# PSFP in TSN Networks: Insights into Some Practical Limitations

Matheus Ladeira[1], Marc Boyer[2], Nicolas Navet[3] and Léo Seguin[1]

[1]*Realtime-at-Work, Nancy, France*
matheus.ladeira@realtimeatwork.com, leo.seguin@realtimeatwork.com

[2]*DTIS, Onera, Université de Toulouse, 31000, Toulouse, France*
marc.boyer@onera.fr

[3]*University of Luxembourg & Cognifyer, Esch-sur-Alzette, Luxembourg*
nicolas.navet@uni.lu

**ABSTRACT**

Per-Stream Filtering and Policing (PSFP), standardized in IEEE 802.1Qci, is a mechanism for providing fault containment in Time-Sensitive Networking (TSN) networks. This paper examines limitations of PSFP, showing that the fault-containment can be insufficient. In particular, the Flow Meter inside PSFP measures traffic in Service Data Unit (SDU) bytes – i.e., from the MAC destination address through the Frame Check Sequence. However, common Ethernet shapers such as the Credit-Based Shaper (CBS) regulate traffic based on the full 'on-wire' packet length, which includes the SDU plus the 8-byte preamble and the 12-byte inter-frame gap. This results in a 20-byte per-frame gap that increases admissible rates: with minimum-size packets, a talker can exceed its contractual bandwidth by up to 30%. In addition to the contract not being enforced, an independent stream might be penalized due to a queue build up. Through simulations with RTaW-Pegase software and hardware-in-the-loop experiments on a TSN testbed, we quantify these effects and evaluate configuration-level mitigations. We then discuss possible evolutions of the standard, including overhead-aware byte counting, which could address these gaps.

## 1. INTRODUCTION

Time-Sensitive Networking (TSN) has emerged as the de-facto toolbox for mixed control, audio-video and best-effort traffic on a shared Ethernet backbone. Standards such as 802.1Qbv (Time-Aware Shaper – TAS) and 802.1Qav (Credit-Based Shaper – CBS) provide latency and bandwidth guarantees. In this landscape, Per-Stream Filtering and Policing (PSFP) defined by IEEE 802.1Qci is often promoted as a "safety net" that converts a well-engineered TSN configuration into a fault-free network: any stream that violates the contract is assumed to be dropped before it can harm others.

PSFP doesn't actually prevent faults: instead, it limits their impact using traffic filters that discard individual packets or that entirely block streams. As we will show, those mechanisms, in some cases, are not 100% effective. One mechanism is especially problematic: the byte-count mismatch [1] between the PSFP's Flow Meter (which considers only the bytes between the MAC addresses and the Frame Check Sequence) and the *idleSlope* parameter for a CBS (which considers the full on-wire packet length including the Ethernet preamble and the Inter Frame Gap). The 20-byte gap per frame especially affects streams that use small packets, potentially inflating their throughput by as much as 30%. This issue is explored in detail in Section 3.

The purpose of this article is to provide an analysis of this limitation, showing a practical example of how it can present itself in a real network. Then, we outline possible solutions to avoid the issue, ranging from parameter tuning to possible standard amendments.

The remainder of the paper is organized as follows: Section 2 recaps relevant TSN concepts, notably the PSFP mechanism. Section 3 explains the byte-counting mismatch. Section 4 presents a practical example of how the mismatch may cause issues, both with a numerical simulation using RTaW-Pegase and with an implementation on a hardware test-bench. Section 5 presents mitigation strategies that can be already adopted in the current situation. Section 6 discusses possible changes to the current standard and products' implementation that could mitigate the issue. Then, Section 7 concludes the article, with perspectives for future research on the topic.

## 2. BACKGROUND

### 2.1. Ethernet Packet, SDU and MSDU

An Ethernet Packet is composed of several parts, which are depicted in Figure 1. The preamble consists of 8 bytes, the Interpacket Gap (IPG) consists of (at least) 12 bytes, and the Service Data Unit (SDU) makes up the Ethernet Frame. The whole Ethernet packet overhead with respect to the SDU is 20 bytes.

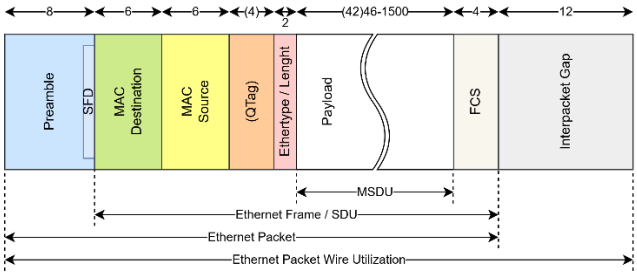


Figure 1: Structure of an Ethernet packet. The SDU spans from the MAC destination address to the Frame Check Sequence (FCS), excluding the 8-byte preamble and 12-byte inter-packet gap (IPG), which make up 20-byte media overhead sent on the wire but not accounted for by the PSFP Flow Meter.

The SDU itself is composed of several parts: 6 bytes of a destination MAC address, another 6 bytes for a source MAC address, optional 4 bytes of a Q-Tag in case VLANs are used, 2 bytes encoding the type of payload (EthType), the payload itself (also known as the MAC-layer SDU, or MSDU, which must be at least 46 bytes long and at most 1500 bytes long, although frames with a Q-Tag present can have a payload as short as 42 bytes [IEEE 802.1Q, Annex G]), and finally 4 bytes for a Frame Check Sequence (FCS). Therefore, the overhead of the SDU with respect to the MSDU is 18 bytes (or 22, if a Q-Tag is present). The SDU's minimum size is 64 bytes whether a Q-Tag is present or not, and its maximum size is 1518 bytes without a Q-Tag, and 1522 with it.

To calculate the full on-wire length of a frame of MSDU size of $N$, in bytes:

1. First, compute the SDU length $L$: $L = N + 12$ (MAC destination + MAC source) + 2 (EthType) + 4 (FCS) + 4 (Q-Tag, optional) + padding (if necessary, so that $L \geq 64$ bytes). In other words, $L = N + 22$ bytes + possible padding, considering there is a Q-Tag since we are working with VLANs.
2. Then, add the 8-byte preamble and 12-byte IPG, so that the full Ethernet packet length on the wire $= L + 20$ bytes.

