On the Performance of Stream-based, Class-based Time-aware Shaping and Frame Preemption in TSN

David Hellmanns*, Jonathan Falk*, Alexander Glavackij*, René Hummen‡, Stephan Kehrer‡, Frank Dürr*

* University of Stuttgart
Institute of Parallel and Distributed Systems
70569 Stuttgart, Germany
firstname.lastname@ipvs.uni-stuttgart.de

‡ Hirschmann Automation and Control GmbH
CTO Office
72654 Neckartenzlingen, Germany
firstname.lastname@belden.com

***Abstract*—Time-sensitive Networking (TSN) is an evolving group of IEEE standards for deterministic real-time communication making standard Ethernet technology applicable to safety-critical application domains such as manufacturing or automotive systems. TSN includes several mechanisms influencing the timely forwarding of traffic, in particular, a time-triggered scheduling mechanism called time-aware shaper (TAS) and frame preemption to reduce the blocking time of high-priority traffic by low-priority traffic. Although these mechanisms have been standardized and products implementing them begin to enter the market, it is still hard for practitioners to select and apply suitable mechanisms fitting the problem at hand. For instance, TAS schedules can be calculated for individual streams or classes of traffic, and frame preemption with strict priority scheduling (w/o TAS) might seem to be an option in networks with extremely high data rates. In this paper, we make a first step towards assisting practitioners in making an informed decision when choosing between stream-based TAS, class-based TAS, and frame preemption by comparing these mechanisms in selected scenarios using our TSN network simulation tool NeSTiNg. Moreover, to facilitate the application of class-based TAS, we derive a formula for calculating class-based TAS configuration.**

***Index Terms*—real-time communication, time-sensitive networking, TSN, scheduling, frame preemption**

I. Introduction

"Hard" realtime communication with deterministic bounds on network delay and delay variance (jitter) is of paramount importance for many safety-critical applications from the realm of cyber-physical systems (CPS). Two prominent domains where deterministic realtime communication can already be found today are manufacturing systems (Industry 4.0) and automotive systems, where the violation of time bounds could lead to damaged machines, work pieces, vehicles, or even physical harm to people.

Traditionally, realtime communication solutions, so-called fieldbuses, have been proprietary Ethernet extensions. These proprietary extensions led to communication silos in factories, hampering the interoperation between different vendors since gateways must translate between different protocols. The IEEE Time-Sensitive Networking (TSN) Task Group (TG) was initiated to define additional mechanisms providing realtime-capability to standard Ethernet according to IEEE 802.1 and IEEE 802.3. Since IEEE Ethernet is the de-facto Layer 2 standard in the IT sector, it now has the potential to become ubiquitous in industrial realtime communication.

Nevertheless, the standardization of the TSN mechanisms is only the first step. One of the key components in the TSN standards is the Time-Aware Shaper (TAS). TAS enables the implementation of a time division multiple access (TDMA) scheme on each egress port. Another important concept of TSN is frame preemption enabling the preemption of best-effort frames to reduce the blocking time of high-priority traffic. Equipped with all these new concepts, practitioners are currently discussing how these can be used to fulfill the requirements in the industrial sector.

Our paper aims at supporting practitioners and ongoing TSN-related standardization efforts by exploring characteristic behaviors of different TAS configurations and frame preemption. Furthermore, we intend to create awareness for class-based scheduling. Class-based scheduling is another use-case of the TAS whose configuration requires less computational effort compared to stream-based scheduling which is currently prevailing in academia. Therefore, we provide a qualitative comparison of stream-based TAS scheduling, class-based TAS scheduling and frame preemption. For an informed discussion, we also derive a formula for class-based TAS configuration. In addition, we present the results of a simulation-based evaluation of these mechanisms in a concrete scenario using our open-source TSN network simulator NeSTiNg [1].

The remainder of this paper is structured as follows. Sec. II gives an overview over the related work. In Sec. III, we present the system model. Next, we introduce the different packet scheduling approaches in Sec. IV, and analyze the approaches qualitatively. In Sec. V, we evaluate the different approaches in a simulation. Finally, we conclude the paper in Sec. VI.

II. Related Work

TSN is a topic of high interest in academia and industry. A recent overview of TSN can be found in [2]. With regard to TAS, the related work consists of two clusters, namely worst-case analysis and schedule synthesis.

Worst-case analysis is used to formally derive deterministic bounds for an existing system. For example, in [3], [4], individual (asynchronous) shaping mechanisms are analyzed. Other work focuses on the various integration effects of combining TAS with credit-based shaping [5].

