FlexTAS: Flexible Gating Control for Enhanced Time-Sensitive Networking Deployment

Jiashuo Lin, Weichao Li, Xingbo Feng, Shuangping Zhan, Lewei Ning, Yi Wang, *Member, IEEE*, Tao Wang, Hai Wan, Bo Tang, and Xiaofeng Tao, *Senior Member, IEEE*

***Abstract*—Time-sensitive networking (TSN), essential in industrial networks for its promise of reliable and deterministic data transmission, faces deployment challenges due to the limitations of existing time-aware shaper (TAS)-based scheduling algorithms. Specifically, the size of the generated gate control lists (GCLs) is usually too large to be deployed in actual devices. To bridge the gap between theory and practice, we propose FlexTAS, a flexible and practical solution for TSN. The key insight behind FlexTAS is that relaxing gating does not introduce uncertainty, as long as nonoverlap reserved time slots are guaranteed. FlexTAS is comprised of two main components: first, a novel gating model deviates from the conventional TAS model by incorporating selective relaxation of gating at certain nodes; and second, a deep reinforcement learning-based engine to rapidly generate valid schedules. We build a real testbed and validate the effectiveness of our proposed solution. Our evaluation demonstrates that FlexTAS effectively controls the number of gate entries within the GCL capacity of devices, while simultaneously meeting the Quality of Service(QoS) requirements of time-triggered streams. It significantly reduces the number of GCL entries by 60% to 80% , and facilitates deployment in heterogeneous networks, thus offering a practical solution for TSN.**

Received 22 October 2024; revised 13 February 2025; accepted 12 March 2025. This work was supported in part by the Guangdong Basic and Applied Basic Research Foundation under Grant 2024A0101010001, in part by the Major Key Project of PCL under Grant PCL2023AS1-3 and Grant PCL2023A03, in part by the Guangdong High-Level Talents Special Support Program under Grant 2021TX05X205, in part by the National Key Research and Development Program of China under Grant 2020YFB1806400, and in part by the PCL-CMCC Foundation for Science and Innovation under Grant 2024ZY1A0010. Paper no. TII-24-5555. *(Corresponding authors: Weichao Li; Yi Wang; Bo Tang.)*

Jiashuo Lin and Yi Wang are with the Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, China, and also with Pengcheng Laboratory, Shenzhen 518055, China (e-mail: linjs2021@mail.sustech.edu.cn; wy@ieee.org).

Weichao Li, Shuangping Zhan, Lewei Ning, and Xiaofeng Tao are with Pengcheng Laboratory, Shenzhen 518055, China (e-mail: liwc@pcl.ac.cn; zhanshp@pcl.ac.cn; ninglw@pcl.ac.cn; taoxf@bupt.edu.cn).

Xingbo Feng and Bo Tang are with the Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, China (e-mail: xingbfeng3-c@my.cityu.edu.hk; tangb3@sustech.edu.cn).

Tao Wang is with the School of Automation, Guangdong University of Technology, Guangzhou 510006, China (e-mail: wangtao_cps@gdut.edu.cn).

Hai Wan is with the School of Software, Tsinghua University, Beijing 100084, China (e-mail: wanhai@tsinghua.edu.cn).

Digital Object Identifier 10.1109/TII.2025.3554251

***Index Terms*—Deterministic network, flexible gating control, network scheduling, time-sensitive networking (TSN).**

I. Introduction

THE industrial Internet of Things, crucial in manufacturing and factory automation, relies on specialized networks for data exchange among sensors, actuators, and production devices. As Industry 4.0 unfolds and smart devices become ubiquitous, there's a growing need for highly collaborative and intelligent production systems, which traditional dedicated networks cannot adequately support. Consequently, the IEEE's time-sensitive networking (TSN) task group is developing real-time communication mechanisms on top of existing Ethernet standards, specifically tailored for industrial networks [1]. These emerging TSN standards, currently in the drafting phase, represent a significant advancement in meeting the real time, reliable data communication needs of modern industrial environments.

Achieving deterministic data transmission in TSN hinges on the synthesis of gate control lists (GCLs), which regulate the gates of switches and determine the exact transmission timing of frames in the network. Coupled with precise time synchronization mechanisms, this setup enables submicrosecond precision in controlling queues across all network ports, thereby ensuring the deterministic forwarding of time-sensitive traffic. However, as the number of time-triggered (TT) streams increases, the required GCL length can dramatically exceed the typical switch capacity, which ranges from 8 to 1024 entries [2], [3], [4]. Fig. 1 shows the rapid growth in GCL length as TT stream numbers increase, using streams from *IEC/IEEE 60802 use cases* (c.f. [1, Fig. 25, 26, 29]). The GCL capacity of commercial switches, typically 256 or 1024, can support only up to 60 streams from `Stream class 1`, and a mere 15 streams when capacity is at 256, which is insufficient for large networks managing hundreds of TT streams. Existing scheduling algorithms [5], [6], [7] often overlook the practical constraint of GCL length, leading to scalability challenges in real-world deployments.