### 2.2. TSN Tool-Chain Recap

Time-Sensitive Networking (TSN) is built as an extension to standard Ethernet (IEEE 802.3) via a sequence of IEEE 802.1Q amendments that, when combined, guarantee bounded latency, jitter, and fault containment. Figure 2 illustrates how a packet is processed through a TSN-enabled bridge or switch [2].

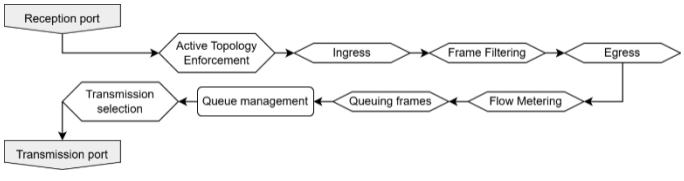


Figure 2: Processing steps in a TSN-enabled bridge. Each incoming packet traverses a sequence of TSN functions before reaching the output port.

In TSN, data flows are called "streams", and must have a unique identifier. This identifier is used for filtering, metering, etc.

#### 2.2.1. 802.1Qbv – Time-Aware Shaper (TAS)

The Time-Aware Shaper (TAS) enforces a time-division multiple access (TDMA) schedule by gating traffic open or closed according to a Gate Control List (GCL) running on a synchronized time base (IEEE 802.1AS). It does not meter bytes itself. Instead, it ensures that packets of a given class can only be transmitted during their assigned time window, as defined by the periodic *CycleTime* and each GCL entry's *GateOpenInterval* and *GateCloseInterval*. To prevent interference, when the TAS gate assigned to a high-priority queue is open, the gates for other queues are typically closed, a configuration known as "exclusive gating".

Once all GCLs have been configured in an adequate way, TAS guarantees real-time latency and bounded jitter to each frame. Nevertheless, TAS configuration is a hard problem, especially when the load increases [3] [4].

Except for very specific cases, it is, to the best of our knowledge, common practice to assign exclusive time windows to only 10 to 30% of the link capacity to TAS.

Moreover, to minimize end-to-end latencies, the use of TAS requires a synchronization between the applications and the network. Indeed, since some time intervals are devoted to frames, the frames must be ready when the window opens. This implies that the schedule of the tasks and the network must be co-designed. This use-case – supporting data flows with strict latency and jitter requirements – is one of the primary applications of TAS.

#### 2.2.2. Qav – Credit-Based Shaper (CBS)

The Credit-Based Shaper (CBS) was developed for Audio-Video Bridging (AVB) and provides per-class bandwidth reservation and bounded delays. Each output queue can be managed by a CBS shaper. Its most important shaping parameter is the *idleSlope*, which specifies the long-term transmission rate in bytes per second. The aim of CBS is to do *as if* the queue was connected to a link whose throughput is equal to the *idleSlope* (when we consider a sufficiently large time window). CBS guarantees a minimal service for the shaped queue while limiting its burstiness, leaving room for frames in lower priority queues. This is done using one single counter per queue, the *credit*. Note that some deviations between CBS and an ideal isolated link with configured throughput exist, but this is beyond the scope of this article [5].

Whenever a frame is eligible for transmission, CBS checks that the credit bucket is not negative. If this is the case, the frame transmission starts immediately, and credit is progressively reduced during the transmission by an amount proportional to the frame's full on-wire length (SDU + 20), possibly reaching negative values.

When the traffic class is not transmitting, if the credit is negative or if there are frames waiting in the queue to be sent, the credit amount increases at a rate defined by the *idleSlope*. The credit

will stop increasing when it reaches zero if there are no frames waiting in the queue, but it may become larger than zero if there is a frame in the queue that needs to wait for the end of transmission of a frame from another traffic class. Because CBS considers the full Ethernet-packet size per frame, the long-term transmission rate approaches the one a real link would have if its transmission rate were equivalent to the *idleSlope*.

To guarantee bounded delays and prevent buffer overflows during traffic bursts, the idleSlope of a CBS-shaped queue must be strictly greater than the average throughput of the traffic class it shapes [7]. It must be set so that, over time, the shaper accumulates enough credit to transmit possible bursts without violating delay constraints.

For example, to illustrate this behavior, in Figure 3 we represent a non-CBS frame (depicted in green) that is being transmitted when a burst of CBS-shaped messages (depicted in blue) arrives in the sending queue. The credit, represented as the red line, which was zero in the beginning, starts increasing in the moment the CBS-shaped burst arrives, with a rate defined by the *idleSlope*, until the non-CBS frame is done and the first of the CBS frames starts its transmission. From that moment onward, the CBS credit decreases at a rate equal to the link's transmission rate minus the *idleSlope*. Once the frame completes its transmission, the credit starts increasing again at a rate equal to the *idleSlope*. Although CBS-shaped frames are already waiting in the queue, they cannot be sent before the amount of CBS credits reaches a non-negative value.

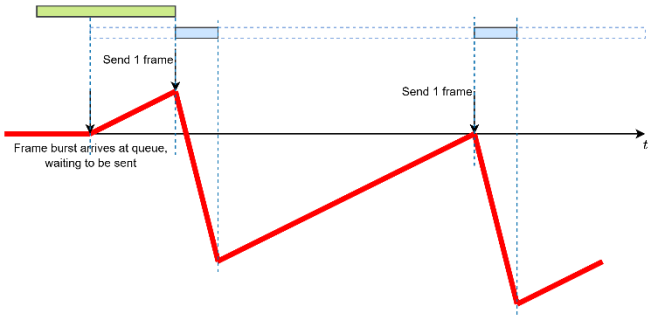


Figure 3: Evolution of CBS credit during burst arrival.

CBS provides real time guarantees to data flows, and there are several efficient methods to configure and analyze CBS flows in TSN networks [6] [7]. CBS is particularly well suited for bursty or asynchronous traffic, without strong jitter constraints.

