator in P4-PSFP. The packet generator in P4-PSFP is configured for one ingress port at a time. This results in the hyperperiod packet being generated with a small delay ε_2 in between the start of each periodic trigger for each port. Stream GCLs on different ingress ports must therefore be synchronized with each other. This is illustrated in Figure 11.

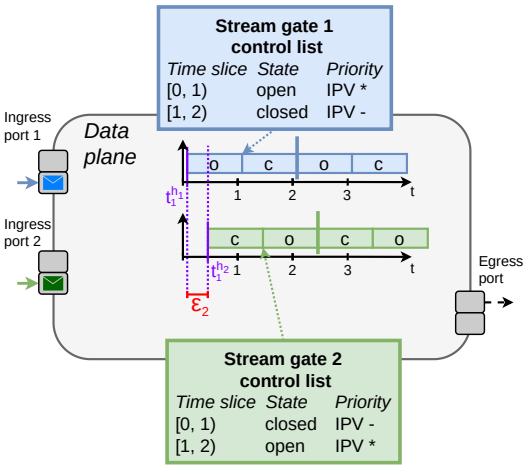


Fig. 11. Two stream GCLs with complementary gate states are configured on two different ingress ports. Therefore they do not share the same hyperperiodic trigger. The second stream GCL starts with a small delay of ε_2.

Two streams on different ingress ports arrive at the P4-PSFP switch in Figure 11. The stream GCLs of both streams are configured to have complementary gate states, i.e., only one stream at a time is allowed to transmit. As both streams are physically on two different ingress ports, they do not share the same hyperperiodic packet generator trigger even though they may have the same hyperperiod duration. The hyperperiodic timestamp t_j^h is required to calculate the relative temporal position t_i^{rel} of a frame i, as described in Equation 2. Let $t_1^{h_k}$ be the arrival of the first hyperperiod packet on port k. The delay ε_2 between the two stream GCLs can then be calculated as the difference between the first hyperperiodic timestamps on each port. This is shown in Equation 6.

$$\varepsilon_2 = t_1^{h_2} - t_1^{h_1} \tag{6}$$

Due to this delay, the time slices of both stream GCLs partially overlap by ε_2 in the open state and do not allow for exclusive transmission of one stream at a time.

2) Time Synchronization by Δ-Adjustment: In the following, we present a mechanism to adjust the calculated relative position in the stream GCL by a dynamic offset value $\Delta = \delta + \varepsilon_1 + \varepsilon_2$. With the proposed approach, P4-PSFP is able to synchronize the data plane with a time-synchronized control plane, account for clock drifts during the runtime, and synchronize stream GCLs on different ingress ports. The control plane measures the delay by reading the hyperperiodic timestamp registers from the data plane and calculates ε_1 and ε_2 according to Equation 5 and Equation 6. The sum of all sources of time inaccuracy $\Delta \in \mathbb{Z}$ is then written by the control plane to a MAT in the data plane. The data plane modifies the relative temporal position t_i^{rel} of a frame in a stream GCL according to Δ. The mechanism is illustrated in Figure 12 and further explained below.

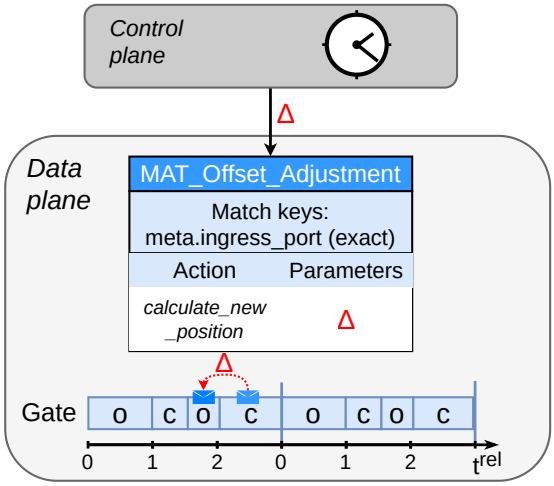


Fig. 12. Due to time inaccuracy, a frame was erroneously assigned to the fourth time slice in the stream GCL. In fact, this frame arrived in the third time slice of the stream GCL. The error is corrected by subtracting an offset value Δ from the previously calculated relative position t_i^{rel}.

First, the control plane determines the dynamic offset value $\Delta = \delta + \varepsilon_1 + \varepsilon_2$ as described in Section V-C1. The value of ε_1 is updated every 100 ms to account for varying time inaccuracies and clock drifts during runtime. The sum of all individual sources of time inaccuracy Δ per ingress port is written to a MAT. It is important to keep this MAT update atomic to not introduce inconsistencies due to multiple MAT updates in multiple operations. Once this value is written to the MAT, the Δ-adjustment is performed entirely in the data plane and adds no further delay to packet processing.

In the data plane, the action associated with the Δ-adjustment MAT modifies the relative position t_i^{rel} of frame i. The value of Δ is added to the previously calculated relative position t_i^{rel} described in Section V-B2. The Δ-

[4]If ε_1 is larger than h, it simply skips over a complete hyperperiod.

adjusted relative position of a frame i in a stream GCL is determined in Equation 7.

$$t_{i_\Delta}^{rel} = t_i^{rel} + \Delta \tag{7}$$

By adding the offset value Δ to the relative position t_i^{rel}, the inaccuracy in the relative position of the frame is corrected. The frame may be assigned to a different time slice in the stream GCL than it was before.

Altering the relative position and assigning a different time slice introduces a new problem. Δ is added as an absolute value to the relative position t_i^{rel} of the frame. Consequently, the Δ-adjustment action may assign a position in the stream GCL that does not exist. The new temporal position may result in an underflow or overflow of a frame relative to its hyperperiod and the packet is erroneously dropped. For example, a frame arriving at $t_i^{rel} = 0$ with an offset value of $\Delta = -1$ is assigned to a non-existent negative time slice $t_{i_\Delta}^{rel} = -1$ according to Equation 7. P4-PSFP must ensure that the periodicity is maintained even if arbitrary values are added to or subtracted from the relative position without disrupting correct packet forwarding at any time. A solution to this problem, i.e., the calculation of the relative position $t_{i_\Delta}^{rel}$ with respect to underflows and overflows, is illustrated in Figure 13.

