TSN Gatekeeper: Enforcing stream reservations via P4-based in-network filtering

Nurefşan Sertbaş Bülbül, Joshua Jannis Krüger and Mathias Fischer
University of Hamburg, Germany
Email:{nurefsan.sertbas, joshua.krueger, mathias.fischer}@uni-hamburg.de

***Abstract*—Real-time communication is crucial for mission critical scenarios, such as industrial automation and automotive applications. To meet these applications' strict quality of service (QoS) requirements, a new set of specifications, known as time-sensitive networking standards (TSN), has been proposed. TSN requires pre-registration of data streams before actual communication to help guarantee bandwidth and ensure constrained end-to-end latency. However, this mechanism is vulnerable to traffic overload and denial of service (DoS) attacks. This paper proposes a P4-based dynamic attack filtering as a link-layer network function to defend TSN against malicious network elements, such as faulty talkers or switches, directly on the data plane. Our experiments indicate that our P4-based implementation can filter malicious traffic with minimal overhead and minimize frame losses for legitimate traffic.**

***Index Terms*—time-sensitive networks, programmable data planes, ingress filtering, per-stream policing, babbling idiots**

I. Introduction

Ethernet is a family of standards that enables high-throughput communication in diverse wired-networking environments. It has also been proposed for use in modern embedded environments with real-time requirements, such as industrial and in-car networks [2]. However, these new application domains have introduced new requirements that Ethernet was not initially prepared to satisfy. For instance, self-driving cars with hundreds of time-sensitive sensors create new challenges for the underlying Ethernet architecture, such as timely delivery and bounded latency guarantees. To address these challenges, the IEEE time-sensitive networking task group has proposed a set of standards that can be used when the application has no tolerance for frame loss due to congestion and guaranteed upper bounds on end-to-end latency [3]. These standards also allow the coexistence of traffic with different priorities, such as high-priority time-critical and low-priority best-effort traffic, to be sent over the same physical infrastructure.

Time-sensitive networking relies on end hosts' pre-registration of traffic requirements to allocate the necessary resources along the end-to-end path. With this, the end host agrees with the network before the actual communication, and thus the network can provide a certain level of QoS for the requested end host, the talker in TSN. Every entity in TSN is expected to obey its reserved resource limits, such as staying within the allocated bandwidth. However, this mechanism is vulnerable to malicious network elements, such as faulty talkers or switches. A malicious talker may send more traffic than it previously reserved, which may cause congestion at the switches on the path and can violate bandwidth guarantees on all streams, even the legitimate ones. This phenomenon is called the *babbling idiot*, and whether it is intentional or not, it must be avoided to maintain determinism in such a network.

To overcome this issue, the IEEE TSN Task group has proposed the IEEE 802.1Q Qci Per-Stream Filtering and Policing (PSFP) standard [1]. The standard introduces a cascaded filtering mechanism that blocks or limits excessive amounts of data to protect queues from DoS attacks. Moreover, it enables the application of fine-grained policing decisions. However, the filtering approach has yet to be explicitly defined; it is only conceptually defined.

In this paper, to dynamically enable such filtering functionality within the network at line rate, we leverage network programmability via P4 language, which has recently attracted attention from both the research community and the industry [4]. P4 enables the implementation of novel network functions for various use cases, such as fine-grained packet handling and advanced packet forwarding, even without a centralized controller. It can also handle packets at line rate dynamically and flexibly. This makes it possible for network designers to create fine-grained networks aware of the applications and data being delivered, allowing the network to meet mission-critical requirements such as latency assurances [5]. Thus, due to such benefits, P4 has a good potential to accelerate innovations in time-sensitive networks [6]. Accordingly, our contributions are:

- We present two ingress filtering mechanisms that comply with the concepts introduced in the IEEE 802.1 Qci standard to safeguard switch queues against DoS attacks. We leverage P4 to implement the proposed filtering functionality directly at the data plane, eliminating the need for a centralized controller
- We evaluate the effectiveness of our approach by conducting experiments using randomly generated network topologies. We compared the performance of our proposed filtering mechanisms with the scenario where no filtering policy is deployed. Our evaluation results indicate that our filtering approaches incur minimal overhead, even with an increasing number of attackers, and effectively mitigate the impact of DoS attacks on switch queues and prevent the loss of legitimate traffic.

The remainder of this paper is structured as follows: Section

ISBN 978-3-903176-57-7 © 2023 IFIP

II describes basic TSN mechanisms and summarizes the state of the art. In Section III, we introduce our overall architecture. We evaluate our approach and describe our evaluation results in Section IV. Finally, Section V concludes the paper and summarizes future work.

II. Background and Related Work

A. Time Sensitive Networking

In a typical time-sensitive network, the sender of the data, a talker, announces the desire to send data by sending a talker-advertise message that defines the traffic characteristics of the stream. In this context, a stream refers to a data flow between the sender and receiver, such as the talker and listener(s) in TSN, and is identified by a unique stream identifier (StreamId). The talker-advertise message is propagated over the network depending on the configuration scheme, either centralized or distributed. All listeners receive the message, and only the listener(s) interested in receiving the related stream replies. The reply message, listener-ready, is forwarded in the reverse direction of the talker-advertise message back to the talker. On a listener-ready message switches on the path, re-check whether the resources to guarantee fault-free transmission are available. If so, the resources are reserved, and the listener-ready message is forwarded to the next switch. Eventually, the listener-ready message reaches the talker, which initiates the process, and the stream transmission can begin. If a stream is no longer needed, the talker and listener can cancel the associated resources by sending a cancel message [7].