Note that when TAS and CBS are used together on the same output port, CBS is paused while TAS is active to ensure priority traffic isolation. To maintain the intended average bandwidth, the CBS idleSlope must be scaled proportionally, using the formula

$$idleSlope = operIdleSlope \times \frac{OperCycleTime}{GateOpenTime}$$

where *OperCycleTime* is the TAS GCL period, and *GateOpenTime* is the total amount of time that the CBS gate is open during this period.

### 2.2.3. 802.1Qcr – Asynchronous Traffic Shaping (ATS)

ATS is introduced by IEEE 802.1Qcr (years after TAS and CBS) whose goal is to give every hop along a path the same tight per-flow traffic profile without relying on a global time schedule or on strict credit accounting. Unlike TAS, it needs no 802.1AS time-synchronisation and unlike the classic Credit-Based Shaper, it never lets "positive credit" build up, so burst cascades cannot occur. In addition, unlike CBS, which applies shaping per traffic class, ATS provides native per-flow shaping.

## 2.3. PSFP Fundamentals

Per-Stream Filtering and Policing (PSFP) is a per-stream "firewall" that contains misbehaving talkers by dropping any packets that exceed traffic contracts and, in stricter configurations, even block entire streams that fail filtering conditions. PSFP comprises three sequential stages: Stream Filter, Gate Filter and Flow Meter. These stages are represented in Figure 4.

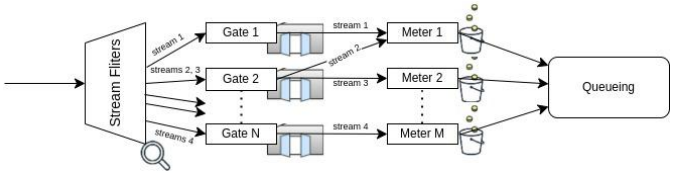


Figure 4: Illustration of filtering steps in PSFP.

### 2.3.1. Stream Filter (IEEE 802.1Qci § 8.6.5.1):

The Stream Filter is mainly a selection and orientation stage, augmented with a few filtering and monitoring mechanisms. An incoming frame passes through a list of filter-matching tests and gets processed by the first filter that fits.

The matching is based on the Stream Handle (defined by stream identification methods that may use source and destination MAC addresses, VLAN, IP and other parameters, in combination with the input port) and/or the priority of the frame. Once selected, the filter tests if the frame size exceeds its *maxSDUSize* parameter. Such frames are dropped. If the packet matches the parameters of the filter and respects the *maxSDUsize*, it proceeds to a Gate Filter defined by this Stream Filter. If the frame is not dropped by the Gate Filter, it then proceeds to the Flow Meter defined by this same Stream Filter.

If the packet does not match any filter entry in the list, it is forwarded. To avoid that behavior and drop all unknown frames, the user can define a filter entry at the end of the Stream Filter list that matches any incoming frame using wildcards for the stream handle and the priority fields, pointing them to a permanently closed Gate Filter.

### 2.3.2. Gate Filter (IEEE 802.1Qci § 8.6.5.2):

Each Gate Filter holds a Stream Control List (SCL): a time-indexed gate state (open/close). If the gate is "closed" at packet arrival, the packet is dropped. If "open," the packet proceeds to the Flow Meter assigned by the Stream Filter.

This gate filter can be used to enforce TAS time-window policies: if clocks of the upstream TAS switch and the local switch are synchronized (which is a common requirement for TAS correct behavior), the SCL can be built to ensure that packets arriving outside their assigned time window are discarded, therefore avoiding errors to propagate across the network.

### 2.3.3. Flow Meter (IEEE 802.1Qci § 8.6.5.3):

Each Flow Meter implements either a single or a double token-bucket meter, depending on the *dropOnYellow* boolean parameter (if true, then a single token-bucket is implemented; otherwise, two are implemented). The token-bucket that is always present has two key parameters:

1. Committed Information Rate (CIR), expressed in SDU bytes per time interval, defining the rate at which the token bucket is filled.
2. Committed Burst Size (CB-Size), also in SDU bytes, defining the maximum number of tokens that can exist in the token-bucket. Note that, in the IEEE standard, the CB-Size and the Credit-Based Shaper are both referred to as "CBS". However, to avoid ambiguity, we deliberately chose to use a different acronym, CB-Size, for the Committed Burst Size throughout this article.

If *dropOnYellow* is false, then two similar parameters are set: the Excess Information Rate (EIR), analogous to the CIR, and the Excess Burst Size, analogous to the CB-Size.

A frame can pass a flow meter by consuming one token per byte. If *dropOnYellow* is false, if the first token-bucket does not have enough tokens for the size of the frame candidate, the frame is forwarded to the second token-bucket; then, if the second token-bucket does not have enough tokens, the frame is dropped. However, if *dropOnYellow* is true, if the first token-bucket does not have enough tokens for the size of the frame candidate, it is immediately dropped.

Due to the token bucket mechanism that is similar to the one used in CBSs, Flow Meters can be used to enforce CBS contracts and avoid errors or misconfigurations. Nonetheless, their token bucket mechanisms present important differences that result in the impossibility of perfectly filtering CBS non-conformities: PSFP's Flow Meter token bucket charges only the SDU length $L$ – that is, $N$ (MSDU) plus 22 bytes (MAC dest + MAC src + + QTag + EthType + FCS, plus any padding). It does not include the 8-byte preamble or 12-byte IPG. Thus, while CBS accounts (SDU + 20) bytes per frame, PSFP only "sees" SDU bytes. Over many small packets, the 20-byte per-packet discrepancy accumulates and may result in the flow exceeding its contractual limit without it being detected by PSFP (Section 3).

## 3. The 20-Byte Omission

The Flow Meter in PSFP considers exactly one token per SDU byte. For a frame whose SDU length is $L$ bytes, the meter checks if $L$ tokens are available and, in that case, it then debits them from the token bucket before forwarding the frame. Otherwise, the frame is marked as yellow and, considering that the parameter *dropOnYellow* is set to True, the frame is dropped. However, transmitting frames with CBS debits $L + 20$ bytes of credit from its token bucket. The 20-byte overhead comprises the 8-byte physical preamble and 12-byte inter-frame gap mandated by IEEE 802.3.