Another approach is to provide deterministic bounds "by design", i.e., by a suitable synthesis of the network config-

Accepted for publication in Proceedings of *2020 IEEE International Conference on Industrial Technology (ICIT)*, Buenos Aires, Argentinia, February 26–28, 2020

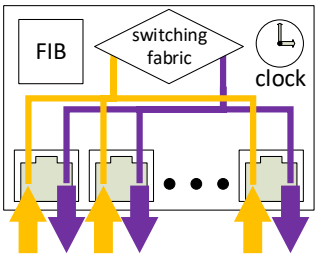


Fig. 1: Simplified switch schematic.

uration. Schedule synthesis is required, e.g., to implement a TDMA scheme with TAS. In the context of TAS, this schedule is referred to as Gate Control List (GCL). The calculation of GCLs for stream-based scheduling is challenging because it is in general an NP-hard problem, and is often solved by constraint-based programming or heuristics [6]–[8]. In [9], [10], the schedule synthesis problem is extended by computing routes and GCLs jointly.

However, these papers only consider stream-based scheduling and require that applications and network are fully synchronized, which is not necessary for class-based scheduling. Here, we broaden the scope by discussing the advantages and drawbacks of different ways to use TAS and frame preemption, to help practitioners to understand, which mechanism is well-suited for which requirements.

III. System Model

In this section, we introduce our system model, which is a condensed version of the IEEE TSN standards [11]. We describe the architecture of a TSN switch, our definition of streams, and our model of end stations. Finally, we discuss which delays a frame experiences on its path.

A. Switch Architecture

We start by introducing the architecture of a TSN switch conforming to [11]. To this end, we give a brief overview over the processing pipeline in the switch and present TAS and frame preemption in detail in the following. For better understanding, we refer to individual mechanisms by the names of the amendments to the IEEE standards (even though by now, all discussed amendments have been merged into IEEE Std 802.1Q-2018 [11] or IEEE Std 802.3-2018 [12]).

We consider multi-port switches with full-duplex links and egress queuing (cf. Fig. 1). Incoming frames are subject to filtering, metering, etc., before they are forwarded to the appropriate egress port. At the egress port, frames are sorted into up to 8 (FIFO) egress queues to differentiate different traffic classes (cf. Fig. 2). The switch assigns frames to queues based on their priority code point (PCP) values. Transmission Selection (TS) retrieves eligible frames from the head of the queue with highest priority for transmission over the link. A frame is eligible for transmission if the shaper of the queue has released it (cf. Fig. 2). The TSN TG standardized different shapers, e.g., strict-priority and credit-based shaping (formerly IEEE Std 802.1Qav). As Fig. 2 indicates, the Time-Aware Shaper (TAS) takes effect *after* the other shapers. After passing

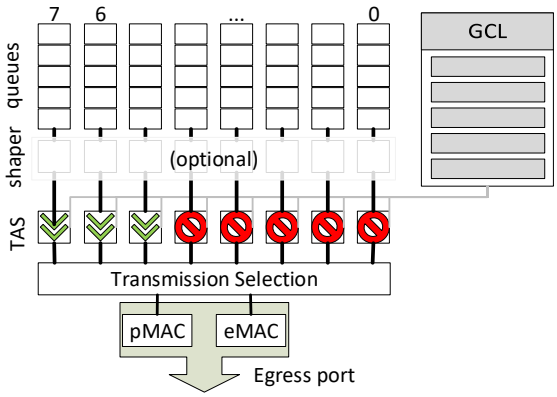


Fig. 2: Simplified TSN switch output port schematic.

TS, Media Access Control (MAC) is responsible to transmit the frame. Frame preemption is also implemented at this stage. Next, we present the TAS and, thereafter, we describe frame preemption in more detail.

1) Time-Aware Shaper (TAS): Intuitively, TAS functions as a gate for its egress queue. This gate can enable or disable its egress queue by opening or closing it, respectively. Frames in disabled queues are not eligible for transmission. A gate driver changes the gate state according to a cyclic schedule called Gate Control List (GCL). In the GCL, the opening time and opening duration of gates are specified. Here, the time it takes to iterate once through the GCL is identical to the *network cycle* time. In general, we assume that the clocks of all gate drivers in the network are synchronized, e.g., using IEEE Std 802.1AS [13]. TAS allows to isolate traffic classes or single streams in time.