With this pre-configuration, required resources, e.g., bandwidth, are reserved before communication. So that network can provide a specific latency guarantee for the related transmission. Here it has been assumed that every TSN node obeys its reserved resource limits, e.g., stays within allocated bandwidth. However, when there is an unauthorized attempt to use resources, e.g., DoS attacks by flooding, the network may no longer provide the promised QoS. To deal with traffic exceeding its pre-defined limits, a threshold-enforcing can be applied, in which traffic up to the advertised limit is forwarded, and any exceeding traffic is blocked. It does have a clear benefit in certain situations where a temporarily faulty end-host, which returns to normal operations after a period of violation, could be kept in the network while effectively containing its threats. Alternatively, the stream can be entirely blocked when it exceeds the advertised limit. Unlike the first one, all traffic is dropped at the ingress, even if the stream would later returns to behave as advertised.

The IEEE task group has proposed IEEE 802.1 Qci standard for ingress filtering traffic that exceeds its registered bandwidth guarantees. This filtering standard helps maintain the established service quality for specified traffic and streams, protect queues from unwarranted traffic (e.g., deliberate DoS attacks), and mitigate the effects of bandwidth violations and malfunctioning. The standard proposes a cascaded layer of filtering and policing as shown in Fig. 1. In the first layer, stream filters decide which gates and meters will be responsible for the arriving frame. In other words, when a frame arrives at the switch's ingress, the stream filter matches the StreamId to a specific, though not necessarily dedicated, stream gate and flow meter for policing and filtering. The second layer, a stream gate, is a two-state filter. In the *open* state, frames can pass to the responsible flow meter for further filtering. In the *closed* state, frames will be dropped. The state can change on a schedule which can be carried out by defining the gate control list or based on interaction by the control plane. Lastly, the flow meters enable the deployment of more fine-grained algorithms and decide whether the frame is allowed to pass. After the flow meter allows a frame, it gets queued in the network node for remaining forwarding or processing. This filtering mechanism supports the following policing actions [8]:

- Time-based policing: This can be carried out using stream gates as it has two states: *open* and *closed*. Frames that arrive when the gate is closed are directly filtered (discarded) so that this mechanism aims to support ap-

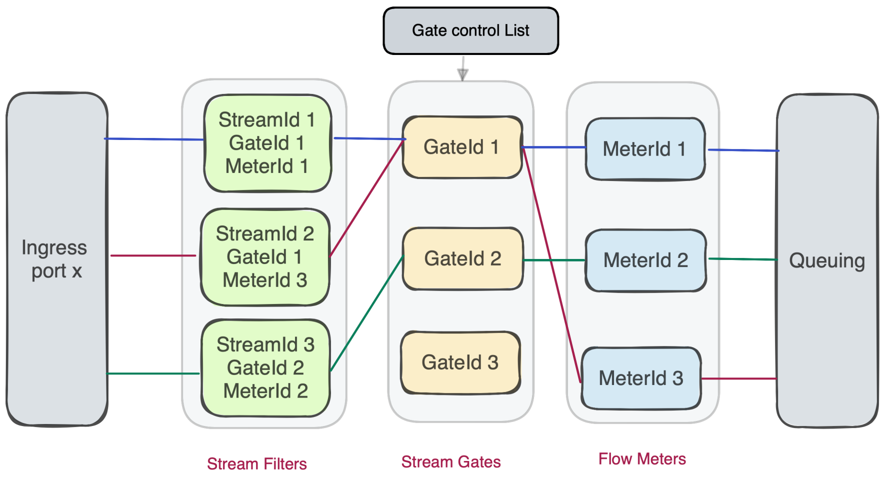


Fig. 1: IEEE 802.1Qci per-stream filtering and policing [1]

plications where the transmission and reception of frames across the network are coordinated.
- Rate-based policing: This can be carried out using flow meters by specifying frame rate parameters. Then, the meters apply to stream(s) and allow policing of streams that exceed the configured rate.
- Burst-based policing: This can be carried out using flow meters so that the length of the supported burst is set, and frames will be filtered accordingly.
- Frame length-based policing: This can be carried out using flow meters to filter frames based on the maximum frame lengths.

Authors in [9], propose an IEEE 802.1Qci-based attack detection system that applies filtering based on bandwidth and arrival time. For that, they propose a two-rate, three-color marker-based mechanism that can successfully drop illegitimate traffic. However, the configuration parameters of the presented approach are assumed to be set at the beginning and remain the same during the network's lifetime. Such a static configuration could be a valid assumption for specific use cases, e.g., in-vehicle networks. Still, such a solution would only partially benefit from the protocol's capabilities. Also, only the periodic, time-triggered traffic has been taken into account; but it needs to be clarified how to derive the filtering rules for aperiodic traffic, e.g., burst traffic. In [10], a filtering mechanism based on one of the existing traffic shapers in TSN, Credit Based Shaper, is proposed, and its compatibility with the 802.1Qci filtering is shown via OMNeT++ simulations. However, feasibility analysis with other traffic shapers is left as a future work.