Consider a CBS that is configured to have an incoming rate $\bar{B}_{MAX}$, measured in wire utilization bytes per second (the bar on top indicating that we are talking about wire utilization bytes, including everything from the Ethernet preamble to the IPG). At the ingress of a downstream bridge, if we want a PSFP filter to check that the CBS contract is not breached, a Flow Meter must be used. The question is: how to configure the Flow Meter?

Consider first that there is a single stream of frames in the CBS queue, of which all frames have the same SDU size $L$.

In this Flow Meter, since only SDU bytes are considered, the corresponding CIR will need to be $B_{MAX} = \frac{L}{L+20}\bar{B}_{MAX}$, with $L$ the SDU size of the frame and $\bar{L} = L + 20$ the wire utilization of the frame. For example, if the filter is configured for Ethernet frames of maximum size, i.e., $L = 1522$, the Flow Meter shall be configured using a CIR equal to $B_{MAX} = \frac{1522}{1542}\bar{B}_{MAX}$.

But what happens if the upstream node sends frames of minimal size $L^* = 64$ instead of maximal size, and still respects the SDU byte rate $B_{MAX}$? In this case, the stream conforms to the Flow Meter, but the wire rate incoming to the CBS queue will be:

$$\bar{B}^*_{MAX} = \frac{\bar{L}^*}{L^*}(B_{MAX}) = \frac{84}{64} \cdot \left(\frac{1522}{1542}\bar{B}_{MAX}\right) \approx 1.2955\bar{B}_{MAX} \quad (1)$$

This means that the filter will allow up to almost 30% more traffic than the limit that was expected, which may lead to buffer overflow.

Inversely, if the filter is configured for Ethernet frames of minimum size ($L = 64$), the Flow Meter will be configured using a CIR of $B_{MAX} = \frac{64}{84}\bar{B}_{MAX}$, and for that same SDU transmission rate value, a talker that tries to send larger frames will reach that rate at:

$$\bar{B}^*_{MAX} = \frac{1542}{1522} \cdot \left(\frac{64}{84}\bar{B}_{MAX}\right) \approx 0.7719\bar{B}_{MAX} \quad (2)$$

Therefore, if larger frames are used, the filter might block up to 22.8% of the incoming traffic even if the CBS works correctly. However, the *maxSDUSize* parameter in the Stream Filter can avoid this behavior. A *minSDUSize* parameter would also be able to avoid the up-to-30% excess traffic for smaller frames, but the Stream Filter has no such parameter.

Consider now a stream sending frames of variable size. The size to consider for CBS configuration will depend on the type of contract.

Either the contract conforms to the TSpec contract, and consists in a maximal number of frames, with a maximal size per frame, sent on some period of fixed duration (the "Class Measurement Interval", CMI). In this case, the CBS should be configured to accept the maximal bandwidth, computed with the maximal

frame size (cf. §34.3 in [2]). As previously shown in eq. (1), the bigger the frame size is, the bigger the underestimation of the flow meter, leading to increased risk of undetected contract violations

Alternatively, the contract may be defined using a token-bucket, which specifies a target throughput and an admissible burst in bytes. As shown previously, one must specify which bytes are considered in this contract. If only MSDU bytes are considered, then CBS must take as reference the smallest frame size, since this is the one that generates the highest possible on-wire' throughput. If 'on-wire' bytes are considered, then the CBS configuration becomes independent on the frame size.

However, a CBS queue is, in the general case, not dedicated to a single stream, it serves multiple streams. In such a setup, the total out-of-contract usage at the CBS's queue's output is the sum of per-stream excesses permitted by the individual PSFP meters forwarding streams to this queue.

The excess traffic can be specially problematic if a CBS is used to shape the traffic that has just passed through the Flow Meter: if more bandwidth is let through than the contract had previously allowed, the load on the CBS-shaped output port that follows the Flow Meter might become larger than the configured *idleSlope*, leading to waiting queues that get completely filled and, therefore, to frames being dropped. Hence, this phenomenon results in a negative interference with every frame flow that shares that same waiting queue with the non-conformant frame flow. That is to say, a single stream sending out-of-contract frames (ie. smaller frames than expected) may penalize all other streams in the same CBS queue. To illustrate these effects, some practical experiments are shown in the next section.

## 4. Experiments

The setup is composed of three end-systems (Talker1, Talker2 and Listener) and a switch connected to each one of them, as shown in Figure 5. This minimal setup is designed to clearly illustrate the issue at hand. It is important to notice that the issue can happen in any network set up so that more than one frame flow use the same waiting queue after passing through a Flow Meter.

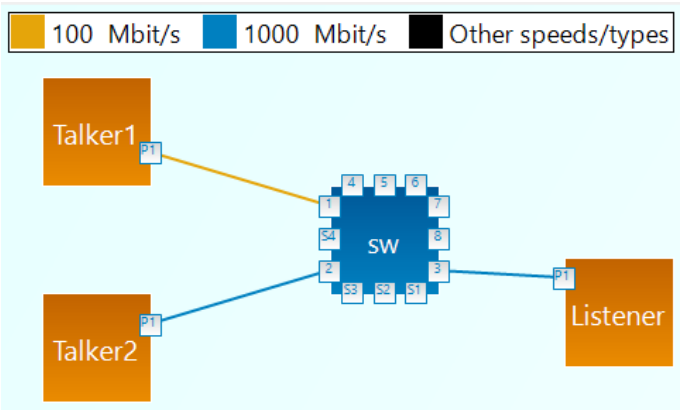


Figure 5: Testbed topology used for evaluation, as visualized in RTaW-Pegase.