Moreover, existing algorithms often fall short in supporting mixed heterogeneous networks. TSN, as a new paradigm, introduces new features such as time-aware shaper (TAS) [8], which are not typically supported by legacy switches. Most existing research assumes a homogeneous network where all devices are time-aware and TSN-capable. This assumption implies that all devices without TSN support must be replaced with TSN-enabled ones, leading to significant deployment costs. To

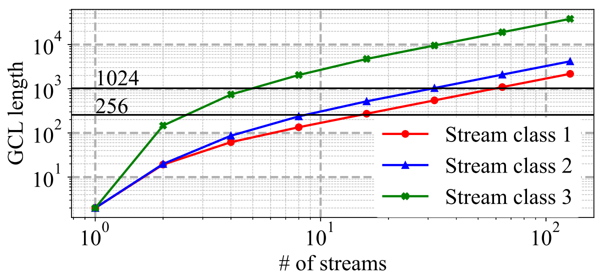


Fig. 1. Required GCL length exceeds switch capacity rapidly due to varying stream cycles. The two horizon lines represent the typical GCL capacity of commercial switches.

facilitate a smooth transition from traditional to deterministic networks, there is a need for a solution that ensures deterministic networking in heterogeneous environments, coexisting TSN and non-TSN devices.

In this article, we present FlexTAS, a practical solution that significantly reduces the required number of GCL entries while ensuring bounded latency and jitter, and meeting the quality of service (QoS) requirements of TT streams. The fundamental idea behind FlexTAS is intuitive: by enabling gate control selectively for only a subset of TT streams and leaving the gate continuously open for others, we effectively reduce the required GCL length. However, enabling gate control for only some streams introduces challenges in ensuring deterministic transmission.

1) *Accumulated jitter:* Without appropriate gating, TT frames may experience delays caused by non-TT frames. This jitter accumulates with each hop, potentially breaching the QoS thresholds specified for TT streams.
2) *Inter-stream interference:* In the absence of gating, frames from different TT streams may interfere with each other, resulting in unpredictable network behavior.
3) *Scheduling complexity:* Relaxing gating rules complicates the scheduling process.

The scheduling algorithm must now effectively manage not only the timing and order of TT streams but also the gating control, which adds layers of complexity to the network management tasks.

FlexTAS comprises two key components—a flexible gating model and a deep reinforcement learning (DRL)-based scheduling engine—to tackle these challenges. The flexible gating model allows selective gating control at various nodes, significantly reducing the length of GCLs. Difference from traditional TAS models, FlexTAS employs a `close-open` gating model for TT stream queues, which keeps the queue open, allowing normal transmission of frames while providing flexible control to eliminate jitter only when needed. Furthermore, the DRL-based scheduling engine addresses the challenge of joint scheduling and gating deployment problems, enabling swift and efficient solutions resolving. Although the relaxation of gate control may result in additional delay and jitter, we show that the end-to-end delay and jitter still remain bounded, as long as scheduling is managed appropriately by the DRL-based scheduling engine to avoid overlap in reserved time slots for TT streams. By flexibly enabling gating at various nodes for different TT streams,

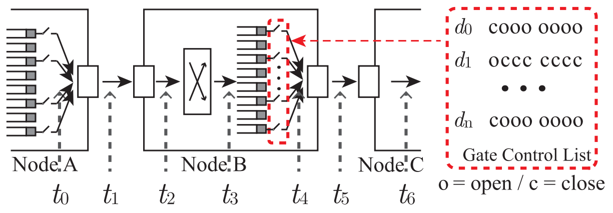


Fig. 2. TSN switch workflow. Frames are forwarded only when the gate is open (denoted as “o”), which is precisely controlled by the GCL.

FlexTAS effectively addresses the *GCL expansion* issue arising from varying stream cycles (c.f. Section II), while still preserving the QoS of TT streams.

Overall, our contributions can be summarized as follows.

1) We introduce a novel gating model that brings flexibility in selecting nodes for enabling gating control, along with a detailed analysis of its timing model compared to traditional methods. We show that it can save up to 80% GCL requirements compared to the traditional TAS-based model. (c.f. Section III-B)
2) Building on the flexible gating model, we design a DRL-based scheduling engine to quickly generate valid schedules. The DRL agent is developed with a generalized design, enhancing its robustness to function effectively across diverse network environments and varying traffic conditions. We open source our implementation[1] to contribute to the community and promote further research and development in this area. (c.f. Section III-C)
3) We build a real-world testbed to validate the correctness of our proposed novel gating model and conduct experiments to further analyze the performance of our proposed method. The experiment results confirm the efficiency of FlexTAS, and demonstrate its capability to facilitate deployment in heterogeneous networks. (c.f. Section IV).

II. Preliminary

A. Background

We begin by introducing the TSN background, followed by a discussion on the resulting *GCL expansion* issue.

TSN concepts: We consider periodic critical applications in this article, which are synchronized to the network clock and transmit traffic (denoted as `TT streams`) periodically. For example, in an industrial network, a sensor may periodically transmit sensing data to the controller, which sends the control signal to executors [1]. In TSN, these TT streams should be scheduled properly to ensure that the end-to-end delay and jitter are bounded and satisfies the requirements of the application. To achieve this, TSN provides a bunch of tools, including synchronization [9], preemption [10], scheduling [8], etc. We focus on IEEE 802.1qbv [8], which defines TAS, which is the basic mechanism of TSN, as shown in Fig. 2. It introduces a time-aware gating control mechanism, where the gate of a port is controlled by a GCL to ensure that frames are transmitted