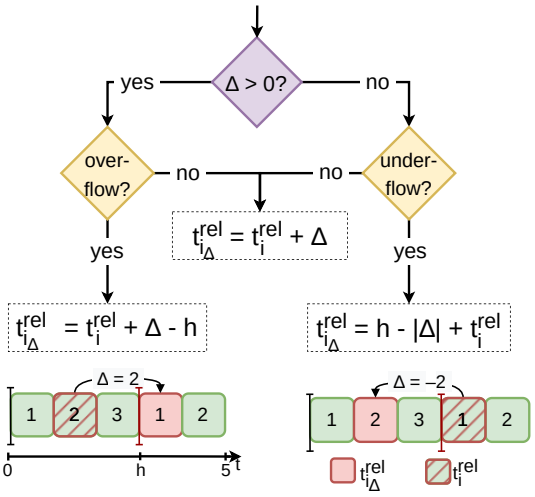


Fig. 13. Possible new positions in the hyperperiod for frames with a hyperperiod of the duration $h = 3$ time steps and an offset $\Delta = \pm 2$ time steps. Simply adding the offset $\Delta = -2$ to the relative timestamp $t_i^{rel} = 1$ results in a new relative position $t_{i_\Delta}^{rel} = -1$, which is not in a valid time slice. P4-PSFP has to account for an under- or overflow with the provided formulas.

Figure 13 shows the Δ-adjustment mechanism in the data plane with respect to over- and underflows of the relative position of a frame. First, we have to detect whether the offset Δ is positive or negative, i.e, if Δ needs to be added or subtracted. If it is positive, $t_{i_\Delta}^{rel}$ can potentially overflow, or underflow if it is negative. As the data type of Δ is unsigned, we cannot check for negative values in the data plane. The sign of Δ is identified by the control plane and written into a MAT with a single entry per port in an atomic operation [5].

[5]By having one MAT for positive values and one MAT for negative values, the order of updates of the sign MAT and of the Δ-MATs ensures that no inconsistency is introduced by performing the two MAT updates.

Letting the control plane handle the sign determination also saves computational resources in the data plane.

Second, we have to detect an underflow or an overflow. On an overflow, the relative position corrected by Δ will exceed the hyperperiod and be assigned to a non-existent time slice larger than the hyperperiod. Similarly, on an underflow, the relative position corrected by Δ is assigned to a non-existent time slice smaller than zero. An underflow or an overflow is detected if the newly calculated position $t_{i_\Delta}^{rel}$ results in a very large value, i.e., larger than a hyperperiod or the offset. If no over- or underflow is detected, $t_{i_\Delta}^{rel}$ can be calculated according to Equation 7.

Third, if an over- or underflow is detected, we have to adjust the relative position t_i^{rel} to keep its position in a hyperperiod. To that end, Equation 8 must be used on an overflow and Equation 9 must be used on an underflow.

$$t_{i_\Delta}^{rel} = t_i^{rel} + \Delta - h \tag{8}$$

$$t_{i_\Delta}^{rel} = h - |\Delta| + t_i^{rel} \tag{9}$$

For an overflow, it holds that $t_i^{rel} + \Delta > h$. Therefore, in Equation 8 the duration of a full hyperperiod h is subtracted to keep the relative position in the range from 0 to h. Similarly, for an underflow, it holds that $t_i^{rel} - \Delta < 0$. Therefore, Equation 9 takes a full hyperperiod, subtracts the absolute value of Δ, and adds the relative position t_i^{rel} to stay in the range from 0 to h.

The explained approach for time synchronization by Δ-adjustment is performed in step ❷ in Figure 9, i.e, in egress processing before the recirculation.

VI. Evaluation of P4-PSFP

In this section, we evaluate the implementation of PSFP in P4 on the Intel Tofino™ switching ASIC. First, we describe the testbed environment used for our evaluation. Second, we test the functionality of flow meters, stream GCLs, and the Δ-adjustment. Third, we evaluate the performance of the implementation in an overloaded network environment with and without PSFP enabled and show how it affects the latency. Finally, we analyze the scalability of the implementation on hardware with respect to the number of supported streams when using different stream identification functions.

A. Testbed Environment

The testbed environment used for our evaluation consists of two switches, one running the traffic generator P4TG [28] and the other running the implementation of P4-PSFP. The environment is illustrated in Figure 14.

An Edgecore Wedge 100BF-32X switch with an Intel Tofino™ ASIC running the P4-PSFP implementation is connected via 100 Gb/s links to another switch with an Intel Tofino™ ASIC running the P4TG [28] traffic generator. P4TG generates constant bit-rate (CBR) traffic at a transmission rate of 100 Gb/s and a frame size of 1280 bytes. As frames generated by P4TG do not contain a VLAN tag, we prepend an additional VLAN header to the frame before processing the PSFP pipeline. This header is removed when the frame leaves the P4-PSFP switch. The P4-PSFP policed traffic is fed

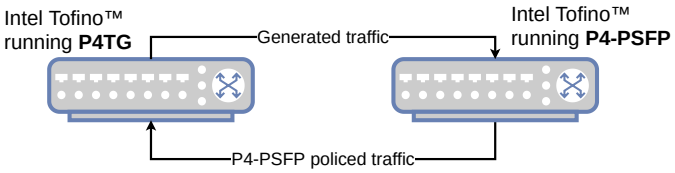


Fig. 14. The testbed environment used for evaluation. Traffic generated by an Intel Tofino™ running P4TG is sent through an Intel Tofino™ running our P4-PSFP implementation and back to P4TG.

back to the input ports of the P4TG switch to perform latency analysis.

B. Functionality of PSFP Components

This section presents the experiments to test the functionality of the PSFP components in the implementation. We perform tests to verify the functionality of flow meters, i.e., the credit-based metering, stream gates, i.e., the periodicity of stream GCLs and the time-based metering, and the approach for time synchronization with Δ-adjustment. Stream filters and their stream identification are implicitly verified by the following experiments as stream gate and flow meter instances are not applied if a stream filter has not identified them. We evaluated the implemented PSFP parameters *MaxSDUExceeded*, *GateClosedDueToInvalidRX*, *GateClosedDueToOctetsExceeded*, *MarkAllFramesRed*, and *DropOnYellow* as described in Section III-B4. However, for the sake of clarity, only the latter two are described in the evaluation.

1) Functionality of the Flow Meter: In this section, the functionality of the flow meter instance, i.e., the credit-based metering and its additional parameters as summarized in Table I are evaluated.

The traffic generator P4TG [28] is used to generate a $100\,\mathrm{Gb/s}$ stream which passes through the PSFP mechanism in the implementation. The P4-PSFP switch applies the flow meter to the generated traffic stream to label frames in their respective colors. The control plane measures the traffic rates for each frame color.