Talker1 and Talker2 are implemented in the hardware setup each by a computer with Linux (Ubuntu), a simple script in C to periodically generate network frames, and an "individual switch" between each Talker and SW that implements TAS gates: the goal is simply to guarantee strict periodicity and compensate for the imprecisions in the non-real-time operating system of each Talker. The Listener is a RELY-TSN-REC device, and the switch SW is a Relyum RELY-TSN12, with an internal memory of 32 kB for each output port. This switch has a granularity of 1 $Mb/s$ for CBS *idleSlopes*, 8 $kb/s$ for Flow Meter CIRs and 1 $B$ for Flow Meter CB-Sizes. As mentioned before, in the testbench setup, to overcome imprecisions in the Talker's clocks and their non-real-time operating systems, as well as the impossibility to implement TAS directly on them, a Relyum switch is inserted between each Talker and the main switch SW. This setup enables the use of TAS with precisely scheduled transmission windows.

Two distinct and strictly periodic frame flows are sent by the talkers to the Listener: F1 from Talker1, and F2 from Talker2. Both belong to the same traffic class. F2 is 500 bytes long (SDU, so $L_2 = 500$ and $\bar{L}_2 = 520$), transmitted at every millisecond ($P_2 = 1ms$). For F1, three test cases are constructed: Nominal, Control and Faulty.

The Nominal case is set to represent the expected behavior of the network given the nominal behavior of its talkers. The Control scenario represents an error or misconfiguration of Talker1 and the behavior of the network when the Flow Meter can successfully catch the discrepancies to the Nominal case. The Faulty scenario represents an error or misconfiguration of Talker1, making it behave as a "babbling idiot", but in a way that cannot be caught by the Flow Meter, even though it increases the on-wire bandwidth utilization beyond what was specified by the frame's contract. In the following, F1 has the following parameters for each case:

1. **Nominal:** F1 is 1500 bytes long ($L_1 = 1500\ B$ and $\bar{L}_1 = 1520\ B$) and its period $P_1 = 1ms$. Therefore, $B_1 = 12\ Mb/s$ and $\bar{B}_1 = 12.16\ Mb/s$.
2. **Control:** F1 has $L'_1 = 1500\ B$ and $P'_1 = 0.5ms$. Therefore, $B'_1 = 24\ Mb/s$ and $\bar{B}'_1 = 24.32\ Mb/s$.
3. **Faulty:** F1 has $L''_1 = 64$ and $P''_1 = 0.043ms$. Therefore, $B''_1 = 11.91\ Mb/s$ and $\bar{B}''_1 = 15.63\ Mb/s$.

PSFP filters are defined in the switch SW so that F1 always passes through a specific Flow Meter FM1, and F2 through another Flow Meter, FM2. For simplicity, their *dropOnYellow* parameter is set to *true*. Also, a small margin is set to numerical values to avoid any effects related to the precision of clock synchronization. Their unique parameters are set as follows (for every test case):

1. **FM1:** designed for F1 with a 1% margin for the CIR and a single byte margin for the CB-Size.
    a. **CIR:** 12120 $kb/s$
    b. **CB-Size:** 1501 $B$

2. **FM2:** designed for F2 with a 1% margin for the CIR and a single byte margin for the CB-Size.
    a. **CIR:** 4040 $kb/s$
    b. **CB-Size:** 501 $B$

Also, a CBS is set in the output port of the switch that connects it to the Listener node. The CBS is set with respect to the nominal case and, therefore, its *idleSlope* shall be no less than $12.16 + 4.16 = 16.32\ Mb/s$. Due to the $1\ Mb/s$ granularity imposed by the hardware, it is set up to $17\ Mp/s$.

### 4.1. Nominal Case Results

A simulation of 10 seconds was conducted for the Nominal case using RTaW-Pegase and, as we can see in Figure 6, F1 can seamlessly pass through its Flow Meter token bucket, as expected: no frames are dropped. In that figure, time is represented as the horizontal axis, green rectangles are frames of F1 sent in the Talker1-SW link, and the saw-shaped curve represents the amount of credits the Flow Meter for F1 deployed in SW has.

F2 can also pass seamlessly through SW, and we can see from Figure 7 (from RTaW-Pegase's TraceInspector module) that the distance between subsequent F2 frames is distributed around its nominal period of 1 ms. The simulation of clock imprecisions can explain the non-negligible width of that distribution.

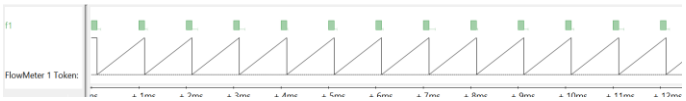


Figure 6: Simulation results for the link Talker1-SW in the Nominal case.

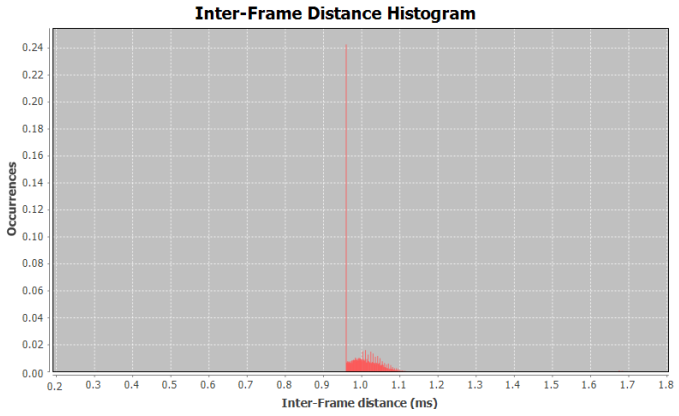


Figure 7: Simulation for inter-frame distance distribution for F2 in the SW-Listener link for the Nominal case.

Then, the hardware setup was tested and the frames received in the Listener were captured to be later analyzed. Figure 8 shows the distribution of distances between consecutive F2 frames, as extracted using RTaW-Pegase's TraceInspector module. No significant jitter is observed in the reception times of F2 frames at the Listener node.