[1][Online]. Available: https://github.com/yue2388253/FlexTAS

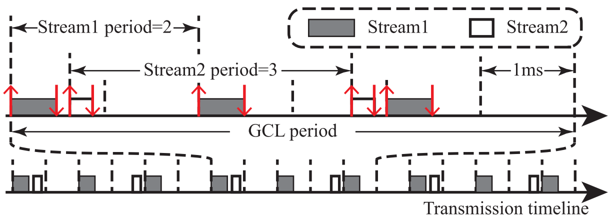


Fig. 3. *GCL expansion:* For two TT streams with distinct periods, a minimum of 10 gating control entries is required, each represented as a red arrow.

at deterministic time moments, coupled with clock synchronization. By properly scheduling the streams to achieve strict isolation between TT streams, thus avoiding competition, TSN can guarantee the deterministic forwarding of TT streams. For example, the time slots allocated for the streams shown in Fig. 3 must NOT overlap, a concept referred to *temporal isolation* [5]. This is to prevent TT streams from interfering with each other, thereby ensuring that each stream transmits its data without disruptions.

TSN workflow: In the data plane, the workflow of transmitting a TT frame along a TAS-enabled switch is as follows, as shown in Fig. 2.

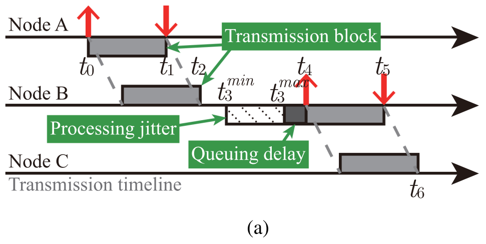
1) The TT frame starts to be transmitted from Node A toward Node B at t_0, and the transmission is finished at t_1.
2) After a link propagation delay d_{prop}, the frame is received at $t_2 = t_1 + d_{\text{prop}}$, and is stored in the switch buffer.
3) The frame is processed by the switch fabric and is then enqueued at $t_3 = t_2 + d_{\text{proc}}$.
4) At t_4, the port starts to transmit the frame and finishes at t_5.
5) Finally, the frame is received by Node C at $t_6 = t_5 + d_{\text{prop}}$.

We denote the minimum and maximum value of time moment t_i as $t_i^{\min}$ and $t_i^{\max}$, respectively. For example, since the processing time of a device d_{proc} can fluctuate within a range $[d_{\text{proc}}^{\min}, d_{\text{proc}}^{\max}]$, and the clock synchronization between devices can have a deviation δ, the actual moment t_3 the frame is put into the queue is bounded within $[t_3^{\min}, t_3^{\max}]$, where

$$t_3^{\min} = t_2^{min} + d_{\text{proc}}^{\min} - \delta t_3^{\max} = t_2^{\max} + d_{\text{proc}}^{\max} + \delta. \quad (1)$$

GCL expansion issue: The periodic TT streams can have different periods depending on the overlaying applications, and cannot be changed with reasonable effort [1]. This variability in TT stream periods can lead to an excessive number of GCLs, surpassing the hardware capacity of actual TSN devices. Fig. 3 provides a simple example by showing the transmission timeline at a port with two TT streams having different periods of 2 and 3 units of time, respectively. Thus, the GCL cycle is 6, the least common multiple of their periods. Since 2 gate control entries are needed for each instance—one for opening and the other for closing—a minimum of 10 GCL entries is required.[2] This GCL expansion issue becomes more severe

[2]We do not consider GCL merging for the sake of simplifying the description. In fact, the effectiveness of GCL merging is limited, as it does not fundamentally resolve the GCL expansion issue.



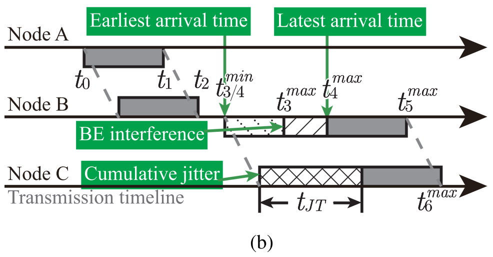


Fig. 4. Per-hop delay models of traditional methods. (a) TAS: Precise control of frame forwarding via gating. (b) Legacy SP: Lack of gating control leads to inherent jitter.

with the variety of periods. For instance, IEC/IEEE 60802 profile [1] provides use cases where TT streams have different periods of $\{62.5\ \mu\text{s}, 1\ \text{ms}\}$, $\{31.25\ \mu\text{s}, 50\ \mu\text{s}, 125\ \mu\text{s}, 200\ \mu\text{s}\}$; and $\{31.25\ \mu\text{s}, 250\ \mu\text{s}, 1\ \text{ms}, 16\ \text{ms}\}$. In such a case, the required GCL length can grow dramatically, making the deployment of TSN impractical, as shown in Fig. 1.

B. Motivation

We begin by analyzing the time slots for frame forwarding in TAS, noting that the gating of high-priority queues exhibits a one-to-one mapping with TT streams [3], [5], [11]. This mapping implies that each TT frame is aligned with a distinct gating window, as demonstrated in Fig. 3. Fig. 4(a) illustrates the detailed gating operation for the forwarding of a TT frame at a specific hop. To ensure that the frame is transmitted at a precise moment t_4 from Node B toward Node C, the gate should not be opened before t_3^{max}, such that we can guarantee that the frame has been already put into the queue when the gate is opened at t_{open}, i.e., $t_4^{\min} = t_4^{\max} = t_{\text{open}} \geq t_3^{\max}$.