B. P4 based Security

Several studies exist in the literature to add such attack filtering functionality to the network, either using centralized or distributed configuration options. Here, software-defined networks (SDN) could be an option with the global network view of the centralized SDN controller [11]. However, a centralized control plane can easily be a bottleneck in many scenarios, especially for recovery and connectivity protection [12], [13]. Also, the communication delay between data and the control plane may limit the reaction time of the controller as well [14]. Thus, such an implementation suffers from centralized controller dependency and causes scalability problems. Besides, standard SDN-based solutions are able to configure only per-flow forwarding policies and do not allow per-packet priority enforcement. Thus, these shortcomings of the SDN further motivate researchers to use the P4-based data planes, which enable processing packets at a line rate and remove the need for a centralized controller. Moreover, unlike the OpenFlow protocol, which is used as a standard protocol between controller data plane communication in SDN, P4 enables the definition of more fine-grained packet fields and, therefore, deployment of more use case-specific security solutions.

P4 has been used frequently in the literature to bring security functionality to the data plane and reduce the experienced latency by removing the remote controller involvement [15]. Using P4-based data planes in blockchain architecture, authors in [14] show that several attacks on the blockchain, including DoS, can be discovered before transaction packets get to the control plane. In [16], P4-based data paths are used to minimize latency and add security monitoring functionality for industrial 5G networks. Their results show that latency can be decreased by around half. In [17], authors use P4 to improve the routers to filter router spoofing and man-in-the-middle attacks directly on the data plane. Authors in [18] use P4 to detect spoof and volumetric attacks close to the source in order to protect the network. Their approach is scalable and controller-independent, thanks to the P4.

The flexible configuration options of the P4-based switches, either centralized or distributed, make P4 a promising solution for mission-critical networks. Thus, we believe that combining the merits of IEEE 802.1Qci and P4 marks a significant step in protecting future time-sensitive networks.

III. P4-BASED INGRESS FILTERING

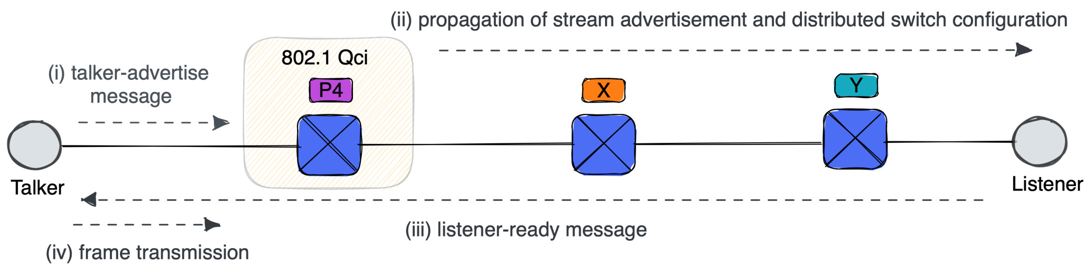
This section introduces our P4-based ingress filtering approaches for TSNs and how we dynamically deploy that strategy in time-sensitive networks. We first describe the overall frame filtering procedure, and afterward, we explain our filtering approaches in detail.

A. Overall System

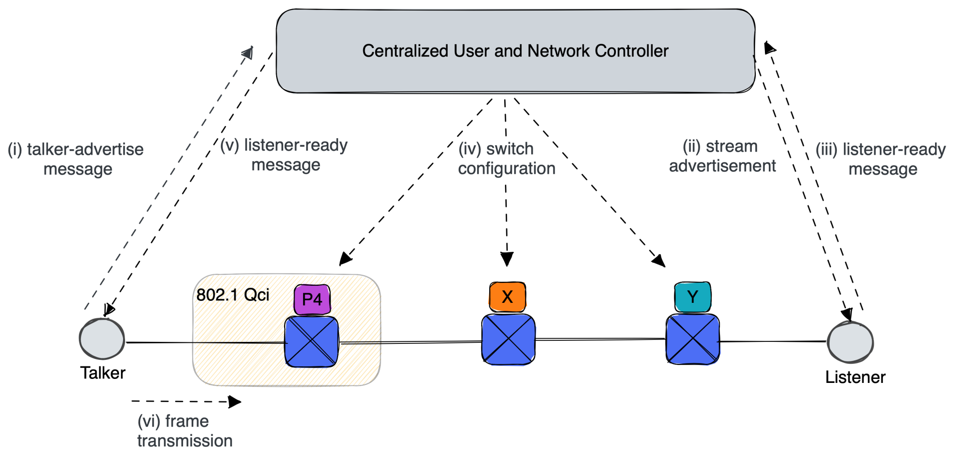
In time-sensitive networks, as explained in Section II-A, talkers must inform the network about the required resources before initiating transmission. Once the reservation is made, it is crucial to ensure that the talker complies with the declared traffic requirements. To achieve this, we propose a P4-based ingress filtering solution designed to run on the programmable data plane of a P4-enabled TSN switch. The solution uses P4 to implement a lightweight firewall at the edge of the TSN, which protects switches from being attacked or overloaded.