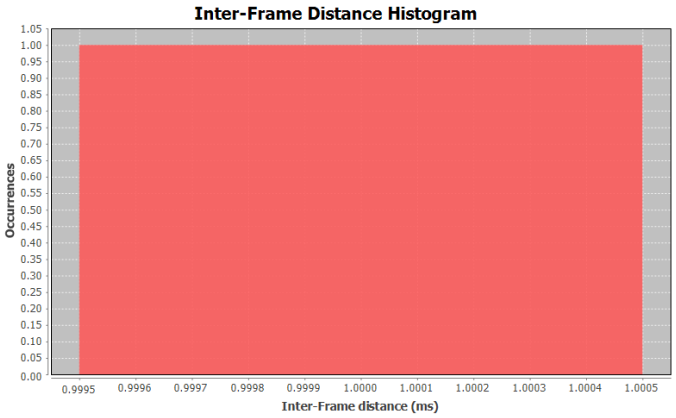


Figure 8: Inter-frame distance distribution for F2, captured in the hardware setup at the Listener.

### 4.2. Control Case Results

A simulation of 10 seconds was conducted for the Control case using RTaW-Pegase and, as can be seen in Figure 9, one out of every two F1 frames is dropped by the Flow Meter. This is expected, since F1 breaches the specified contract for that frame ($\bar{B}_1' > \bar{B}_1$ and $B_1' > B_1$). F2 can pass seamlessly by the filters, and we can see from Figure 10 that the distribution of the distance between subsequent F2 frames is the same as in the Nominal case.

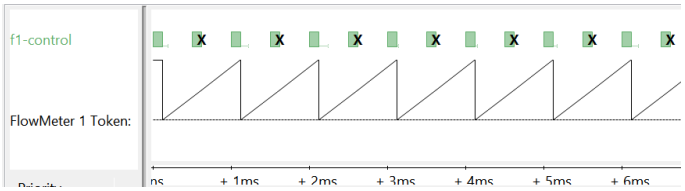


Figure 9: Simulation results for the link Talker1-SW in the Control case (frame drops represented as black Xs).

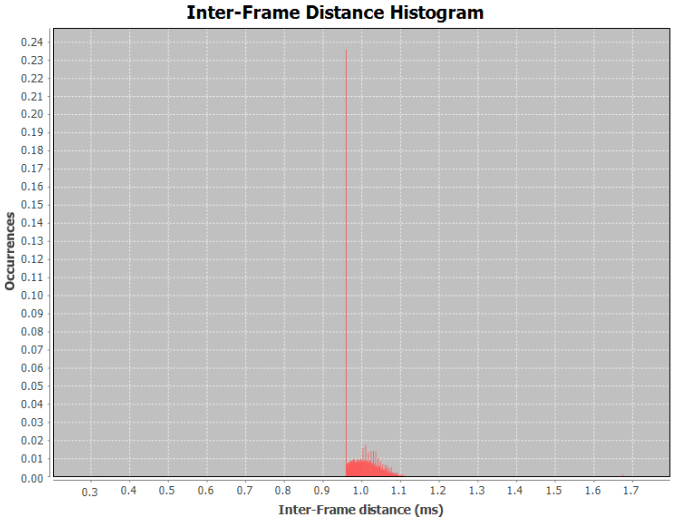


Figure 10: Simulation for inter-frame distance distribution for F2 in the SW-Listener link for the Control case.

As shown in Figure 11, the hardware frame capture does not reveal any noticeable effects associated with the Control case.

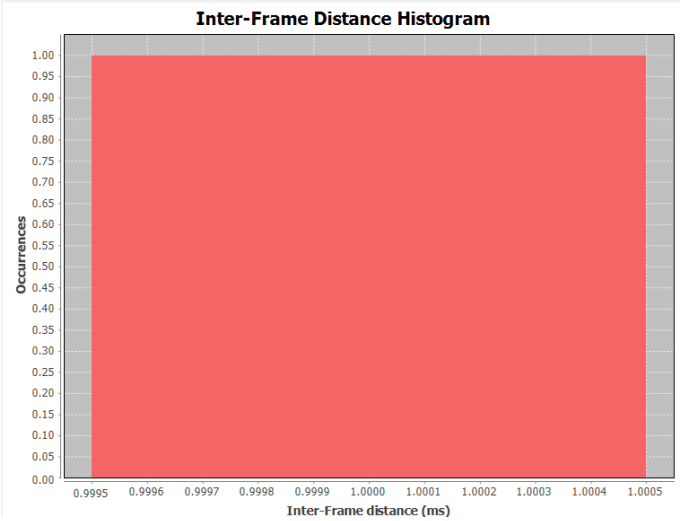


Figure 11: Inter-frame distance distribution for F2, captured in the hardware setup at the Listener for the Control case.

### 4.3. Faulty Case Results

A simulation of 10 seconds was conducted for the Faulty case using RTaW-Pegase. Figure 12 shows that, although F1 does not conform to the on-wire bandwidth contract ($\bar{B}_1'' > \bar{B}_1$), the Flow Meter is incapable of filtering the excess traffic, since $B_1'' < B_1$ due to the 20-byte overhead.

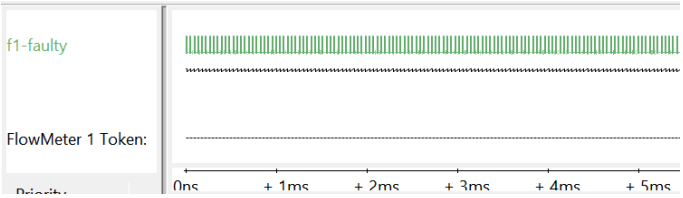


Figure 12: Simulation results for the link Talker1-SW in the Faulty case.

Since the filter let through excess traffic, it can be seen from Figure 13 that a significant jitter appears for F2. More than that, the concentration of the distribution around the values of twice, three times and four times the original period indicates important frame losses: one for each 2 frames, 2 for each 3 frames and even 3 for each 4 frames.