Next, we analyze scenarios without gating, using the legacy strict priority (SP) model for example. As illustrated in Fig. 4(b), a TT frame can be transmitted immediately upon entering the queue ($t_4^{\min} = t_3^{\min}$). However, it may also experience delays due to interference from unscheduled BE traffic. In such cases, if a BE frame is already being transmitted, the TT frame must wait until the transmission is complete. The maximum possible waiting time t_{IF} corresponds to the transmission duration of the largest BE frame, which is typically 1518B, or 123B in the case of frame preemption [10], plus the interframe gap (IFG). This leads to an additional jitter per hop t_{JT} that consists of device processing variation, synchronization deviation, and BE traffic interference

$$t_4^{\min} = t_3^{\min} \quad \text{and} \quad t_4^{\max} = t_3^{\max} + t_{\text{IF}}$$
$$t_{\text{JT}} = d_{\text{proc}}^{\max} - d_{\text{proc}}^{\min} + 2\delta + t_{\text{IF}}. \quad (2)$$

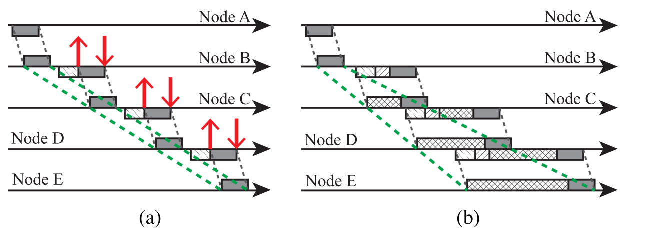


Fig. 5. TAS versus Legacy SP: 5-hop delay models. (a) TAS: Zero-jitter is achieved via per-hop gating operations. (b) Legacy SP: Additional jitter per hop leads to significant end-to-end jitter.

In the analysis above, we assume no competing interference delay, as appropriate scheduling can be applied to ensure strict isolation constraints [5] between TT streams are met, thus preventing competition [2], [6], [7]. For legacy SP, although it lacks gating to control the exact forwarding time of frames at each hop, isolation between TT streams can still be achieved by controlling the timing of frame injection into the network.[3]

Fig. 5 provides an intuitive comparison between TAS and legacy SP in a 5-hop link time slot model, characterized by a no-wait transmitting pattern (i.e., $t_{\text{open}} = t_3^{\max}$) [11]. In this setup, the TAS model guarantees zero jitter due to its per-hop gating operations. In contrast, the legacy SP model introduces additional jitter at each hop, which accumulates to a significant level of jitter at the receiving end. The differences between the two models are visually highlighted by the green dashed lines in the figure. In conclusion, we have the following key insight. For the correctness, we have constructed a real-world testbed to validate the analysis result. (c.f. Section IV).

Takeaway: Relaxing gating control introduces additional per-hop jitter, accumulating along the path and enlarging the reserved time slot, potentially impacting competition with other TT streams. Nonetheless, the end-to-end jitter remains bounded, provided these reserved time slots do not overlap.

III. FlexTAS Design

This section presents the core design of FlexTAS, which comprises two key components: a novel flexible gating model and a DRL-based scheduling engine.

A. Overview

The key insight of FlexTAS is that *instead of deploying gating operations for all streams at all hops, it is more efficient to select a subset of streams and hops for gating.* Specifically, for each frame at each hop, the FlexTAS gating model (c.f. Section III-B) offers two options: a) When gating is NOT enabled, the forwarding process is similar to the legacy SP model, as discussed above. b) When gating is enabled, FlexTAS employs a novel `close-open` gating operation, which contrasts with the traditional `open-close` operation in TAS. The decision on whether to enable gating at each hop for each frame is managed by the DRL-based scheduling engine (see Section III-C), which

[3]This can be achieved by deterministic terminals [12] or by adding shaping gateways, which is beyond the scope of this article.

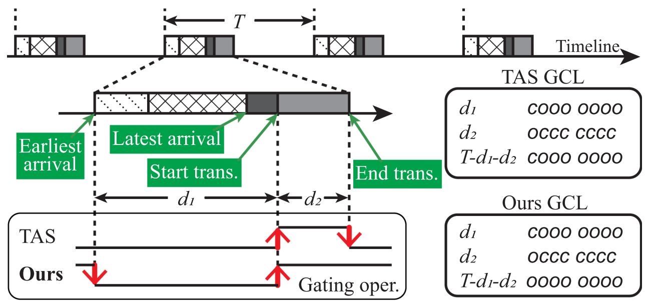


Fig. 6. TAS's `open-close` versus FlexTAS's `close-open` gating.

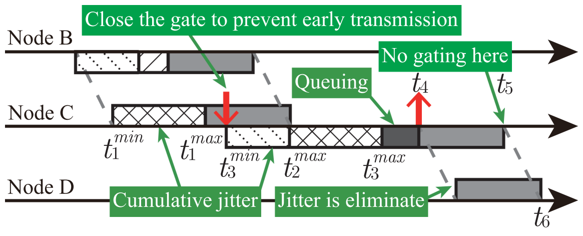


Fig. 7. FlexTAS can eliminate the cumulative jitter via `close-open` gating.

also determines the TT frames' injection times and the precise gating moments if gating is enabled, ensuring that the QoS requirements are met, and GCL length is under the device's capacity.