2) Frame Preemption: Frame preemption (IEEE Std 802.1Qbu and IEEE Std 802.3br) enables so-called express frames to preempt the transmission of preemptible frames. Without frame preemption, an MTU-sized frame ($1500\,\mathrm{B}$) could block a port for about $12.5\,\mu\mathrm{s}@1\,\mathrm{Gbit/s}$, whereas frame preemption limits the maximum blocking time to ca. $1\,\mu\mathrm{s}$. If preemption occurs, the resulting parts of the preempted frame are called fragments. The fragment size must be at least $64\,\mathrm{B}$. To determine if a frame should be handled as express frame, the egress queues of the port are marked as express queue or preemptible queue. Express frames and preemptible frames are handled in a dedicated express MAC (eMAC) and a preemptible MAC (pMAC), respectively (cf. Fig. 2).

B. Stream

We call a directional flow of data *stream*. The sending end stations are called `talkers`, and the receiving end stations are called `listeners`. The roles of end stations are stream-specific, meaning a host can be a `talker` for one stream and a `listener` for another stream. Moreover, we distinguish between isochronous streams, cyclic streams, and best effort streams. For isochronous streams, the talker cycle and the network cycle are aligned, and the transmission time of the stream is precisely specified as an offset to the network cycle

start. Isochronous streams typically require minimal end-to-end delays and minimal jitter bounds, e.g., for motion control applications. In contrast, cyclic streams do not require a synchronization of talker cycle and network cycle. Consequently, the start of transmission of a cyclic stream cannot be specified with regard to the *network cycle*. For cyclic traffic, we assume that the end-to-end delay may not exceed one network cycle time and that the jitter may be in the magnitude of one cycle time. We also assume that isochronous and cyclic streams transmit one frame per cycle and have a maximum size of 300 B, which is a common value for control traffic. The network needs to guarantee delay bounds for isochronous and cyclic streams irregardless of the best-effort traffic load. For additional information regarding the traffic classes, we refer to [14].

C. Frame Delays

To enable an informed discussion about end-to-end delays and jitter bounds, we introduce the considered delay components which make up the total delay and an according nomenclature based on IEEE Std 802.1Qcc [15]. As an introductory example, consider a stream with path length 3: `Talker - Switch - Listener`. To begin with, the talker transmits a frame of the stream. The time span from the start of the transmission of the frame by the talker to the moment the frame is fully received at the `listener` is denoted as *end-to-end delay (e2e delay)*. We denote the time between start of transmission by the `talker` and receiving of the first bit at the `switch` as *propagation delay*. The propagation delay is a physical quantity and depends on the material of the cable and its length ($\approx 5\,\mathrm{ns/m}$). We denote the time span from receiving the first bit of the frame to having received the complete frame as *frame-dependent delay* also known as store-and-forward delay because the frame needs to be fully received before it can be forwarded. The frame-dependent delay depends on the frame length and the data rate: $\frac{l_{\text{frame}}\,\text{bit}}{r_{\text{link}}\,\text{bit/s}}$. After receiving the frame, the `switch` processes the frame and assigns it to an egress port. We denote the time of processing and assignment to the egress port as *frame-independent delay* because it is independent of the frame size. At the egress port, the frame waits until it is selected by the Transmission Selection. Since the frame waits in an egress queue, we denote this delay as *queuing delay*. The queuing delay depends on how many frames are ahead in the queue and depends on the schedule of the TAS gate of the egress queue. Therefore, we further split up the queuing delay into the *gate delay* and the *interference delay*. The *gate delay* describes the time span a frame is queued due to closed gates and, thus, it is deterministically bounded. The interference delay denotes the time a frame is queued because of transmissions of other frames. As a result, the interference delay depends on the current load of the egress port and, thus, it is subject to variations which need to be upper-bounded to give e2e delay guarantees. We aggregate propagation delay, frame-dependent and frame-independent delay as *path delay*. The path delay is constant for every stream because it only depends on the path length and frame size.

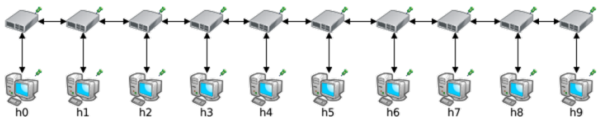


Fig. 3: Example line topology with 10 switches and 10 hosts as seen in NeSTiNg.

Finally, we assume that the link speeds and frame-independent delays are constant.