We set the Committed Information Rate (CIR) to $70\,\mathrm{Gb/s}$ and the Excess Information Rate (EIR) to $20\,\mathrm{Gb/s}$. The stream gate remains in a permanently open state, i.e., no time-based metering is applied.

We test the implementation in three settings which are all verified in a single experiment over time. First, no additional parameters are set to verify the functionality of the token bucket algorithm and the frame coloring. Second, the *DropOnYellow* parameter is set which additionally marks yellow-labeled frames red and drops them. Third, the *MarkAllFramesRed* parameter is set which permanently closes the flow meter instance if a single frame exceeds the EIR, i.e., once a single frame is colored red.

The experiment runs for $20\,\mathrm{s}$. Up to $t_0 = 2000\,\mu\mathrm{s}$, no additional parameters are set. At t_0, we set the parameter *DropOnYellow* and at $t_1 = 4000\,\mu\mathrm{s}$, we set the parameter *MarkAllFramesRed*. Figure 15 compiles the measured traffic rates per frame color over time for the first $6\,\mathrm{ms}$ in the experiment. After $6\,\mathrm{ms}$, the measured traffic rates do not change anymore.

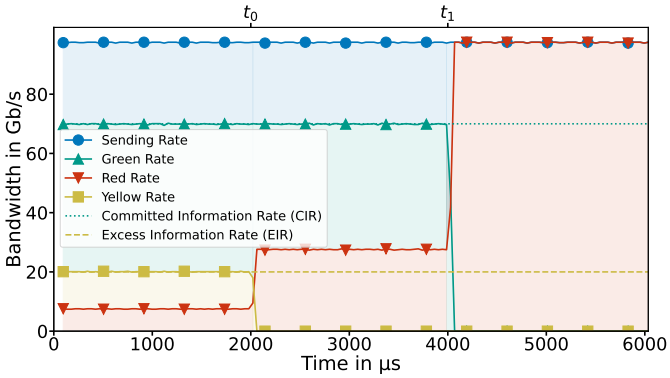


Fig. 15. Rates of colored frames in the flow meter instance while receiving frames at a rate of $100\,\mathrm{Gb/s}$. The horizontal dashed lines show the configured CIR and EIR. At t_0, the parameter *DropOnYellow* is set and at t_1 the parameter *MarkAllFramesRed* is set.

Up to time t_0, the measured rates for the green and yellow labeled frames align with the configured rates of the CIR and EIR. $70\,\mathrm{Gb/s}$ of the total generated traffic stream at a rate of $100\,\mathrm{Gb/s}$ is labeled green while $20\,\mathrm{Gb/s}$ is labeled as yellow. The remaining portion of the generated traffic stream is categorized as red. The sending rate is slightly below $100\,\mathrm{Gb/s}$ due to the extra bytes added for the recirculation header by P4-PSFP as described in Section V-B5.

By setting the *DropOnYellow* parameter at t_0, in Figure 15, the yellow rate immediately drops to zero while the red rate is increased by the value of the EIR. Yellow frames are now labeled red and therefore dropped.

At time t_1, we enable the *MarkAllFramesRed* parameter. As a result, the red rate in Figure 15 aligns with the sending rate while the green rate drops to zero. Frames in this flow meter instance are no longer forwarded.

These experiments show that the implementation of the PSFP flow meter instance correctly applies a credit-based metering approach to an incoming stream. The parameter *DropOnYellow* successfully drops yellow-labeled frames, and the parameter *MarkAllFramesRed* successfully blocks streams permanently as soon as they do not adhere to their stream descriptor for the first time.

2) Functionality of the Stream Gate: We define the *1-4-2-1* stream GCL to verify the periodicity and the time-based metering in the stream gate implementation. The *1-4-2-1* stream GCL consists of four time slices of the duration $100\,\mu\mathrm{s}$, $400\,\mu\mathrm{s}$, $200\,\mu\mathrm{s}$ and $100\,\mu\mathrm{s}$. Its period is therefore $800\,\mu\mathrm{s}$. The time slice durations are chosen arbitrarily and have no further meaning. The stream gate states of the *1-4-2-1* stream GCL are alternating, starting in the open state.

Again, the traffic generator P4TG [28] is used to generate a $100\,\mathrm{Gb/s}$ CBR traffic stream. The flow meter instance is disabled for the evaluation of the time-based metering, i.e., no credit-based metering is applied in this experiment.

The objectives of this experiment are twofold. First, the time-based metering and the periodicity of stream GCLs are verified. Second, the ability of the Δ-adjustment to apply an

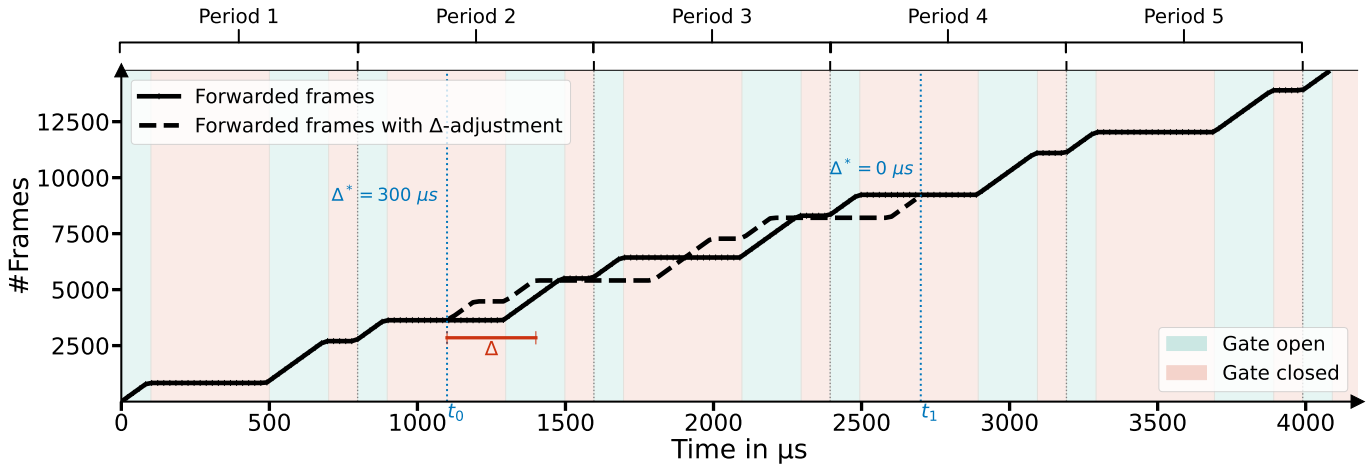


Fig. 16. Plot of forwarded frames in the stream gate instance. Periods are indicated by vertical lines. Gate states of the *1-4-2-1* stream GCL are illustrated as either red (closed) or green (open). The solid line indicates the forwarded packets without applying the Δ-adjustment. There are five repeating periods of the *1-4-2-1* stream GCL. For the dashed line, a Δ-adjustment of $\Delta^* = 300\,\mu s$ is applied between t_0 and t_1.

offset Δ^* at any given time without introducing inconsistencies is verified. For the first objective, we apply the *1-4-2-1* stream GCL to the generated traffic stream. For the second objective, we additionally introduce a Δ-adjustment of $\Delta^* = 300\,\mu s$ at $t_0 = 1100\,\mu s$. At $t_1 = 2700\,\mu s$, we reset the offset Δ^* to $0\,\mu s$. We measure the number of forwarded frames by the stream gate instance in the P4-PSFP switch over time to verify the objectives.