B. Flexible Gating Model

We develop a novel flexible gating model that enables network operators to selectively and flexibly choose, which streams and at which hops to deploy gating. Unlike the traditional TAS model's `open-close` operation, this model employs an innovative `close-open` gating strategy. Fig. 6 shows the difference of the two gating operations, as well as their GCLs. As depicted, FlexTAS keeps gates in an open state by default, closing them only when necessary. When gating is required to manage specific TT streams, our model achieves precise control and eliminates jitter by initially closing the gate at the earliest possible arrival time of the TT frame. The gate then remains closed until the frame's latest possible arrival time to ensure that the frame has already been enqueued. After transmitting a TT frame, the TT queue remains open, while the lower priority queues are opened to resume data transmission. This approach ensures that TT frames not requiring gating are not delayed by closed gates, enabling normal forwarding similar to Legacy SP.

Fig. 7 illustrates the timing analysis of the flexible gating model. A frame has accumulated some jitter at Node B due to the absence of gating operations in the preceding path (similar to legacy SP in Fig. 5). This jitter is eliminated by enabling gating operations at Node C. Specifically, the gating operations are as follows: i) the gate is first closed no later than the earliest possible arrival time $t_3^{\min}$ to prevent potential early transmission; ii) the gate remains closed until the latest possible arrival time $t_3^{\max}$. Once the gate is opened at $t_4 \geq t_3^{\max}$, the frame is guaranteed to be in the queue and is forwarded immediately. This gating

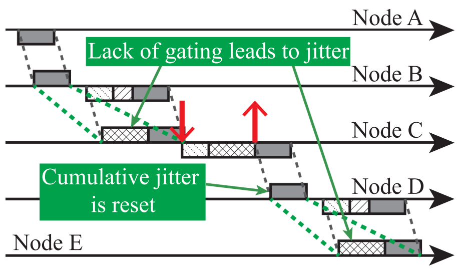


Fig. 8. FlexTAS along a 5-hop path: Gating at Node C eliminates accumulated jitter and reduces reserved slot length.

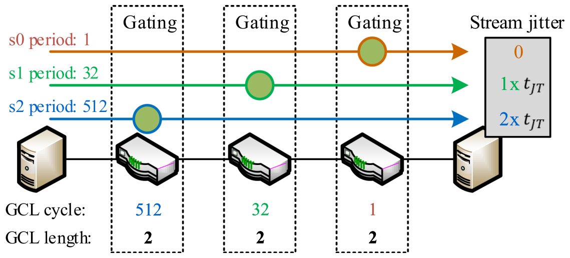


Fig. 9. FlexTAS use case: At each node, gating is applied to a single stream, minimizing the required gating entries while meeting the QoS requirements.

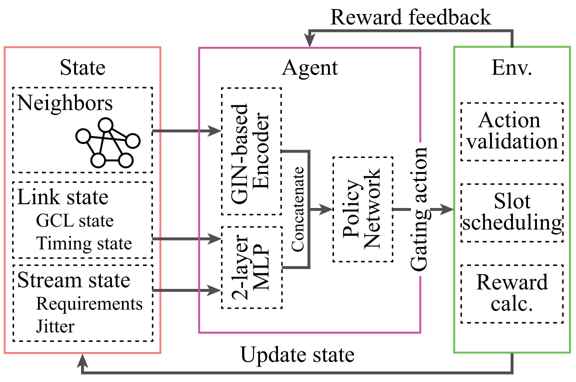


Fig. 10. DRL-based scheduling engine.

eliminate the accumulated jitter and ensures the precise control over the forwarding time at Node C: no matter when the frame arrives, it will be forwarded at t_4. As a result, the frame arrives at Node D without any jitter, and the reserved time slot is also significantly reduced to the minimal time slot.

Fig. 8 shows a sample time slot allocation of FlexTAS along a 5-hop path, where gating is deployed only at the intermediate node, Node C. The jitter experienced at the receiving end originates solely from BE traffic interference at the final hop, Node D. FlexTAS eliminates the need for gating at every hop, thus effectively reducing the GCL length.

Additional jitter: In case that gating is not enabled, FlexTAS can introduce an additional jitter of t_{JT} per hop, similar to Legacy SP, as discussed in (2). However, it is crucial to remember that the original goal of TSN is NOT necessary to achieve the lowest possible latency and jitter, but to provide strictly bounded and reliable network services. Two key considerations must be noted: first, this additional jitter introduced is still bounded above by t_{JT} (c.f. Section II). Second, for streams requiring zero jitter, implementing a gating operation at the final hop can eliminate any accumulated jitter from the stream's preceding path. The remaining challenge is how to arrange gating deployment, and also the time slot scheduling, such that *the introduced jitter and delay still align with the QoS requirements*, which will be addressed by the DRL-based scheduling engine, as will be discussed in the following section.

Isolation: To ensure deterministic transmission, frame isolation should be guaranteed: *there are only frames of one flow in the queue at a time* [5]. For TAS-based solutions [5], [6], [11], since the arrival time of frames is always predetermined, the reserved time slots can be computed straightforwardly. However, in FlexTAS, some TT frame instances are not gated, making their arrival time uncertain. Their earliest start time and latest end times falling within the intervals $[t_3^{\min}, t_3^{\max}]$ and $[t_5^{\min}, t_5^{\max}]$, respectively. Therefore, the time slot reserved for each frame must span from $[t_3^{\min}, t_5^{\max}]$ to ensure that strict isolation constraints are not violated.