We envision the proposed P4-based filtering solution as a link-layer network function, as there could be other functions for different purposes, such as intrusion detection, load balancing, etc. (represented by X and Y in Fig. 2). The P4-based filtering solution can work independently at the edge switch without a centralized controller. However, it is still possible to configure P4-enabled TSN switches with different policies using a centralized network controller. We illustrate the overall architecture in Fig. 2 for both distributed and centralized configuration architectures, as our P4-based filtering approach works in both cases.

In the distributed architecture, a talker communicates with the edge switch to declare its traffic requirements (i), and the switch forwards the requirements to the other core switches in the network (ii). Here, switches are not configured by a central entity but in a distributed manner with their local knowledge. Our P4-based ingress filtering approach can also be configured as it derives filtering rules from the stream reservation messages. Then, interested listeners will subscribe to that stream (iii), and the talker will be informed to start transmission (iv). In the centralized architecture, the talker


(ii) propagation of stream advertisement and distributed switch configuration
(i) talker-advertise message
802.1 Qci
P4
X
Y
Talker
Listener
(iii) listener-ready message
(iv) frame transmission

(a) Distributed configuration architecture


Centralized User and Network Controller
(i) talker-advertise message
(v) listener-ready message
(iv) switch configuration
(ii) stream advertisement
(iii) listener-ready message
802.1 Qci
P4
X
Y
Talker
Listener
(vi) frame transmission

(b) Centralized configuration architecture

Fig. 2: P4-based ingress filtering as a link-layer network function for TSNs

and listener communicate directly with the centralized user and network controller (i, ii). Then, the central controller sends switch configurations to the switches (iii). Here, the centralized controller can specify filtering rules and configure the ingress filtering module accordingly. Lastly, the talker is informed of the transmission, and then it starts stream transmission (v).

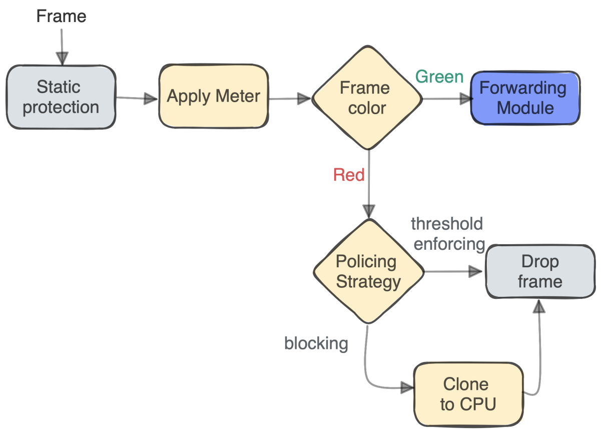
The presented filtering approach has two main blocks static and dynamic. We present a static mechanism as a first checkpoint, including maximum frame size and ingress port verification. Here, frames exceeding the maximum size are dropped to avoid possible switch congestion. Since attackers can still flood frames with the spoofed StreamIds to block or harm the transmission of other (legitimate) talkers, it is also necessary to verify the ingress port. After the initial verifications as a second checkpoint, we deployed our dynamic filtering solutions, namely *metered ingress filtering* and *gated ingress filtering*, as we describe in the following sections.

B. Metered Ingress Filtering

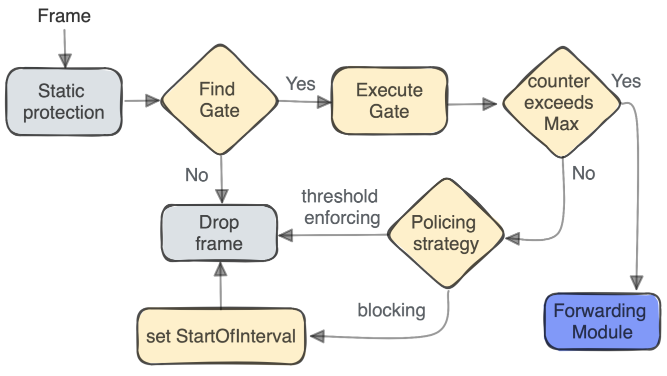
In the deployment of metered ingress filtering, we benefit from the portable switch architecture (PSA), which is a target architecture that defines standard data types, counters, meters, and other externs that P4 programmers can use as required. The P4 language design aims to maintain minimal consensus between switch vendors, excluding extended features. However, switch vendors can utilize architecture definitions to support more features and enable rapid innovation and proof of concepts before all parties accept them. Here, the PSA makes such P4 programs portable across different targets.

Based on the PSA primitive, we use direct meters derived from the RFC 2698 [19]. Conceptually, they are similar to the buckets and are defined by the initial burst size (BS). Then, the bucket size is increased by the pre-configured information rate (IR) times per second and decreased on the arrival of a packet. Here, the IR parameter can be computed based on the talker-advertise message as it represents the number of frames per measurement interval. The burst size can be interpreted as the number of frames by which a stream is allowed to exceed the advertised rate. Such a parameter would be helpful in case of frame delays that may normally require the frame, even the legitimate streams. If the bucket size falls below zero, packets are marked red, while otherwise, they are green. We use this mechanism for filtering by attaching a direct meter to the forwarding table, which is automatically executed in case a matching entry exists. The burst size and information rate parameters can be configured per table entry via the P4Runtime API, which conceptually stands as a control plane. Note that that is the control plane of the P4, not a centralized controller.