The results are compiled in Figure 16. In this plot, a rising edge indicates active frame forwarding while a plateau indicates the suspension of frame forwarding. In addition, the time slices of the stream GCL in the closed state are highlighted in red whereas those in the open state are marked in green. The solid line illustrates five consecutive cycles of the *1-4-2-1* stream GCL. The number of cycles measured during the runtime of $4000\,\mu s$ corresponds to the expected number of cycles of the stream GCL with a period of $800\,\mu s$. Moreover, the rising edges correspond exactly to the areas marked in green and the plateaus correspond to the areas marked in red, i.e., the four measured alternating time slices per period exactly align with the predefined time slices specified in the stream GCL. This experiment validates the time-based metering, and the proposed solution for achieving the periodicity of stream GCLs by using the internal traffic generator as described in Section V-B2.

The second experiment is represented by the dashed black line in the plot. In this experiment, we set the offset value $\Delta^* = 300\,\mu s$ at $t_0 = 1100\,\mu s$. Consequently, the curve follows the curve without the Δ-adjustment for $1100\,\mu s$ and is then shifted by Δ^* to the left during the second period. Each frame is now assigned to a time slice determined by its ingress timestamp plus $300\,\mu s$. The third period corresponds to a period in which all time slices of the *1-4-2-1* stream GCL are shifted by Δ^*. In the fourth period, we reset the offset value Δ^* to $0\,\mu s$ at $t_1 = 2700\,\mu s$. At this point, the curve immediately returns to its original path without the Δ^* adjustment. Finally, the fifth period mirrors the characteristics of the first period because no Δ^* value is applied. Through this experiment, we demonstrate that the proposed approach to account for time synchronization can seamlessly introduce an arbitrary offset value at an arbitrary time. This compensates for time inaccuracies such as accumulated clock drift or inaccuracies without causing inconsistencies in forwarding.

C. PSFP Application to Competing Streams in an Overloaded Network

In this section, we evaluate the P4-PSFP implementation in a setting where two streams compete for limited resources in an overloaded network. First, we give an overview. Then we describe the setup and finally, we present the results.

1) Overview: In a typical TSN setup, time-triggered talkers communicate in real-time. A traffic shaper such as the TAS is used to schedule this time-sensitive traffic. Since the stream gate component of PSFP can be explained as a TAS not bound to priority queues, we leverage the implementation of P4-PSFP to protect the schedules of a hypothetical TAS by dropping frames received in a disallowed time slice before they enter the TAS. We leverage the traffic generator P4TG to force queueing by overloading the capacity of a $100\,\mathrm{Gb/s}$ link with two $90\,\mathrm{Gb/s}$ streams, thereby introducing queueing and increasing latency. Higher latency can cause frames to arrive outside their intended time slice in the TAS schedule. We aim to protect the schedules of the hypothetical TAS by applying different configurations of stream GCLs in PSFP. We hypothesize that the latency on a congested network link can be reduced with P4-PSFP.

2) Experiment Setup: We use the traffic generator P4TG to generate two streams of $90\,\mathrm{Gb/s}$ each. Each stream has its own ingress and recirculation port in the P4-PSFP switch. However, both streams are sent back to the P4TG switch via the same egress port after PSFP is applied. This will overload the egress network link which has a capacity of $100\,\mathrm{Gb/s}$. An overview of the experiment setup is given in Figure 17.

Two streams generated by P4TG are considered in our experiment setup. The first is the P4TG-RT stream which has

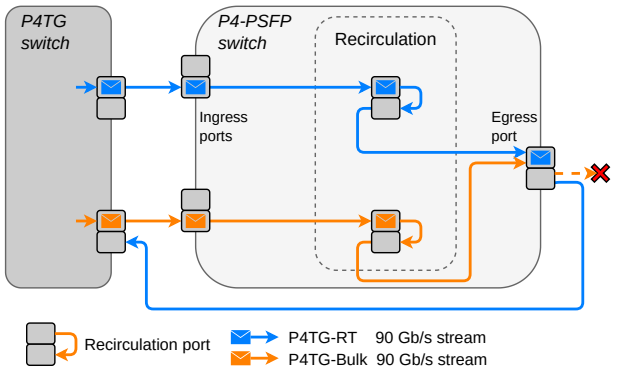


Fig. 17. Two 90 Gb/s streams are generated by P4TG and sent to the P4-PSFP switch. Each stream has its own ingress and recirculation port, but both streams are sent back through the same egress port which causes queueing. Frames from the P4TG-RT stream are used to measure the latency while frames from the P4TG-Bulk stream are dropped after queueing.

a strict real-time requirement and a tight latency constraint. Some industrial control applications require meeting strict latency limits, sometimes as low as a few microseconds, as stated in [25]. Second, the P4TG-Bulk stream represents traffic that interferes with the P4TG-RT stream and has no real-time requirement.

Since we do not have a TSN testbed environment with synchronized talkers, P4TG transmits both streams continuously, regardless of the allowed transmission times. Frames received during time slices in the closed state are dropped by the stream gate instance in P4-PSFP before egress queuing and therefore do not affect the latency.

Sending two 90 Gb/s streams through the same egress port will overload the egress network link and cause queueing, which increases the latency. We consider the latency caused by queueing in the egress port of the P4-PSFP switch by applying three different stream GCLs in the implementation. First, we apply a stream GCL that leaves the stream gates of both streams in a permanently open state to measure the latency in the congested network as a baseline without PSFP. Second, we apply the *50-50* stream GCL, which consists of a 200 ms time slice in the closed state, followed by a 200 ms time slice in the open state to the P4TG-Bulk stream. The stream gate instance associated with the P4TG-RT stream remains permanently open. Third, we apply the *50-50* stream GCL to the P4TG-Bulk stream and an inverted *50-50* stream GCL to the P4TG-RT stream, i.e., only one stream is allowed to transmit at a time.