Use case: Fig. 9 demonstrates an example of FlexTAS addressing the GCL expansion issue (c.f. Section II), featuring three streams with cycles of 31.25 μs, 1 ms, and 16 ms, normalized to 1, 32, and 512, respectively. The jitter QoS requirements for these streams are 0, 15 μs, and 30 μs, respectively, and the additional jitter per hop $t_{\mathrm{JT}} = 10$ μs. In such a case, a traditional TAS solution would require 1058 GCL entries, exceeding device capabilities. In contrast, the legacy SP approach would result in a jitter of 3$\times$ t_{JT} for each stream, failing to meet application requirements. FlexTAS, by enabling gating for only one stream per node, reduces the GCL requirement to just 2 entries per node, delivering a remarkable improvement in efficiency and scalability for TSN systems.

C. DRL-Based Scheduling Engine

While the flexible gating model improves network efficiency through selective gating, a key challenge is determining how to schedule TT streams, such that the QoS requirements are satisfied, while also respecting the limitations of actual device GCL capacity. Consider the gating deployment in the example network shown in Fig. 9, each stream has 2^3 gating deployment options, leading to $(2^3)^3 = 512$ potential solutions, while most of which are invalid—either violating QoS requirements or exceeding GCL capacities. In addition, the NP-hard scheduling problem of TT streams [6] must also be considered to ensure nonoverlap reserved time-slot, making the problem even more complex. To address these issues, we propose a DRL-based scheduling engine, as illustrated in Fig. 10. The use of DRL is motivated by its ability to navigate complex decision spaces and find feasible solutions in environments where traditional algorithms may fail due to the high dimensionality and constraints, as demonstrated in recent studies [7], [13], [14]. This scheduler utilizes a DRL framework similar to DeepScheduler [13] but uniquely incorporates gating variables, tailoring it to the specific demands of FlexTAS and making it a more practical solution for TSN environments.

1) Markov Decision Process (MDP) Formulation: The task of joint scheduling and gating deployment can be serialized into per-stream, per-hop decision-making, thereby formulating an MDP, as follows.

State: The observation state space of the agent encompasses the following three types of information.

1) The first type pertains to the current stream, which includes the period, the frame size, the jitter requirements, the cumulative jitter up to the current node, and the current hop index. In addition, we also present the link features along the path of the stream. Only features of the remaining hops are presented. This provides the agent with a comprehensive understanding of the current stream's characteristics and requirements.
2) The second type of information relates to the current link, including its link utilization rate, GCL cycle, GCL length, and statistical information about the streams in each cycle. This information allows the agent to understand the current state of the link and its capacity to schedule future streams.
3) The third type of information is about the graph.

First, the original topology is transformed into a line graph, a type of graph representation where each edge in the original graph is represented as a node, and two nodes in the line graph are connected if their corresponding edges share a common endpoint in the original graph. Second, to make the pretrained model more general and be able to cope with different topologies, instead of presenting all nodes of the topology, we only present the current node and its neighbors.

Action: We address the challenge of multidimensional actions, including time slot and gating scheduling, by decoupling them into action space and environment resolving. In particular, the agent only needs to decide *whether to enable gating for the current TT stream at the current node*, while the timing scheduling is delegated to the environment, as will be discussed in the following paragraph.

State transition: After the agent determines the gating action, i.e., whether to enable gating for the current stream at the current node, the environment (Env.) attempts to allocate a valid time slot for the ongoing operation, where *valid* means that the slot satisfies all constraints [5], such as isolation constraints, GCL capacity constraints, end-to-end delay constraints, etc. In particular, it seeks the earliest feasible time slots by employing the no-wait time-tabling algorithm [11], where the computation of timing moments is based on the delay analysis discussed in Section III-B.[4] Note that it is not always feasible to successfully find valid time slots for the stream. In this case, it terminates the scheduling trajectory and issues a penalty reward. Then, the environment reverts to its initial state and restarts the scheduling process. To enhance training efficiency, the order of the streams to be scheduled is shuffled each time the environment resets. This dynamic adjustment helps in developing more robust scheduling strategies under varying conditions.

[4]This might necessitate delaying prior operations of the stream to adhere to the no-wait condition.

Only when a stream is successfully scheduled at the final node are all its operations conclusively determined, and the timing moments at each hop are calculated. Once all streams are successfully scheduled, the scheduling process is considered complete. GCLs are then synthesized based on the scheduling results, i.e., the gating actions and the timing moments.

Reward: In the design of the reward function, the ultimate goal of the agent is to complete the scheduling of all streams. Initially, a simplistic reward function was considered, providing a reward only upon task completion. However, due to the NP-hard nature of TSN scheduling, this sparse reward approach hindered effective learning. Consequently, we employed reward shaping, whereby the agent earns a minor reward for successfully scheduling each stream at a node, provided all prior constraints are met

$$R(s_t, a_t) = 1 - \alpha \cdot \frac{g([u,v]_t, s_{t+1}) - g([u,v]_t, s_t)}{G([u,v]_t)} - \beta \cdot \frac{t_{\mathrm{IF}}}{\mathrm{f.d}} \tag{3}$$

where $g([u,v]_t, s_t)$ denotes the GCL length of link $[u,v]_t$ at t, $G([u,v]_t)$ denotes the link's GCL capacity, and f.d denotes the stream end-to-end delay requirement. This reward consists of three parts, a constant, the second represents the impact of the increased gate control entries, and the third represents the impact of interference if the agent choose not to enable gating control.