D. Network Topology

In this paper, we focus on line topologies, which are prevalent in industrial use cases. If redundancy is required, a line can be extended to a ring by connecting the two ends to prevent network partitioning. Despite the simplicity of the line topology, we can still observe the different properties of the considered scheduling approaches. Here, we consider the special case where to each `switch` a single host (with dual role of `talker`/`listener`) is attached, and each `switch` (excluding the terminal switches) is connected to its neighboring switches (cf. Fig. 3). We therefore can define the size of a network with line topology by n_l, which denotes the number of switches in the topology.

IV. Packet Scheduling in TSN

In the previous section, we introduced the Time-Aware Shaper (TAS) and frame preemption. However, the design space for how to use these mechanisms to fulfill the stream requirements with regards to e2e delay and jitter is vast. Therefore, we start by discussing the simplest mechanism from the planning perspective, which is frame preemption, and increase the complexity by looking at class-based and finally stream-based scheduling.

A. Frame Preemption

To use frame preemption, only little planning effort (configuring the express queues) and no time synchronization are required. Frame preemption reduces the queuing time for express frames by preempting best-effort transmissions. Therefore, frame preemption can provide worst-case bounds for e2e delay and jitter of a single frame (taking the worst-case interference time on each hop into account). However, with increasing number of express frames in a network, frame preemption degrades to strict-priority scheduling since express frames cannot preempt each other. In the worst case, all express streams in the network use the same egress port and, therefore, express frames are transmitted based on strict-priority order. Note that the interference delay decreases with increasing data rate since the transmission time of a frame reduces. Consequently, the e2e delay and jitter bounds of frame preemption mainly depend on the knowledge about the behavior of talkers. Assuming that all express streams are cyclic and a-priori known, upper bounds for the queuing delay at each hop can be calculated by considering the worst case.

With regard to bandwidth usage, frame preemption is a work-conserving mechanism, i.e., if there are queued frames for an output port the link never idles. Furthermore, frame

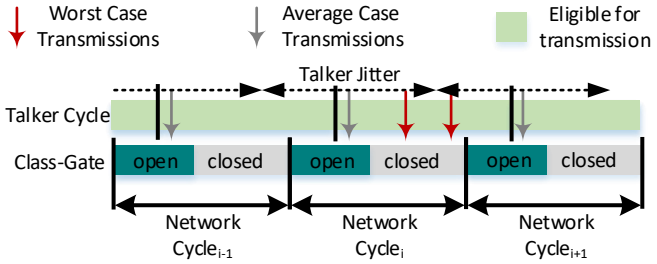


Fig. 4: Class-based scheduling: The talkers are sending unsynchronizedly but *only once per cycle time*. Worst-Case assumption: All talkers transmit at the worst case transmission time in Cycle *i* and in Cycle *i + 1*. To this end, the *class-window* in the i-th cycle is unused and in the i+1-th the network needs to transport twice of the usual amount of traffic. As a result, the *class-window* needs to be designed to be capable of delivering twice the amount of traffic.

preemption does not have specific buffer requirements. All in all, frame preemption is capable of distinguishing two types of traffic, namely express and preemptible traffic without any synchronization or planning effort but the guarantees strongly depend on the knowledge about all potentially conflicting express streams in the network.

B. Class-based Scheduling

Class-based scheduling is a TAS-based approach to give e2e delay and jitter guarantees to cyclic streams. We assume the following talker behavior for cyclic streams (cf. Fig. 4): 1) The talker cycle has the same length as the network cycle. 2) The talker cycle and network cycle are not aligned. 3) The sending time of the talker can deviate $\pm 50\%$ of the talker cycle, i.e., a complete cycle in total.

In class-based scheduling, streams sharing jitter and deadline requirements are scheduled as a group. To provide isolation among different groups, all streams of one group are assigned to a dedicated traffic class. This class needs to be mapped to a dedicated egress queue, because the TAS can only distinguish between egress queues. We propose to open the gate of this traffic class once per cycle. We call the gate opening duration for the considered class *class-window*. The class-window can be placed arbitrarily in the schedule but it needs to be aligned among all switches in the topology. Since the gate is only open once, the class-window must be sufficiently large to guarantee that all streams are delivered within one cycle time considering the defined talker behavior. In the following, we propose a proof-of-concept formula to calculate the size of the class-window.