After applying the meter and marking the frame as red or green, frames are handled differently depending on the filtering



(a) Flowchart of the metered ingress filtering



(b) Flowchart of the gated ingress filtering

Fig. 3: Flowcharts of the proposed filtering approaches

strategy, either threshold-enforcing or blocking, as shown in Fig. 3a. In the threshold-enforcing strategy, a frame is directly dropped if marked as red. Otherwise, if the frame is marked as green, we forward them to the egress to be forwarded to the next hop. In the blocking strategy, red-marked frames are cloned to the CPU port, and the control plane is notified. Then, the control plane removes the registration of this stream, and the frame will be dropped. The green marked frames are handled similarly as in threshold-enforcing.

C. Gated Ingress Filtering

Another data type that the P4 language supports are registers which serve a general purpose and ease the implementation of fully customized algorithms. However, unlike the meters, it is not possible to use registers as per-table-entry or per-stream. Thus, we use a concept called *gates*, which works like a per-class filtering approach in the IEEE 802.1Qci standard. A gate merges the traffic characteristics of multiple streams and handles them as a single stream. Using more gates will allow more fine-grained filtering while increasing the memory requirements. Another critical point here is since it does not enforce per-stream policies, as long as the sum of the traffic at that gate is not exceeded, it does not limit the transmission of any stream. In other words, the allowed transmission capacity for that gate may not be shared equally along the streams assigned to the same gate. However, that is still reasonable as we deal with class-based queuing delays in the TSN.

The basic flowchart of the gated ingress filtering approach is shown in Fig. 3b. When a frame arrives at the switch, a static check is performed as described previously. Then, it looks for a predetermined gate for the stream. If there is no such configuration, the frame will be dropped directly. Otherwise, it fetches the other gate configuration parameters using its gate identifier, GateId.

A core pillar of the gate mechanism is the gate selection function, as it decides which stream is handled by which gate and directly affects filtering. In order to design a *good* gate selection function, there are some issues to be addressed. For instance, deterministic algorithms ease the prediction of the GateId; an attacker with sufficient knowledge can abuse this mechanism to target specific streams by injecting traffic to that gate. Thus, a hash function like a gate selection function would not be appropriate. Another problem can be using large frames as the frame size is checked independently from the gate so that an attacker sending a low number of frames with large frame sizes can inject his/her frames in a gate with streams that have the opposite characteristic and exceed bandwidth by a high degree. Forcing all streams to adhere to the same maximum frame size is a solution but one unsuitable for practical use. A non-deterministic gate selection algorithm is expected to sufficiently mitigate this attack angle, as it does not allow the specific targeting of a gate with a large number of allowed frames.

An important design principle is finding a proper gate selection function to address all mentioned drawbacks. For simplicity, we left the gate selection function selection out of this paper's scope and used a simple strategy: fill empty first (FEF). Thus, it fills empty gates first so that any stream violation will have a limited effect on others as it is also limited to a particular gate. Such an approach may suffer if many streams exist on the gate or the attacker advertises early.

After executing the related gate, a frame counter which is increased for every frame arrival is checked. Here the *Max* value needs to be calculated and configured by the controller. Since different traffic classes in TSN send traffic at varying intervals and a varying number of frames, we used a common observation interval and recalculated the number of frames for that common observation interval. Therefore, this counter checkpoint behaves as a bandwidth check for the gate, which rejects and drops the frame if it exceeds the predefined bandwidth. Unlike metered ingress filtering, the blocking mechanism involves no control plane; gates can be closed at a line rate. This also means less memory as it does not require CPU cloning.

D. Compatibility with the IEEE 802.1Qci Standard

The filtering mechanisms presented in this paper align with the IEEE 802.1Qci standard. In the metered ingress filtering with thresholding, all three steps of the PSFP pipeline are handled by a single table. The forwarding table has an attached colored meter. Therefore, it is both a stream filter and a flow meter. In this case, the relationship between stream filters and flow meters is one-to-one, which is not required, but explicitly allowed in PSFP. The stream gate does not have a technical expression, but conceptually it can be interpreted as always in the *open* state. The metered ingress filtering with a blocking mechanism sends a notification message to the controller once the colored meter is triggered. Then, the associated table entry from the forwarding table is deleted. The PSFP pipeline can conceptualize this by the control plane setting the stream gate to the *closed* state.

In the gated ingress filtering approach, the stream gates are always in the *open* state. Otherwise, the approach is aligned with the PSFP structure. As described in Section III-C, there are two functions, find-gate and execute-gate, which are implemented as tables and are precisely aligned in both the conceptual and the technical sense to the stream filter and the flow meter, respectively. The arbitrary algorithm the flow meter executes is based on a counter reset with each measurement interval and differs between the threshold-enforcing and blocking variants.

IV. Evaluation

In this section, we measure the performance of presented filtering approaches for time-sensitive networks and compare them against the normal case when no filtering solution is applied. For that, we briefly explain the evaluation setup and then evaluate the performance of filtering approaches regarding frame loss and end-to-end latency of frames.