In addition, to encourage the agent to schedule all streams, whenever the agent completes scheduling of a small part (e.g., 10%) of the streams, an additional reward is given as follows:

$$R(s_t, a_t) = R(s_t, a_t) + \gamma \cdot P(s_t)^2 \tag{4}$$

where $P(s_t)$ denotes the percentage of streams that have been scheduled, and α, β, and γ are hyperparameter. This portion of the reward is proportional to the square of the number of scheduled streams. This is due to the fact that at the onset of each trajectory, scheduling streams is generally straightforward. At this stage, both GCL entries and slot space are relatively abundant, enabling the agent to easily complete the scheduling of the initial part of the stream. The crux of the challenge lies in the later stages of the trajectory, where there are limited GCL entries and slot space available.

2) Policy Network Design: The design of the policy network in our study comprises two parts: an encoder network and a policy network. The encoder network is to extract the features provided by the environment. We use graph isomorphism network (GIN) for encoding the neighbor features. GIN is a type of graph neural network that is capable of learning node representations that capture the graph structure around them, making it an effective tool for tasks that involve graph-structured data [15]. A two-layer fully connected network is used to encode the status information of the current stream and node. The outputs from these two processes are then concatenated and fed into the Policy network.

The policy network employs the proximal policy optimization (PPO) algorithm [16], a popular reinforcement learning method. PPO is an actor–critic method that uses a surrogate objective to improve the policy, with the advantage of being much simpler to implement, more sample-efficient, and more robust than other policy optimization methods.

3) Heterogeneous Networks: The flexible gating operation design of FlexTAS makes it suitable for deployment in heterogeneous networks, where there are non-TSN switches (or bridges) that do not support the gating mechanism. For these devices, we can simply disable all the gating control actions. The only requirement is that these devices should follow an SP transmission model, which is a common feature in modern commercial network devices. Therefore, all TT frames across these devices have a bounded wait time for the low-priority BE frames which are unscheduled, as discussed in Section III-B. The challenge is how we ensure that the TT frames do not interfere with each other at these devices. We tackle this challenge by simply treating these devices as a special case of the FlexTAS model, where the gating control actions are disabled. This can be easily achieved by masking the gating control action for these devices at the DRL scheduling engine.

D. Training

1) Enhance the Training Process: Training a DRL agent to solve such a complex, NP-hard scheduling problem is a challenging task. To enhance the training process, we employed the following strategies.

Action dimension reduction: The application of FlexTAS necessitates the determination of two actions: gating control, and if engaged, the subsequent waiting time. However, allowing an agent to learn both gating actions and waiting times simultaneously presents a significant challenge. Consequently, our methodology concentrates on training the agent on gating actions alone, while disregarding the waiting time during gating. Despite the gating control being active, the system still adheres to the no-wait model for calculating the current data frame's reservation slot. Notably, our initial iteration of FlexTAS included wait time in its action space, but this led to considerable difficulties during training and was ultimately abandoned.

Action masking: To accelerate model training and enhance performance, we further implement action masking at each step of the agent's decision-making process. Specifically, we mask invalid actions based on two criteria: a) if enabling gate control for the current stream at the current node would exceed the node's capacity for gate entries, and b) if the current node is the last hop of the stream and the accumulated delay has already surpassed the jitter requirements of the stream, disabling gate control at this point would only exacerbate the jitter, leading to unacceptable levels.

Curriculum learning: It is difficult for the agent to learn to schedule a complex problem from the beginning. This is akin to teaching calculus immediately after teaching addition, subtraction, multiplication, and division in the first lesson; it is challenging to train an effective agent model. Therefore, in our training process, we initially only let the agent learn to schedule a small portion of the streams ($\delta\%$) and gradually increase the number of streams with an ($\epsilon\%$) step size once the agent has learned how to solve the current small-scale problem (successfully passed k times).

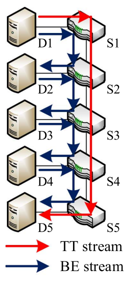


Fig. 11. Real testbed deployment. (a) Logical topology. (b) Physical deployment.

2) Generalization: To utilize the machine learning-trained model as a solution to the scheduling problem, it is essential to consider the model's generalization capability. We carefully designed the generalization of FlexTAS to ensure that the trained model can effectively adapt to various network topologies and streams.

Topology input: In the initial version of the design, we provided the GIN model with all nodes from the entire topology graph as input. However, we discovered that this approach was not only inefficient but also lacked generalization capability. Specifically, when the topology graph is large, inputting the entire graph includes a substantial amount of irrelevant information, introducing noise and resulting in unnecessary computation. Instead, we extract information only from the k-nearest neighbor nodes within a distance of 2 from the current node as input. The nodes at the remaining path of the stream are also included as input.

Environment generalization: Besides the topology input, we also designed the training environment to enhance the generalization of the model. Specifically, similar to the curriculum learning approach discussed in the previous section, we change the scheduling problem whenever the agent successfully learns to schedule the current problem. In detail, we generalize the stream input to ensure that the agent learns to schedule streams with different characteristics and QoS requirements. This iterative process aims to improve the model's adaptability and effectiveness across a wide range of scheduling scenarios.