Stream GCLs configured on different ingress ports are not synchronized with each other because the hyperperiodic packet generator trigger is configured on a per-ingress-port basis, as explained in Section V-C1. As a result, time slices from the P4TG-RT stream GCL overlap with time slices from the P4TG-Bulk stream GCL by an offset ε_2. To solve this problem, we apply the Δ-adjustment to synchronize the two stream GCLs. The difference between the stored hyperperiod timestamps ε_2 of both stream GCLs is calculated by the control plane at the beginning.

The P4-PSFP switch drops frames from the P4TG-Bulk stream after PSFP processing, i.e., after egress queueing, to measure only the latency of the P4TG-RT stream in P4TG.

3) Results: In this section, we present the results of the three experiments with different stream GCLs. In the setting where PSFP is disabled, the 100 Gb/s link becomes congested and frames are queued in the egress port, increasing the latency. The latency remains consistently high at $\approx$ 98 µs. This can be seen in Figure 18(a).

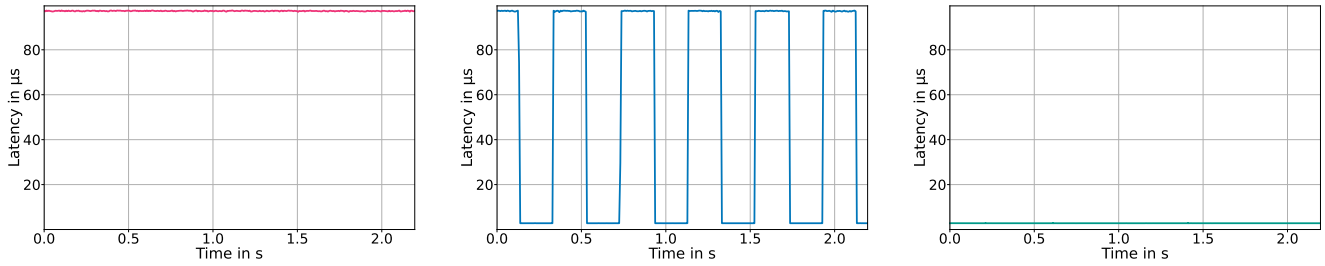
In the second setting, the *50-50* stream GCL is applied to the P4TG-Bulk stream. In this setting, the latency drops to 3 µs for 200 ms during time slices where the P4TG-Bulk stream is blocked, and rises to 98 µs when both streams are transmitting. This is visible in Figure 18(b).

In the third experiment, the stream gate is configured to only allow one stream to transmit at a time. Here, the latency drops to 3 µs permanently. This is shown in Figure 18(c). The exact alignment of the time slices in both stream GCLs allows for the exclusive transmission of one stream at a time without inconsistencies during state transitions. This is achieved by synchronizing the two stream GCLs on two different ingress ports with the Δ-adjustment.

These experiments show that the implementation of P4-PSFP is capable of protecting TAS schedules with time-based metering and appropriate stream GCL configuration. The latency reduction achieved by P4-PSFP effectively shows the elimination of queueing in the overloaded egress port queue. This is accomplished by configuring P4-PSFP with two stream GCLs that alternate between the two streams and by applying the Δ-adjustment. The application of the Δ-adjustment shows that P4-PSFP is able to synchronize multiple stream GCLs on different ingress ports to each other. Frames that enter the TAS after P4-PSFP processing no longer experience congestion-related delay.

D. Scalability

In this section, we evaluate the scalability of the implementation on the Intel Tofino™. The resources used during the processing of a frame, i.e., tables, actions, and externs, are limited to maintain the line rate of 100 Gb/s. All resources used share a common memory. Sufficient memory capacity must be available to hold the used resources. Larger tables, i.e., tables with more key fields, actions, and action parameters, require more memory. The maximum size of MATs, i.e. the number of entries a MAT can hold, must be defined at compile time and cannot be changed at runtime. If the defined maximum size of a MAT exceeds the available resources, a compilation error occurs and the P4 program cannot be executed. An overview of the available MATs in P4-PSFP is given in Section V-A. The maximum number of entries in the stream gate MAT to achieve a successful compilation of P4-PSFP proved to be 2048 entries, i.e., 2048 different time slices of stream GCLs can be modeled. We gradually increase the size of the stream identification MAT while keeping the stream gate MAT size at 2048 entries to achieve a successful compilation of P4-PSFP that finds the maximum possible number of supported streams. We set the size of the flow meter MAT equal to the size of the stream identification MAT.



(a) PSFP disabled.

(b) One stream scheduled with the *50-50* stream GCL.

(c) Both streams scheduled with complementary *50-50* stream GCLs.

Fig. 18. Latency of the P4TG-RT stream in the congested network setting after applying different stream GCL configurations to both P4TG streams.

TABLE II
OVERVIEW OF DIFFERENT STREAM IDENTIFICATION FUNCTIONS ACCORDING TO IEEE 802.1CB [8]. DEPENDING ON THE MATCH FIELDS AND TYPES, DIFFERENT AMOUNTS OF STREAMS CAN BE SUPPORTED IN OUR IMPLEMENTATION OF PSFP.

Stream identification function	Null stream	Source MAC	IP stream (ternary)	IP stream (exact)
Ethernet source address		✓ ex.	✓ ter.	
Ethernet destination address	✓ ex.	✓ ter.	✓ ter.	✓ ex.
VLAN ID	✓ ex.	✓ ex.	✓ ex.	✓ ex.
IP source address			✓ ter.	✓ ex.
IP destination address			✓ ter.	✓ ex.
DSCP			✓ ter.	✓ ex.
Next Protocol			✓ ter.	✓ ex.
Source port			✓ ter.	✓ ex.
Destination port			✓ ter.	✓ ex.
Max. number of stream identification entries	35840	4096	2048	32768

Using different stream identification functions, i.e., a different composition of the stream identification MAT key, results in a different maximum number of streams that can be supported. Matching fields for stream identification are either used as *exact* or as *ternary* type. Ternary matching fields allow for aggregation of streams, or for wildcards, meaning that a single ternary entry in the MAT can match multiple different streams. As a tradeoff, a ternary matching entry consumes more memory than an exact matching entry. An overview of different stream identification MAT compositions and the number of entries supported is compiled in Table II.