We begin with the constraint that the e2e delay may not be larger than one network cycle time for all cyclic streams s:

$$\forall s \in \mathcal{S} : d^s_{\text{e2e}} \leq t_{\text{cycle}}, \tag{1}$$

$$\text{or, equivalently} \quad \max_{s \in \mathcal{S}} (d^s_{\text{e2e}}) \leq t_{\text{cycle}} \tag{2}$$

Based on this constraint, we present a simple derivation of the class-window size t_w subject to Eq. 2. Since the derivation presented below serves as a proof-of-concept rather than a full-fledged scheduling algorithm, it over-dimensions t_w for most scenarios.

TABLE I: Notation

Symbol	Meaning
$\mathcal{V}_H \subset \mathcal{V}$	set of vertices denoting *hosts*
$\mathcal{V}_S \subset \mathcal{V}$	set of vertices denoting *switches*
$\mathcal{S}$	set of streams
$P_S(s) \subseteq \mathcal{V}_S, s \in \mathcal{S}$	sequence of *switches* on path of stream s
$P_E(s) \subseteq \mathcal{E}, s \in \mathcal{S}$	sequence of *edges* on path of stream s
$S_e(e) \subseteq \mathcal{S}, e \in \mathcal{E}$	set of streams using egress port e
t_w	size of the class-window
t_{cycle}	cycle time
$d_{\text{f-ind}}$	frame-independent delay
$d_{\text{f-dep}}$	frame-dependent delay

Even though we focus on line topologies in this paper, the derivation holds for arbitrary networks, which we model by a directed graph $G(\mathcal{V}, \mathcal{E})$. Vertices $\mathcal{V}$ represent switches and hosts. A full-duplex link attached to a switch is represented by an outbound edge and an inbound edge. Switches are equipped with queues and gates at the egress ports (cf. Fig. 2). Further notation is summarized in Tab. I.

Recalling Sec. III, the e2e delay d^s_{e2e} in Eq. 2 consists of a path delay d^s_{path} and time-varying queuing delay d^s_{queue}:

$$d^s_{\text{e2e}} = d^s_{\text{path}} + d^s_{\text{queue}}. \tag{3}$$

The time-varying queuing delay d^s_{queue} is itself composed of two components: $d^s_{\text{queue}} = d^s_{\text{gate}} + d^s_{\text{interference}}$. Term d^s_{gate} denotes the time a frame is "stuck" behind a closed gate, whereas $d^s_{\text{interference}}$ models that part of the queuing delay which is caused by waiting for the completion of the transmission of frames in front of the frame of stream s when the gate is open.

We can bound d^s_{gate} because we open the gate once per cyle for t_w. Thus the frame must wait $d^s_{\text{gate}} \leq t_{\text{cycle}} - t_{\text{w}}$ if a frame arrives right after the class window. Taking this into account and splitting d^s_{e2e} into its components, we can rewrite Eq. 2 as

$$\max_{s \in \mathcal{S}} \left(d^s_{\text{path}} + d^s_{\text{interference}} + t_{\text{cycle}} - t_w \right) \leq t_{\text{cycle}}. \tag{4}$$

Since t_{cycle} and t_w are independent of a particular stream s,

$$\max_{s \in \mathcal{S}} \left(d^s_{\text{path}} + d^s_{\text{interference}} \right) + t_{\text{cycle}} - t_w \leq t_{\text{cycle}} \tag{5}$$

$$\Leftrightarrow \qquad t_w \geq \max_{s \in \mathcal{S}} \left(d^s_{\text{path}} + d^s_{\text{interference}} \right) \tag{6}$$

follows. Starting from Eq. 6, we can successively derive the closed form (coarse) upper bound on t_w

$$t_{\text{w}} \geq \max_{s \in \mathcal{S}} \left(d^s_{\text{path}} + d^s_{\text{interference}} \right) \geq \max_{s \in \mathcal{S}} \left(d^s_{\text{path}} \right) + \max_{s \in \mathcal{S}} \left(d^s_{\text{interference}} \right) \tag{7}$$

by separately considering upper bounds on d^s_{path} and $d^s_{\text{interference}}$. The path delay d^s_{path} can be bounded by considering the longest path and the largest stream size in the network, i.e.,

$$\max_{s \in \mathcal{S}} \left(d^s_{\text{path}} \right) = \max_{s \in \mathcal{S}} \left(\left(|P_S(s)| \right) \cdot \left(d_{\text{f-ind}} + d_{\text{prop}} + d^s_{\text{f-dep}} \right) \right)$$

$$+ \left(d_{\text{prop}} + d^s_{\text{f-dep}} \right) \tag{8}$$

$$\leq \max_{s \in \mathcal{S}} \left(\left(|P_S(s)| + 1 \right) \cdot \left(d_{\text{f-ind}} + d_{\text{prop}} + d^s_{\text{f-dep}} \right) \right) \tag{9}$$

$$\leq \max_{s \in \mathcal{S}} \left(|P_S(s)| + 1 \right) \cdot \left(d_{\text{f-ind}} + d_{\text{prop}} + \max_{s \in \mathcal{S}} \left(d^s_{\text{f-dep}} \right) \right). \tag{10}$$