A. Setup

We have implemented the presented dynamic filtering approaches on the P4 behavioral model version (bmv2) and emulated the attacks on Mininet. Note that the employed software switch bmv2 was not designed for performance evaluations and is not necessarily representative of the performance of the mechanisms on a different target. For example, table access could take significantly longer (or shorter) time in a hardware switch. This could significantly influence the results; however, the relative results will remain the same. Apart from emulating the packet processing logic of a P4-Switch on bmv2, we have implemented the control plane in Python 3.6.9. It further interacts with the data plane using the P4Runtime API. All experiments were run on a dual-core Intel(R) Core(TM) i5-7200U CPU with 2.50GHz and 8GB of DDR4 RAM. The machine runs Ubuntu 18.04.6.

As topology, we use a ring topology containing four switches as ring topologies are commonly used in embedded networks, such as cars and factories; it is also common in TSN [20]. As attack traffic, we use a simple attack model known as a babbling idiot in the literature [21], [22]. In the TSN context, a babbling idiot is a talker who correctly advertises traffic and receives a corresponding listener-ready message but then sends more traffic than advertised and exceeds the allocated bandwidth. We randomly placed talkers, listeners, and babblers in the network. As a TSN traffic, we generated 20 random traffic scenarios, including isochronous, cyclic, event-triggered, and best-effort traffic, which are typical TSN traffic classes as described in [8] as follows:

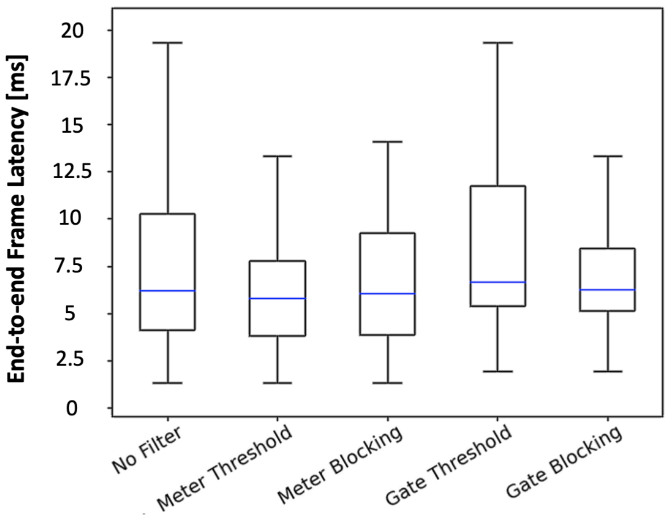
- **Isochronous Traffic:** It is a periodic traffic that requires reserving resources before its period ends. Thus, it can be characterized by an interval and a number of frames per interval. An example of isochronous traffic could be a distance sensor in a car, whose values are constantly required by the emergency brake assistant.
- **Cyclic Traffic:** It is also as periodic as isochronous traffic but contains a fixed length of idle times in between. It sends n frames in every x seconds at an isochronous rate r. An example of cyclic traffic could be a timed sensor or device measurement report.
- **Event Traffic:** It generates single frames sporadically at non-predictable and non-uniform intervals. Network control messages or user input can be an example of such traffic.
- **Best-Effort Traffic:** Most time-sensitive networks also allow a portion of best-effort traffic, i.e., traffic for which the network makes no guarantees regarding arrival or maximum latency. It is sporadic traffic that can be modeled as a random burst.

B. Results

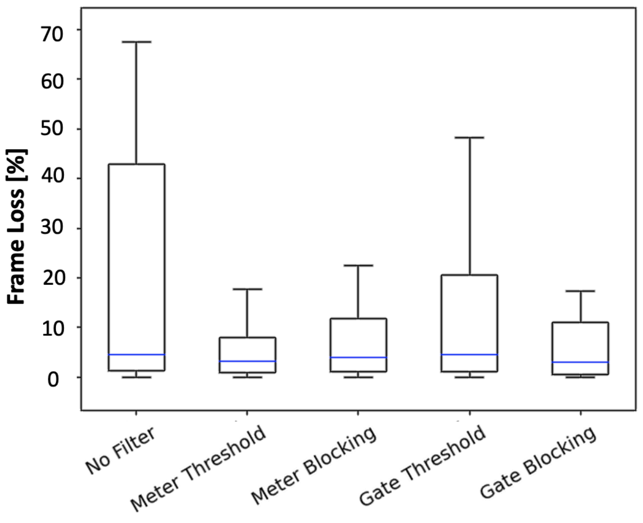
This section evaluates the performance of the presented filtering approaches, Meter, and Gate configured with either Thresholding or Blocking. Then, these approaches are compared with the case there is no filtering applied in terms of their frame loss rate and end-to-end frame latency.

Filtering performance on the delivery of TSN traffic: In order to compare the filtering performance of the presented strategies, we measure the frame loss rate and end-to-end frame latency, which we aim to minimize for legitimate traffic with ingress filtering. Since the results would be highly affected by the type of traffic, we generate 20 scenarios to make a fair comparison between approaches. For that, we generate 60% isochronous traffic with a period of 10 ms, 20% cyclic traffic with a 1-5 s period, 15% event-traffic with a period between 200 ms-1 s, and finally, 5% best-effort-traffic. Results are shown in Fig. 4.