P4-PSFP can support up to 35840 different streams when using 2048 time slices with the mandatory null stream identification function [8] that only matches on the Ethernet destination address and the VLAN ID. If we use exact IP stream identification, the number of entries reduces to 32768 different streams for exact matches, or 2048 entries for aggregated wildcard streams with ternary matches. In the survey conducted by Stüber *et al.* [42], the number of streams discussed in various research papers reached up to 10812 streams in the case of [46]. Consequently, to accommodate the number of streams required for the most extensive TSN configuration described in [42], we can choose either null stream identification or exact IP stream identification while having a total of 2048 time slices in stream GCLs.

VII. DISCUSSION

In this section, we discuss the effect of recirculation on the implementation and requirements to stream GCLs in P4-PSFP. Further, we discuss the requirements for porting P4-PSFP to other hardware targets. We also describe applications that can take advantage of the $100\,\mathrm{Gb/s}$ bandwidth capability of P4-PSFP.

A. The Effect of Recirculation in P4-PSFP

The concept of recirculation allows a packet to be processed multiple times, iteratively manipulating its header data. A packet that undergoes a recirculation requires capacity in terms of bandwidth, e.g., a packet that is recirculated once doubles its total required bandwidth. For this purpose, the Intel Tofino™ has internal ports that provide capacity for recirculation. Physical ports can be configured to be used as recirculation ports to increase the capacity available for recirculation. If the recirculation capacity is exceeded, e.g., by directing more than $100\,\mathrm{Gb/s}$ of traffic to a $100\,\mathrm{Gb/s}$ port, queueing will occur and packets may be dropped. Furthermore, a recirculation on the Intel Tofino™ adds a constant amount of time[6] to the processing delay.

In P4-PSFP, one recirculation is required for each frame to retrieve the frame size for the stream filter instance as described in Section V-B1. Each of the eight ingress ports in P4-PSFP has a dedicated recirculation port to prevent exceeding the available recirculation capacity. If the ingress ports operate at less than $100\,\mathrm{Gb/s}$, e.g., at $10\,\mathrm{Gb/s}$, a single recirculation port is sufficient for the P4-PSFP switch. On the Intel Tofino™, each recirculation port experiences the same

[6]If the recirculation port is not overloaded.

constant sub-microsecond delay. Therefore, the recirculation delay in P4-PSFP can be considered part of the overall processing delay of each frame imposed by the hardware device. For matching the frame to its time slice in the stream GCL, the ingress timestamp before a recirculation is used. This allows a frame to be assigned to a time slice according to its arrival time at the switch.

B. Requirements in P4-PSFP

In this section, we summarize and discuss the requirements to stream GCLs in P4-PSFP. We further discuss the requirements of other P4-based hardware targets to support P4-PSFP.

1) Requirements to Stream GCLs: In P4-PSFP, all individual stream GCLs on the same ingress port must be extended to a hyperperiod which is formed by the LCM of the periods of all stream GCLs. Forming a hyperperiod to model the individual stream GCLs is a common practice according to Stüber *et al.* [42] and is not specific to P4-PSFP. Specifically for P4-PSFP, a hyperperiod must be in the range of $2\,\mu s$ to $\approx 2.1\,s$. According to a survey by Stüber *et al.* [42], this is not a limitation as stream hyperperiods in their survey range from $32\,\mu s$ to $500\,ms$.

2) Requirements to Other Hardware Targets: For P4-PSFP to be portable to other hardware targets, the components of the PSFP mechanism, namely the stream filter, the stream gate, and the flow meter, must be implemented. While the stream filter component is mostly covered by basic MATs, the flow meter component must be implemented in the form of the token bucket algorithm, e.g., by a meter extern. For the stream gate component, the main requirement is a trigger to determine the end of a hyperperiod. This trigger must indicate the end of the hyperperiod with a hardware-based timestamp. On the Intel Tofino™, the internal traffic generator is used for this purpose. On other hardware, a CPU-based generator could be leveraged. Inaccuracies caused by the hyperperiod trigger can be compensated by the mechanism presented in Section V-C2.

Another requirement for the implementation is a strong time synchronization. Ideally, the data plane is synchronized via PTP. Otherwise, the control plane can be synchronized with PTP and the Δ-adjustment mechanism can be used to account for the synchronization in the data plane.

C. P4-PSFP across Other Domains

The provided P4-PSFP implementation supports rates up to $100\,Gb/s$ while most TSN networks operate with line rates of at most $1\,Gb/s$. DetNet extends the reach of TSN networks into the IP and MPLS domain as described in Section II-B2. The DetNet working group focuses on networks that are under a single administrative control, such as private WANs or campus-wide networks. In this environment, TSN subnetworks could operate at higher bandwidths and benefit from the PSFP mechanism.

Furthermore, the IEEE 802.3ch working group is currently discussing a standard for single-pair Ethernet connections for automotive applications with up to $10\,Gb/s$ [47]. Since TSN mechanisms are already deployed in automotive applications, such a standard would benefit from a PSFP implementation capable of higher bandwidths.

Another emerging area is distributed AI in data centers. Here, network latency has become increasingly important, and methods such as Remote Direct Memory Access (RDMA) are being used to drastically reduce latency with a shared memory. Collective AI operations distribute and collect their data over the network to enable for interaction between multiple compute nodes. This requires high bandwidth and low latency. While this is not a typical application for TSN networks, the policing of the PSFP mechanism could help reduce latency in such data centers. However, PSFP also requires a highly synchronized network and scheduled traffic, which is usually not the case in such environments. Applying the concept of PSFP to data-center networks is an interesting concept that is left open for future work.

VIII. Conclusion

This paper presents P4-PSFP, a novel implementation of the PSFP mechanism in the P4 language on a hardware-based switching ASIC. P4-PSFP is, to the best of our knowledge, the first full-fledged implementation of PSFP on real hardware that supports the full set of functions conform to IEEE Std 802.1Qci [6]. We successfully identified and solved challenges encountered when using real hardware, such as constrained computational resources, achieving periodicity of stream GCLs, and time synchronization.

The functionality of each of the PSFP components, i.e., stream filters, stream gates, and flow meters was verified in extensive evaluations, including the correct operation of the credit-based metering, stream GCLs and their periodicity, i.e., the time-based metering, and the Δ-adjustment for time synchronization.

Furthermore, P4-PSFP was evaluated in a network environment where two links with an overall transmitting rate of $180\,Gb/s$ overload a link with a capacity of $100\,Gb/s$, causing queueing in the egress port queue. By configuring stream GCLs to allow only one stream transmission at a time, P4-PSFP effectively eliminates queueing in the congested egress port queue through highly accurate synchronization of the stream GCLs.