Similarly, we assume that in the worst-case, a frame of stream

s is delayed by all other streams which are transmitted on each egress port on the stream's path, to get a (coarse) upper bound on $d^s_{\text{interference}}$. Furthermore, we take into account that in the worst-case, all talkers transmit in cycle *i* after the gates closed and in cycle *i + 1* before the gates open (cf. Fig. 4). Thus, in the worst case one frame of stream s has to wait for *two* instances of every other stream, i.e., we have

(11)
$$\max_{s\in\mathcal{S}}(d^s_{\text{interference}}) \leq 2\cdot\max_{s\in\mathcal{S}}\left(\sum_{e\in P_E(s)}\sum_{s'\in S_E(e)} d^{s'}_{\text{f-dep}}\right)$$

(12)
$$\leq 2\cdot\max_{s\in\mathcal{S}}(|P_E(s)|)\cdot\max_{e\in P_E(s)}\left(\sum_{s'\in S_E(e)} d^{s'}_{\text{f-dep}}\right).$$

Inserting Eq. 10 and Eq. 12 into Eq. 7 thus yields

(13)
$$t_w \geq 2\cdot\max_{s\in\mathcal{S}}(|P_E(s)|)\cdot\max_{e\in P_E(s)}\left(\sum_{s'\in S_E(e)} d^{s'}_{\text{f-dep}}\right) + \max_{s\in\mathcal{S}}(|P_S(s)|+1)\cdot\left(d_{\text{f-ind}}+d_{\text{prop}}+\max_{s\in\mathcal{S}}\left(d^s_{\text{f-dep}}\right)\right)$$

i.e., t_w is sufficiently large to guarantee that every stream can reach its destination taking into account the length of the path and the potential interference with (in the worst-case two frames of) all other streams in the class of s. This formula provides a coarse upper bound for the size of t_w. From our point of view, there are three major points to achieve a tighter bound: first, calculating $\max(d^s_{\text{interference}})$ on a per hop basis. Secondly, limiting the the transmission jitter allowed by the `talker` model. Thirdly, using traffic shaping and policing accordingly to enforce a more predictable behavior.

Class-based scheduling gives guarantees in the range of one cycle time e2e delay and jitter. These guarantees are sufficient for cyclic traffic, and by deriving a simple formula for calculation of t_w, we showed that the planning effort for class-based scheduling is low. Furthermore, the `talker` only needs coarse time synchronization. As result, class-based scheduling allows enabling and disabling `talkers` dynamically at runtime because smaller synchronization accuracy needs less time for synchronization. Moreover, class-based scheduling needs few GCL entries (one open and one close entry per scheduled class). Nevertheless, class-based scheduling requires large high-priority buffers at the switches since no dropping of realtime streams must occur, and in the worst case two instances of every stream can be in the network at the same time. The bandwidth efficiency of class-based scheduling is low since for the calculation of t_w, the worst case queuing is taken into account leading to a large class-window. The idling periods during t_w cannot be used by best-effort traffic.

In summary, class-based scheduling provides a trade-off between end-to-end delay and jitter guarantees, synchronization overhead, and planning effort, but comes at the cost of looser bounds and the requires large buffers.

C. Stream-based Scheduling

Stream-based scheduling is another way to use the TAS. In contrast to class-based scheduling, it considers each stream individually rather than groups of streams. Therefore, it is well-suited for *isochronous* traffic. To implement temporal isolation for each stream, stream-based schedules requires that the `talker` cycle is aligned to the network cycle and the transmission start of the `talker` is precisely configurable. A global configuration tool precisely plans the transmission start and route for each stream. This approach guarantees minimal queuing delays by isolating streams temporally and spatially through configured GCLs and routes. Since the queuing delay is minimal or even non-existing (zero-queuing), the *path delay* of a stream often dominates the e2e delay leading to a high deterministic behavior of the stream. For detailed descriptions of the implementation of such a global configuration tool and algorithms, we refer to Section II.