As shown in Fig. 4a, even though the average latencies do not differ significantly between the tested approaches, meter-based filtering, thresholding, or blocking has lower and more bounded end-to-end latencies, which has essential significance in TSN. Also, it might not be sufficient to look only at the latency values as it shows only the end-to-end latency of successfully transmitted frames. Due to the babblers, we see a high number of frame losses in Fig. 4b. It would be misleading to look only at the averages as the TSN promises a certain



(a) End-to-end latency for mixed traffic scenarios



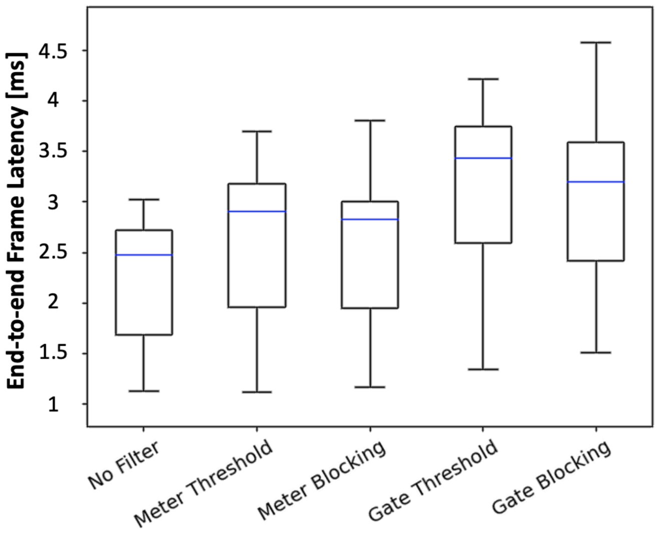
(b) Frame loss for mixed traffic scenarios

Fig. 4: Filtering performance on the delivery of TSN traffic

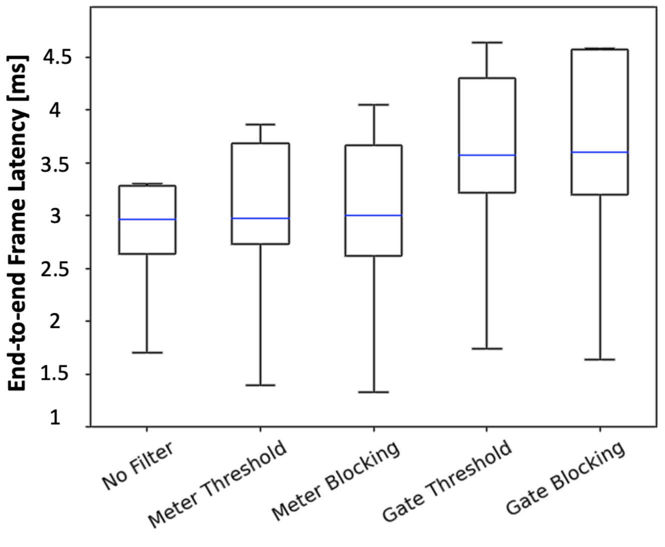
QoS; the worst-case frame loss rate must also be considered. When *No Filter* is applied, the babblers will significantly affect legitimate streams, and they may experience frame losses up to 68%. It is important to note that the maximum frame loss rate in any filtering approach is well below this value. Here, even in the worst case, we could still say that filtering approaches decrease the frame loss rate of the legitimate streams to ≈24%(as in the upper bound of meter-blocking) and ≈48% (as in the upper bound of gate-thresholding). Thus, it is clear that for some scenarios, they have a huge benefit.

Filtering performance regarding an increasing number of babblers: To analyze the filtering performance in case of an increasing number of bubblers, we simulate single and multi-babblers scenarios and measure how it affects the end-to-end latency of the TSN traffic. For the metered filtering, we set the burst size to 100. In gated filtering, we set the number of gates to 64, and we assumed that if the frame is delayed, we can still tolerate that, and we do not block that stream completely. For that, we set the number of tolerated exceeding frames to half the number of streams currently in the gate. Results are shown in Fig. 5.

Filtering adds additional latency to the frame processing; thus, results in Fig. 5a may be interpreted as the difference between *No Filter* and filtering solutions due to the filtering overhead. However, it should be noted that there is also a babbler in the given test scenario, which further delays the frames of the legitimate stream. To clarify, we repeat the experiment by increasing the number of babblers to three, as



(a) End-to-end latency *with* a babbler.



(b) End-to-end latency *with* 3 babblers.

Fig. 5: Effect of babblers on the end-to-end latency

shown in Fig.5b. The additional latency the multiple babblers caused is minor, around 0.5 milliseconds for the *No Filter* case. For the filtering approaches, the effect of babblers is noticeable but very minor, as we expected. Another result that can be derived from here is the overhead of blocking mechanisms due to cloning to CPU, which also increases as the number of babblers increases. Therefore, they are no longer better than the thresholding mechanisms, as in Fig 5a.

V. Conclusion and Future Work

This paper proposes P4-based dynamic ingress filtering approaches for securing time-sensitive networks from denial-of-service attacks. We proposed a *metered filtering mechanism* that operates per stream and achieves low latency results, even for high traffic demands. Alternatively, we also proposed a *gated filtering mechanism* that fits the per-class filtering concept and enables the deployment of more customizable algorithms. We tested the presented approaches in an emulated mininet environment, and the results show that our filtering approaches can limit frame loss rates of legitimate traffic significantly with only a minimal filtering overhead. Thus, the proposed approaches have the potential to meet strict performance requirements in time-sensitive environments.