The scalability analysis showed that P4-PSFP can support 2048 different stream GCL time slices, and up to 35840 different streams, depending on the stream identification function. The code of P4-PSFP is available on GitHub [41].

P4-PSFP can be used to ensure that streams in a TSN network conform to their announced stream descriptors until a full-fledged hardware implementation on appropriate TSN switches is available. Moreover, the concepts implemented in P4-PSFP can be individually leveraged and reused in other implementations. The proposed approach to account for time inaccuracy of time-critical components in the data plane, e.g., stream GCLs, can be utilized in various implementations where highly accurate time synchronization is required and protocols such as PTP are not available. The mechanism described for achieving the periodicity of schedules by using the internal traffic generator of the Intel Tofino™ can be used in other implementations that require periodic behavior.

List of abbreviations

QoS quality of service
TSN Time-Sensitive Networking
GCL gate control list
PTP Precision Time Protocol
IAT inter-arrival time
DPTP Data Plane Time Synchronization Protocol
PSFP Per-Stream Filtering and Policing
IPV Internal Priority Value
PCP Priority Code Point
CIR Committed Information Rate
EIR Excess Information Rate
DEI DropEligibleIndicator
P4 Programming Protocol-independent Packet Processors
MAT match+action table
LPM longest-prefix-match
PSA Portable Switch Architecture
TNA Tofino Native Architecture
TDMA Time Division Multiple Access
LCM least common multiple
TAS Time-Aware Shaper
CBS Credit-Based Shaper
CBR constant bit-rate

References

[1] J. L. Messenger, "Time-Sensitive Networking: An Introduction," *IEEE Communications Standards Magazine*, vol. 2, pp. 29–33, June 2018.

[2] L. Wüsteney, M. Menth, R. Hummen, and T. Heer, "Impact of Packet Filtering on Time-Sensitive Networking Traffic," in *IEEE International Workshop on Factory Communication Systems (WFCS)*, pp. 59–66, June 2021.

[3] "IEEE Standard for Local and Metropolitan Area Networks–Virtual Bridged Local Area Networks Amendment 14: Stream Reservation Protocol (SRP)," *IEEE Std 802.1Qat*, pp. 1–119, 2010.

[4] "IEEE Standard for Local and Metropolitan Area Networks– Virtual Bridged Local Area Networks Amendment 12: Forwarding and Queuing Enhancements for Time-Sensitive Streams," *IEEE Std 802.1Qav*, pp. 1–72, 2019.

[5] "IEEE Standard for Local and Metropolitan Area Networks – Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic," *IEEE Std 802.1Qbv*, pp. 1–57, 2016.

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

[7] "IEEE Standard for Local and Metropolitan Area Networks – Bridges and Bridged Networks," *IEEE Std 802.1Q*, pp. 1–2163, 2022.

[8] "IEEE Standard for Local and Metropolitan Area Networks–Frame Replication and Elimination for Reliability," *IEEE Std 802.1CB*, pp. 1–102, 2017.

[9] R. Serna Oliver, S. S. Craciunas, and W. Steiner, "IEEE 802.1Qbv Gate Control List Synthesis Using Array Theory Encoding," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 13–24, Apr. 2018.

[10] "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems," *IEEE Std 1588*, pp. 1–269, 2008.

[11] P. G. Kannan, R. Joshi, and M. C. Chan, "Precise Time-Synchronization in the Data-Plane Using Programmable Switching ASICs," in *ACM Symposium on SDN Research (SOSR)*, p. 8–20, Apr. 2019.

[12] P. Meyer, T. Häckel, S. Reider, F. Korf, and T. C. Schmidt, "Network Anomaly Detection in Cars: A Case for Time-Sensitive Stream Filtering and Policing," *ArXiv e-prints*, vol. abs/2112.11109, Dec. 2021.

[13] OpenSim Ltd., "A Quick Overview of the OMNeT++ IDE." https://omnetpp.org/documentation/ide-overview/. *Last accessed on 17.05.2023*.

[14] F. Luo, B. Wang, Z. Fang, Z. Yang, Y. Jiang, and K. Demertzis, "Security Analysis of the TSN Backbone Architecture and Anomaly Detection System Design Based on IEEE 802.1Qci," *Security and Communication Networks*, vol. 2021, Jan. 2021.

[15] S. Nsaibi, L. Leurs, and H. D. Schotten, "Formal and Simulation-based Timing Analysis of Industrial-Ethernet Sercos III over TSN," in *International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, pp. 1–8, Oct. 2017.

[16] Sercos International e.V., "Sercos Protocol Structure." https://www.sercos.org/technology/functions-and-features/protocol-structure/. *Last accessed on 23.05.2023*.

[17] "IEEE Standard for Local and Metropolitan Area Networks–Timing and Synchronization for Time-Sensitive Applications," *IEEE Std 802.1AS*, pp. 1–421, 2020.

[18] S. Szancer, P. Meyer, and F. Korf, "Migration from SERCOS III to TSN – Simulation Based Comparison of TDMA and CBS Transportation," in *International OMNeT++ Community Summit*, vol. 56, pp. 52–62, Oct. 2018.

[19] N. S. Bülbül, J. J. Krüger, and M. Fischer, "TSN Gatekeeper: Enforcing Stream Reservations via P4-based In-network Filtering," in *IFIP Networking Conference (IFIP Networking)*, pp. 1–8, June 2023.

[20] P4 Language Consortium, "GitHub: Behavioural Model Version 2 (BMv2)." https://github.com/p4lang/behavioral-model. *Last accessed on 03.06.2023*.

[21] B. Varga, J. Farkas, A. G. Malis, and S. Bryant, "Deterministic Networking (DetNet) Data Plane: IP over IEEE 802.1 Time-Sensitive Networking (TSN)," June 2021.

[22] B. Varga, J. Farkas, A. G. Malis, and S. Bryant, "Deterministic Networking (DetNet) Data Plane: MPLS over IEEE 802.1 Time-Sensitive Networking (TSN)," June 2021.

[23] C. ho Choi, T.-S. Kang, W. Choi, and T. Cheung, "Implementation and Performance Evaluation of 100Gb/s DetNet Packet Forwarding Engine," *International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 1013–1017, 2022.

[24] V. Addanki and L. Iannone, "Moving a step forward in the quest for Deterministic Networks (DetNet)," *IFIP Networking Conference (Networking)*, pp. 458–466, 2020.

[25] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. ElBakoury, "Ultra-Low Latency (ULL) Networks: The IEEE TSN and IETF DetNet Standards and Related 5G ULL Research," *IEEE Communications Surveys & Tutorials*, vol. 21, pp. 88–145, Sept. 2019.