Due to the fact that, by design, no or only small queuing of stream-based scheduled frames occurs, the switches do not need large buffers. Furthermore, bandwidth efficiency is high because the gates for the best effort traffic are only closed for the time the high priority traffic is being forwarded. On the other hand, the planning of stream-based scheduling is an NP-hard problem (cf. Sec II). Especially for large network topologies and many streams, the planning process is challenging. In addition, the involved entities need time synchronization with very high accuracy, such that the `talker` application provides the data just in-time and the network can deliver the data within the reserved time slots. Already small deviations from the schedule can lead to cascading violations of all deadlines, resulting in downtime for resynchronization. To prevent such situations, all streams need to be strictly policed. Moreover, stream-based scheduling requires two gate operations per stream on each switch on its path and, thus, the number of GCL entries is larger than for class-based scheduling.

In conclusion, stream-based scheduling is well-suited for the scheduling of *isochronous* streams, but it comes at the cost of high planning complexity, and with practical scaling issues (planning, length of GCL).

V. Simulative Analysis

While we can analytically derive best- and worst-case bounds of the scheduling mechanisms, a simulation tool is useful to investigate statistical properties for specific scenarios, or the effects on soft-real time or best-effort traffic. For the simulative analysis, we use discrete-event simulation with NeSTiNg. NeSTiNg [1] extends the OMNeT++/INET stack with TSN capabilities, including frame preemption, TAS, and credit-based shaping. In this paper, we use NeSTiNg in conjunction with OMNeT++ 5.4.1 and INET 4.1.0.

A. Evaluation Scenario

For the simulation of the three discussed approaches, we use line topologies (cf. Fig. 3) with size $n_l \in (10{,}20{,}30{,}50)$. Link bandwidth is set to $1\,\text{Gbit/s}$ for all links. For each value of n_l, we randomly generated 30 different scenarios, and for each

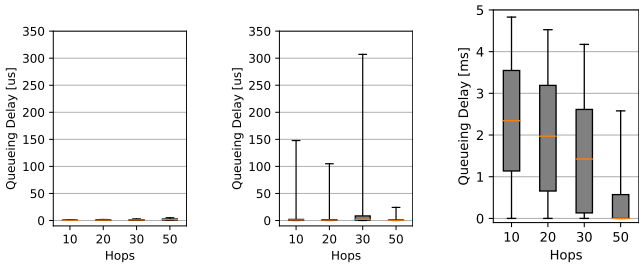
scenario we randomly generated n_l streams, i.e., there are as many streams as hosts in the scenario. Streams are generated as follows: We randomly choose a `talker` from $\mathcal{V}_H$, and then randomly choose the `listener` from $\mathcal{V}_H \backslash$`talker`. In case of stream-based scheduling, every host is the source of one single stream, but can be `listener` of multiple streams. The stream size, i.e., the Ethernet payload is drawn uniformly at random from the interval $[64\text{B}, 300\text{B}]$. For the evaluation with frame preemption, we add one host at each end of the line topology sending preemptible best-effort frames at line rate to the host on the other side.

B. Simulation Results

Figure 5 depicts the accumulated queuing delay for the different packet scheduling approaches. The accumulated queuing delay is the total time which a single frame spends waiting in the queues on its path from talker to listener. In Fig. 5a, we can see that stream-based scheduling can reliably achieve near zero-queuing with a maximal accumulated queuing delay of $5.1\,\mu$s in our evaluation scenarios.

If we look at the results for frame preemption (cf. Fig. 5b), we can see that in our evaluation scenarios, frame preemption yields low average values for the accumulated queuing delay (in the order of few µs). However, we also observe worst-case values of more than $300\,\mu$s, which are orders of magnitudes larger compared to the average accumulated queuing delay. That is, even with a single traffic class using express frames, there is a variance in the e2e delay caused by varying queuing delay. We also observe that a larger value of n_l does not necessarily result in a smaller queuing delay. This can be caused by the random stream distribution. Our statistical evaluation shows that it is hard to make a statement about the behavior of frame preemption.

Lastly, Fig. 5c shows the results for class-based scheduling. As in Fig. 5b, we see that a bigger value of n_l results in smaller values of the accumulated queuing delay. Besides the random stream distribution, there are two reasons for this. Firstly, since t_{cycle} is constant ($5\,\text{ms}$), but the network diameter and the number of scheduled streams increases, the class-window t_w becomes bigger for larger n_l, i.e., the queuing delay caused by arriving when the gate is closed becomes smaller. Secondly, with larger n_l, the value of d^s_{path} grows, i.e., the size of t_w increases proportional reducing the probability of transmissions during the gate closed period.