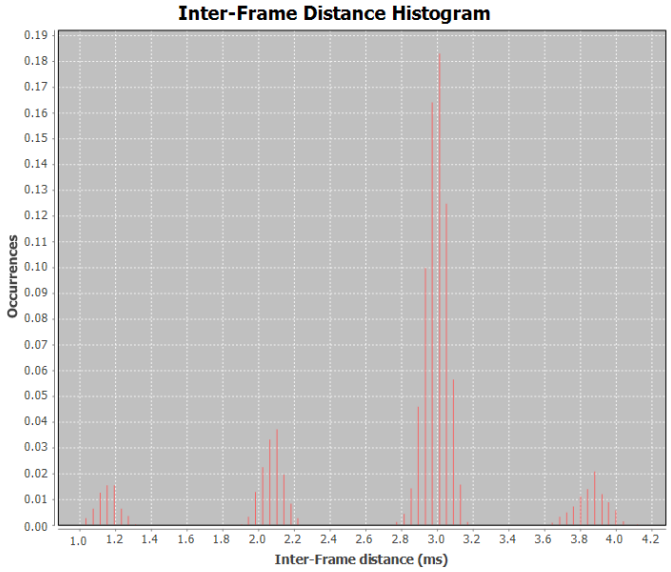


Figure 13: Simulation for inter-frame distance distribution for F2 in the SW-Listener link for the Faulty case.

The scenario observed in the simulation can be verified in the data captured by the Listener in the hardware setup, as can be seen in Figure 14.

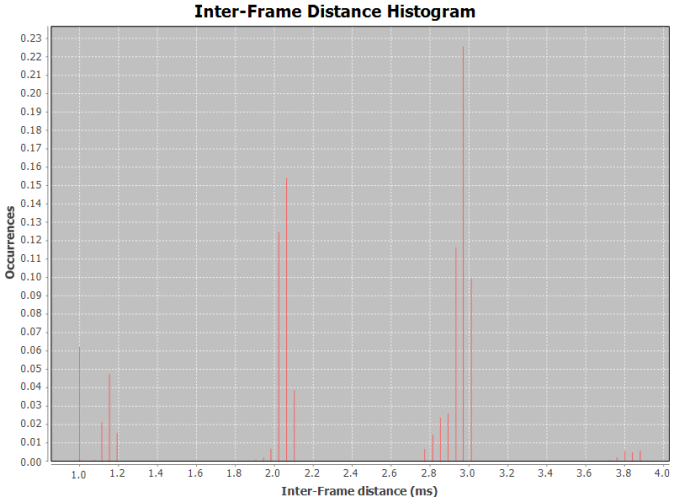


Figure 14: Inter-frame distance distribution for F2, captured in the hardware setup at the Listener for the Faulty case.

### 4.4. Results Discussion

Simulations and network trace captures have shown that Flow Meters might protect from certain CBS non-conformities such as the one modeled by the Control case. However, results from the Faulty case demonstrate that Flow Meters alone cannot fully safeguard the network against all classes of non-conforming behaviors.

The inability to block excess bandwidth usage will have an impact on every flow that shares the same CBS as the faulty traffic, causing frame losses in the event of a “babbling idiot” fault. Such scenario can occur in any network using CBSs despite the use of Flow Meters to enforce their contract, unless specific mitigation strategies are applied.

## 5. Mitigation Strategies

### 5.1. Mitigation with over-provisionning

Since the Flow Meter is inherently unable to fully enforce bandwidth contracts, to avoid frame drops due to queue saturation, mitigation necessarily relies on overprovisioning. Therefore, if a CBS queue aggregates several flows and there is a need to protect one flow from misbehaviors of others, relying solely on per-flow metering via PSFP is insufficient. In such cases, the CBS *idleSlope* must be increased to cope with such a situation.

Figure 15 shows the required correction to be added to the *idleSlope* of a CBS for each flow that passes through it, based on the flow maximum message size (in SDU bytes). The correction is derived from the following analytical expression, where $k$ is the increase in the *idleSlope* and $L$ is the SDU message size in bytes:

$$k = \left(\frac{84}{64} \cdot \frac{L}{L+20} - 1\right) \cdot 100\%$$

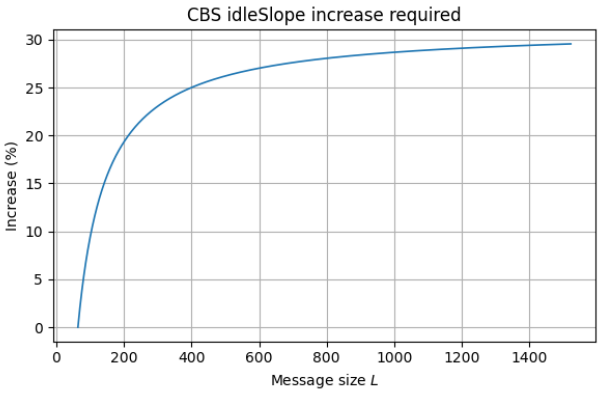


Figure 15: Required idleSlope increase to prevent queue saturation as a function of frame size (SDU bytes).

For example, if a CBS is configured at the output port of a switch that aggregates three distinct flows $X$, $Y$ and $Z$, then in the nominal case, the CBS's *idleSlope* is set as:

$$idleSlope_{Nominal} = \bar{B}_X + \bar{B}_Y + \bar{B}_Z$$

But to protect frame $X$ from a "babbling idiot" fault of $Y$ and $Z$ simultaneously, the corrected *idleSlope* must satisfy the following inequality:

$$idleSlope_{Corrected} \geq \bar{B}_X + (1 + k_Y)\bar{B}_Y + (1 + k_Z)\bar{B}_Z$$

An analogous logic would apply for flows $Y$ and $Z$. A safe *idleSlope* value can then be chosen as the minimum value that satisfies all three constraints simultaneously.

As illustrated in Figure 15, avoiding queue saturation in the presence of a "babbling idiot" fault requires allocating significantly more bandwidth to CBS-shaped traffic—even when the frame size $L$ is relatively small. For instance, at $L =$ 200 bytes, approximately 20% more bandwidth is needed to maintain safe operation. However, this overhead may be acceptable in some scenarios, particularly as the deployment of gigabit-class Ethernet links becomes more common.

### 5.2. Standard-Level or Product Enhancements

While the mitigation strategy is limited, correction can be achieved with minimal changes and could be implemented either through updates to the standard or as extensions provided by TSN switch manufacturers.