As part of our future work, we plan to expand our implementation by incorporating intrusion detection and incident reporting functionality. This enhancement would enable the controller to receive notifications of detected violations, triggering related mitigation mechanisms. Additionally, the PSFP is a proactive approach typically deployed in fixed network positions with a fixed capacity. A promising research direction would be to investigate filtering the attacks that exceed the switch's capacity, for which SDN/NFV-based reactive solutions [23] seem promising as they can flexibly position security functionalities in the network. Moreover, the autonomous configuration of PSFP is another future research direction that aligns perfectly with the concept of self-configured TSN [24]. By adapting related parameters based on changing network conditions, the PSFP's effectiveness could be further enhanced.

References

[1] T. Jeffree, *P802.1Qci – Per-Stream Filtering and Policing*, Sep 2017. [Online]. Available: https://1.ieee802.org/tsn/802-1qci/

[2] S. A. Nsaif and J. M. Rhee, "Seamless ethernet approach," in *2016 IEEE International Conference on Consumer Electronics (ICCE)*, 2016, pp. 385–388.

[3] N. Finn, "Introduction to time-sensitive networking," in *IEEE Communications Standards Magazine*, vol. 2.2, 2018, p. 22–28.

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

[5] R. Bifulco and G. Rétvári, "A survey on the programmable data plane: Abstractions, architectures, and open problems," in *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*, 2018, pp. 1–7.

[6] N. Nayak, U. Ambalavanan, J. M. Thampan, D. Grewe, M. Wagner, S. Schildt, and J. Ott, "Reimagining automotive service-oriented communication: A case study on programmable data planes," *IEEE Vehicular Technology Magazine*, pp. 2–12, 2023.

[7] *Standard for Local and Metropolitan Area Networks - Virtual Bridged Local Area Networks - Amendment: 9: Stream Reservation Protocol (SRP)*, 2010. [Online]. Available: https://www.ieee802.org/1/pages/802.1at.html

[8] "Time sensitive networks for flexible manufacturing testbed characterization and mapping of converged traffic types," Mar 2019. [Online]. Available: https://hub.iiconsortium.org/portal/Whitepapers/5eb04d87d2df3f001102b6fe

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

[10] P. Meyer, T. Häckel, F. Korf, and T. C. Schmidt, "Dos protection through credit based metering - simulation-based evaluation for time-sensitive networking in cars," *Proceedings of the 6th International OMNeT++ Community Summit*, 2019.

[11] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[12] R. Kandoi and M. Antikainen, "Denial-of-service attacks in openflow sdn networks," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2015, pp. 1322–1326.

[13] D. Merling, W. Braun, and M. Menth, "Efficient data plane protection for sdn," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. IEEE, 2018, pp. 10–18.

[14] A. Yazdinejad, R. M. Parizi, A. Dehghantanha, and K.-K. R. Choo, "P4-to-blockchain: A secure blockchain-enabled packet parser for software-defined networking," *Computers & Security*, vol. 88, p. 101629, 2020.

[15] Y. Gao and Z. Wang, "A review of p4 programmable data planes for network security," *Mobile Information Systems*, vol. 2021, pp. 1–24, 2021.

[16] K. Gökarslan, Y. S. Sandal, and T. Tugcu, "Towards a URLLC-Aware Programmable Data Path with P4 for Industrial 5G Networks," in *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*, 2021, pp. 1–6.

[17] M. Mönnich, N. S. Bülbül, D. Ergenç, and M. Fischer, "Mitigation of IPv6 Router Spoofing Attacks with P4," in *Proceedings of the Symposium on Architectures for Networking and Communications Systems*, ser. ANCS '21. New York, NY, USA: Association for Computing Machinery, 2022, p. 144–150.

[18] G. Simsek, H. Bostan, A. K. Sarica, E. Sarikaya, A. Keles, P. Angin, H. Alemdar, and E. Onur, "DroPPPP: A P4 Approach to Mitigating DoS Attacks in SDN," in *Information Security Applications*, I. You, Ed. Cham: Springer International Publishing, 2020, pp. 55–66.

[19] D. J. Heinanen and D. R. Guerin, "A Two Rate Three Color Marker," RFC 2698, Sep. 1999. [Online]. Available: https://rfc-editor.org/rfc/rfc2698.txt

[20] D. Hellmanns, A. Glavackij, J. Falk, R. Hummen, S. Kehrer, and F. Dürr, "Scaling tsn scheduling for factory automation networks," in *2020 16th IEEE International Conference on Factory Communication Systems (WFCS)*, 2020, pp. 1–8.

[21] G. Buja, A. Zuccollo, and J. Pimentel, "Overcoming babbling-idiot failures in the FlexCAN architecture: a simple bus-guardian," in *2005 IEEE Conference on Emerging Technologies and Factory Automation*, vol. 2, 2005, pp. 8 pp.–468.

[22] O. Daniel and O. Roman, "Fault injection framework for assessing fault containment of ttethernet against babbling idiot failures," in *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, 2018, pp. 1–6.

[23] N. S. Bülbül and M. Fischer, "SDN/NFV-based DDoS Mitigation via Pushback," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020, pp. 1–6.

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