(a) Stream-based (b) Frame preempt. (c) Class-based

Fig. 5: Evaluation results from NeSTiNg simulation.

VI. Conclusion

We presented class-based and stream-based scheduling as two ways to use the TAS, and frame preemption as third alternative. The discussion of the approaches showed that each approaches has its strengths and weaknesses. Stream-based scheduling is well-suited for *isochronous* traffic but cannot cope with the loosened `talker` model of *cyclic* traffic. Class-based scheduling is easy to plan but due to its large class-window, but the bandwidth efficiency is low. Frame preemption does not require planning or synchronization but the guarantees depend strongly on the knowledge about the network. In conclusion, each approach can serve a specific use case. In modern factories different requirements on the network exist. Since none of the discussed approaches is capable to serve all use cases, a combination of these approaches is required. Therefore, assessing interactions between these and additional approaches, provides ample opportunity for further research.

References

[1] J. Falk, D. Hellmanns, B. Carabelli, N. Nayak, F. Dürr, S. Kehrer, and K. Rothermel, "NeSTiNg: Simulating IEEE time-sensitive networking (TSN) in OMNeT++," in *Proceedings of the 2019 Int. Conf. on Networked Systems (NetSys)*, Garching b. München, Germany, Mar. 2019.

[2] L. L. Bello and W. Steiner, "A Perspective on IEEE Time-Sensitive Networking for Industrial Communication and Automation Systems," *Proceedings of the IEEE*, vol. 107, no. 6, Jun. 2019.

[3] J. L. Boudec, "A Theory of Traffic Regulators for Deterministic Networks With Application to Interleaved Regulators," *IEEE/ACM Transactions on Networking*, vol. 26, no. 6, Dec. 2018.

[4] J. Cao, P. J. L. Cuijpers, R. J. Bril, and J. J. Lukkien, "Independent WCRT analysis for individual priority classes in Ethernet AVB," *Real-Time Systems*, vol. 54, no. 4, Oct. 2018.

[5] L. Zhao, P. Pop, and S. S. Craciunas, "Worst-Case Latency Analysis for IEEE 802.1Qbv Time Sensitive Networks Using Network Calculus," *IEEE Access*, vol. 6, 2018.

[6] F. Dürr and N. G. Nayak, "No-wait Packet Scheduling for IEEE Time-sensitive Networks (TSN)," in *Proceedings of the 24th Int. Conf. on Real-Time Networks and Systems*, ser. RTNS '16, 2016.

[7] F. Pozo, G. Rodriguez-Navas, and H. Hansson, "Schedule Reparability: Enhancing Time-Triggered Network Recovery Upon Link Failures," in *2018 IEEE 24th Int. Conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Aug. 2018.

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

[9] N. G. Nayak, F. Dürr, and K. Rothermel, "Routing Algorithms for IEEE802.1Qbv Networks," *SIGBED Rev.*, vol. 15, no. 3, Aug. 2018.

[10] Jonathan Falk, Frank Dürr, and Kurt Rothermel, "Exploring Practical Limitations of Joint Routing and Scheduling for TSN with ILP," in *2018 IEEE 24th Int. Conf. on Embedded and Real-Time Computing Systems and Applications*, Hakodate, Japan, Aug. 2018.

[11] I. C. Society, "IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks," *IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014)*, Jul. 2018.

[12] IEEE Computer Society, "IEEE Standard for Ethernet," *IEEE Std 802.3-2018 (Revision of IEEE Std 802.3-2015)*, Aug. 2018.

[13] ——, "IEEE Standard for Local and Metropolitan Area Networks - Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks," *IEEE Std 802.1AS-2011*, Mar. 2011.

[14] IIC, "Time Sensitive Networks for Flexible Manufacturing Testbed-Description of Converged Traffic Types," Tech. Rep., 2018.

[15] "IEEE standard for local and metropolitan area networks–bridges and bridged networks – amendment 31: Stream reservation protocol (srp) enhancements and performance improvements," *IEEE Std 802.1Qcc-2018 (Amendment to IEEE Std 802.1Q-2018 as amended by IEEE Std 802.1Qcp-2018)*, Oct 2018.