One first possible solution, inspired by AFDX, is to enforce a minimum frame size per stream. Since many streams transmit fixed-size frames, allowing PSFP to check for a stream-specific minimum size would prevent overuse of transmission time on the wire. Given that PSFP already supports maximum frame size checks, this would be a relatively minor extension of its capabilities. However, such an approach is not universally applicable. For instance, in video streams, the application-level data is often fragmented into multiple Ethernet frames, and there is no guarantee regarding the size of the final fragment.

A second solution is to have PSFP account for the 20-byte media overhead. Since CBS already includes this overhead in its shaping calculations, the capability to consider it already exists within the device. However, CBS operates at the output port, whereas PSFP is applied at the ingress of the switch. Despite this separation, it is worth noting that ATS—typically implemented immediately after PSFP (see Fig. 8.13 in [2])—does incorporate the media-specific overhead when computing eligibility times, as specified in §8.6.5.6 of [2]. This suggests that integrating similar awareness into PSFP is feasible and aligned with existing TSN mechanisms.

In both cases, such evolutions can be made in the standard itself or proposed as extensions by the hardware providers.

### 5.3. Mitigation in AFDX

AFDX also has policing elements based on token bucket, like the PSFP meters. In AFDX, each flow must have its own token bucket (whereas several flows can share the same PSFP in TSN). But this token bucket takes into account the media overhead (using the term "line size"), *cf.* § 4.1.1.1 and Figure 4-2 in [8].[1] Moreover, a minimal and maximal frame size must be specified for each flow, and the amount of media overhead is computed based on the minimal frame size. Frames with smaller sizes are dropped. Lastly, AFDX may get rid of considering such overhead by using a per-frame policing instead of a per-byte policing, *cf.* § 4.1.1.3.

### 5.4. Mitigation on TSN/ATS

We have shown in previous sections that PSFP is unable to prevent a faulty or malicious switch from increasing the delays

[1] "The frame size value shown in Figure 4-2 corresponds to the actual time a frame occupies the Ethernet line (s). Therefore, all fields must be considered: IPG (12 octets) + Preamble (8 octets) + MAC Frame size (L = 64 to 1518 octets)."

at a downstream switch, leading to increased latencies or even buffer overflows.

For ATS, the situation is slightly different. ATS shaping is based on the computation of an *eligibility time* for each frame, and each frame will be allowed to be transmitted only once that timestamp is reached. Consider first the common case of per stream ATS shaping, where each stream is individually policed by PSFP

As with CBS, the PSFP meter cannot distinguish small frames from big frames. Since the eligibility time calculation includes the "media specific overhead" (cf. § 8.6.5.6 in [2]), sending several small frames instead of one big one will create more overhead and increases latency. But ATS shapers are organized into groups, with one group per input port, that enforces a FIFO behavior inside a group. Consequently, an increase in delay affects all flows within the same group but not flows arriving from other input ports. However, all ATS traffic within a given class is enqueued into the same output queue. This means a faulty or malicious switch, even if unable to delay traffic from other sources, may still attempt to overflow the shared class buffer. Fortunately ATS offers a counter-measure: the standard defines a *MaximumResidenceTime*. Frames that exceed this limit can be dropped. This mechanism allows the system to discard excessively delayed frames—such as those affected by small-frame overhead—thus helping to prevent buffer overflow.

## 6. Conclusion

Time-sensitive applications are using PSFP increasingly to improve safety, but our evaluation shows that it cannot guarantee complete protection from faults in all situations. We found that a small 20-byte accounting mismatch in how frame sizes are counted can let a data flow send up to 30% more traffic than allowed. We also saw that the way CBS recovers its credit after sending a burst can force network engineers to set much higher burst allowances than should be needed, which makes traffic control less accurate. Together, these issues can lead to delays and buffer overflows, even in networks that are set up correctly.

Still, PSFP remains a valuable tool for limiting the effects of faults, if applied carefully. We have shown that problems can be avoided by overprovisioning for CBS *idleSlopes*. Beyond that, we practical improvements to the standard: (1) allowing Flow Meters to account for the actual on-wire size, and (2) checking frame sizes against a minimum allowed value, similar to the existing *maxSDUSize* constraint. Both changes are incremental, compatible with current systems, and preserve the simplicity of PSFP design.

Looking forward, we believe that emerging traffic shaping mechanisms such as ATS hold promise for reducing or eliminating the need for oversized buffers. However, combining ATS with PSFP and ensuring robust performance across diverse traffic patterns remains a complex challenge and an area for future investigation.

We hope that network engineers, tool developers, and standardization bodies will find these observations useful to refine existing mechanisms, and validate behavior through real-world deployments to advance TSN reliability and fault-tolerance.

## References

[1] M. Boyer, "Usage of TSN Per-Stream Filtering and Policing," *hal-04159172v2,* 2023.

[2] IEEE Standards Association, 802.1Q - IEEE Standard for Local and Metropolitan Area Networks--Bridges and Bridged Networks, 2022.

[3] C. Xue, T. Zhang, Y. Zhou, M. Nixon, A. Loveless and S. Han, "Real-Time Scheduling for 802.1Qbv Time-Sensitive Networking (TSN): A Systematic Review and Experimental Study," in *IEEE 30th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Hong Kong, 2024.

[4] 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,* 2023.

[5] H. Hassani, P. J. L. Cuijpers and R. J. Bril, "Work-in-Progress: Layering Concerns for the Analysis of Credit-Based Shaping in IEEE 802.1 TSN," in *16th IEEE International Conference on Factory Communication Systems (WFCS 2020)*, Porto, 2020.

[6] L. Zhao, P. Pop, Z. Zheng, H. Daigmorte and M. Boyer, "Latency Analysis of Multiple Classes of AVB Traffic in TSN With Standard Credit Behavior Using Network Calculus," *IEEE Transactions on Industrial Electronics,* 2021.

[7] J. Migge, J. Villanueva, N. Navet and M. Boyer, ""Insights on the Performance and Configuration of AVB and TSN in Automotive Ethernet Networks"," in *9th European Congress on Embedded Real Time Software and Systems (ERTS 2018)*, Toulouse, 2018.