[26] A. N. Abbou, K. Samdanis, and J. Manner, "URLLC in B5G Networks: Use Cases, TSN/DetNet Extension, and Pending Issues," *International Telecommunication Networks and Applications Conference*, pp. 25–30, 2023.

[27] M. A. Abuibaid, A. H. Ghorab, A. A. Saruhan, M. St-Hilaire, G. Parsons, J. Farkas, B. Varga, I. Moldován, M. Máté, and S. H. R. Naqvi, "Integration of DetNet/TSN Reliability Functions in 5G Systems: A Case Study and Measurements," *IEEE Conference on Standards for Communications and Networking (CSCN)*, pp. 369–375, 2023.

[28] S. Lindner, M. Häberle, and M. Menth, "P4TG: 1 Tb/s Traffic Generation for Ethernet/IP Networks," *IEEE Access*, vol. 11, pp. 17525–17535, Feb. 2023.

[29] P. Emmerich, S. Gallenmüller, G. Antichi, A. W. Moore, and G. Carle, "Mind the Gap – A Comparison of Software Packet Generators," in *ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pp. 191–203, May 2017.

[30] P. Pop, M. L. Raagaard, M. Gutierrez, and W. Steiner, "Enabling Fog Computing for Industrial Automation Through Time-Sensitive Networking (TSN)," *IEEE Communications Standards Magazine*, vol. 2, pp. 55–61, June 2018.

[31] M. L. Raagaard, P. Pop, M. Gutiérrez, and W. Steiner, "Runtime Reconfiguration of Time-Sensitive Networking (TSN) Schedules for Fog Computing," in *IEEE Fog World Congress (FWC)*, pp. 1–6, Oct. 2017.

[32] V. Gavriluţ and P. Pop, "Scheduling in Time-Sensitive Networks (TSN) for Mixed-criticality Industrial Applications," in *IEEE International Workshop on Factory Communication Systems (WFCS)*, pp. 1–4, June 2018.

[33] J. Heinanen and R. Guerin, "RFC2697: A Two Rate Three Color Marker," Sept. 1999.

[34] Edgecore Networks, "Wedge 100BF-32X." https://www.edge-core.com/productsInfo.php?cls=1&cls2=5&cls3=181&id=335. *Last accessed on 17.06.2023*.

[35] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker,

"P4: Programming Protocol-Independent Packet Processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, July 2014.
[36] The P4.org Architecture Working Group, "$P4_{16}$ Portable Switch Architecture (PSA)." https://p4.org/p4-spec/docs/PSA.html, Apr. 2021. *Last accessed on 20.05.2023* (working draft).
[37] P. Meyer, F. Korf, T. Steinbach, and T. C. Schmidt, "Simulation of Mixed Critical In-Vehicular Networks," in *Recent Advances in Network Simulation*, pp. 317–345, Springer International Publishing, 2019.
[38] F. Hauser, M. Häberle, D. Merling, S. Lindner, V. Gurevich, F. Zeiger, R. Frank, and M. Menth, "A Survey on Data Plane Programming with P4: Fundamentals, Advances, and Applied Research," *Journal of Network and Computer Applications (JNCA)*, vol. 212, Mar. 2023.
[39] The P4 Language Consortium, "$P4_{16}$ Language Specification." https://p4.org/p4-spec/docs/P4-16-v1.2.2.pdf, May 2021. *Last accessed on 14.05.2023*.
[40] Intel®, "$P4_{16}$ Intel® Tofino™ Native Architecture – Public Version." https://github.com/barefootnetworks/Open-Tofino/blob/master/PUBLIC_Tofino-Native-Arch.pdf, Apr. 2021. *Last accessed on 07.06.2023*.
[41] Chair of Communication Networks, University of Tuebingen, "GitHub: P4-PSFP." https://github.com/uni-tue-kn/P4-PSFP.
[42] T. Stüber, L. Osswald, S. Lindner, and M. Menth, "A Survey of Scheduling Algorithms for the Time-Aware Shaper in Time-Sensitive Networking (TSN)," *IEEE Access*, vol. 11, pp. 61192–61233, June 2023.
[43] X. Jin, C. Xia, N. Guan, C. Xu, D. Li, Y. Yin, and P. Zeng, "Real-Time Scheduling of Massive Data in Time Sensitive Networks With a Limited Number of Schedule Entries," *IEEE Access*, vol. 8, pp. 6751–6767, Jan. 2020.
[44] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, "Scheduling Real-Time Communication in IEEE 802.1Qbv Time Sensitive Networks," in *International Conference on Real-Time Networks and Systems (RTNS)*, p. 183–192, Oct. 2016.
[45] S. S. Craciunas and R. S. Oliver, "Out-of-sync Schedule Robustness for Time-Sensitive Networks," in *IEEE International Workshop on Factory Communication Systems (WFCS)*, pp. 75–82, 2021.
[46] M. Vlk, K. Brejchová, Z. Hanzálek, and S. Tang, "Large-scale Periodic Scheduling in Time-Sensitive Networks," *Computers and Operations Research*, vol. 137, Jan. 2022.
[47] "IEEE Standard for Ethernet–Amendment 8:Physical Layer Specifications and Management Parameters for 2.5 Gb/s, 5 Gb/s, and 10 Gb/s Automotive Electrical Ethernet," *IEEE Std 802.1ch*, 2020.

Fabian Ihle received his bachelor's (2021) and master's degrees (2023) in computer science at the University of Tuebingen. Afterwards, he joined the communication networks research group of Prof. Dr. habil. Michael Menth as a Ph.D. student. His research interests include software-defined networking, P4-based data plane programming, resilience, and Time-Sensitive Networking (TSN).

Steffen Lindner is a postdoctoral researcher specialized in software-defined networking (SDN), P4, time-sensitive networking (TSN), and congestion management. He studied, worked, and obtained his bachelor's (2017), master's (2019), and Ph.D. (2024) degrees at the University of Tuebingen.

Michael Menth, (Senior Member, IEEE) is professor at the Department of Computer Science at the University of Tuebingen/Germany and chairholder of Communication Networks since 2010. He studied, worked, and obtained diploma (1998), PhD (2004), and habilitation (2010) degrees at the universities of Austin/Texas, Ulm/Germany, and Wuerzburg/Germany. His special interests are performance analysis and optimization of communication networks, resilience and routing issues, as well as resource and congestion management. His recent research focus is on network softwarization, in particular P4-based data plane programming, Time-Sensitive Networking (TSN), Internet of Things, and Internet protocols. Dr. Menth contributes to standardization bodies, notably to the IETF.