IV. Implementation

In this section, we conduct experiments using actual hardware to verify the correctness of the delay analysis presented in the preceding section.

A. Multihop Experiments

We build a real testbed as shown in Fig. 11, where TSN switches are FPGA-based. Traffic are generated and analyzed by Spirent C50, a high-precision network testing platform. The TT stream, transmitted from `D1` to `D5`, is an isochronous stream that sends a 200-B frame each 200 μs. Four low-priority BE streams are set to interfere with the TT stream. For each test case, we set up 5 levels of interference: BE streams were transmitted at

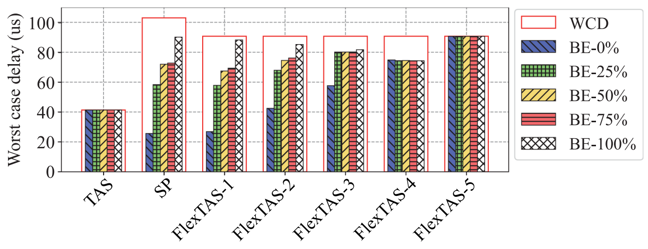


Fig. 12. Theoretical WCD (shown as the large red box), and the maximum delay measured in each case.

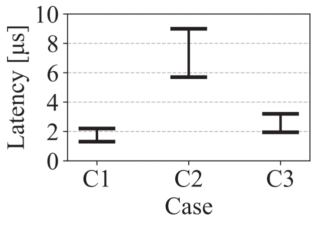


Fig. 13. Processing delay.

0, 25, 50, 75, and 100 percent of the line rate, respectively. The BE frame size was randomly chosen in [64B, 1518B].

Fig. 12 shows the end-to-end delay of the TT stream, where WCD represents the worst-case delay calculated by the theoretical analysis discussed in previous sections, and FlexTAS-k denotes our close-open gating model, with k indicating the specific switch where gating control is enabled. For example, FlexTAS-3 means that gating control is enabled at switch D3. If gating is enabled, t_4 is set to $t_3^{\max}$ to minimize the queuing delay. Otherwise, the switch simply keeps all the gates open.

All test cases exhibited actual delays smaller than the calculated WCD. For TAS, we employed the No-wait constraint model [11] to generate GCLs, which resulted in the smallest WCD. In contrast, the Legacy SP, lacking any gating mechanism, had the largest WCD. In the case of FlexTAS, the end-to-end delay is significantly affected by the location where gating control is implemented. On this 5-hop path, enabling gating later results in smaller jitter, as gating can eliminate the accumulated jitter from previous nodes. Specifically, enabling gating at the last hop (i.e., FlexTAS-5) achieves zero jitter.[5]

B. Single-Hop Experiments

Fig. 13 illustrates the processing delay of switches, corresponding to the interval from t_2 to t_3 in Fig. 2. Specifically, we tested three different versions of devices with varying port speeds. The processing delay for these devices ranged from 1 to 10 μs, with fluctuations between 1 and 5 μs. Consequently, the calculation of $t_3^{\min}$ and $t_3^{\max}$ needs to be determined based on the specific devices used.

Fig. 14 shows the single-hop jitter [t_{JT} in (2)] in the absence of gating. In this scenario, a TT frame may need to wait for a full MTU-sized frame to be forwarded, as discussed in

[5]We also conducted experiments with different TT frame sizes and periods, and the results were consistent with the findings presented here.

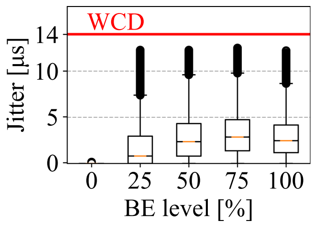


Fig. 14. Single-hop jitter.

Section III-B. WCD represents the calculated worst-case delay, which incorporates the transmission time of a maximum 1518-B BE frame, IFG, and clock synchronization variation. The results validate our analysis that the per-hop jitter remains bounded, as highlighted in the *takeaway* of Section II.

V. Evaluation

A. Experiment Settings

In this section, we present the results of our simulated experiments, which were conducted on a server equipped with an Intel i7-9700 CPU. We evaluate FlexTAS on 3 different kinds of random graph: random regular graph, Erdős–Rényi Graph, and Barabási–Albert graph, which represent common network topologies [13]. We utilize the Python NetworkX library to randomly generate these topologies, each comprising 20 nodes. The links between nodes are configured with a bandwidth of 100 Mb/s. We set the global macrotick and propagation delay at 1 μs. TT streams are generated as follows: the talker (source device) and the listener (target device) are selected randomly from all nodes, the frame size is selected randomly from 64 to 1518 B, and its period is selected randomly from {2 ms, 4 ms, 8 ms, 16 ms, 32 ms, 64 ms, 128 ms}. The end-to-end deadline of each stream is its period, and the jitter requirement is set to θ of its period, where θ is randomly selected from the set $\{0.1, 0.2, 0.5\}$. The switch processing delay is 1 μs to 6 μs. The GCL capacity is set to 256 by default. Unless otherwise specified, any scheduling results that exceed this capacity are deemed unsuccessful.

The timeout value for solving the problem is set to just 5 s. This is because, for such an NP-hard scheduling problem, spending several hours or even days attempting to find a solution is impractical. This is particularly true given the uncertainty of finding a feasible solution, as network resources are fixed and the capacity to admit streams is limited. In practical applications, such as in the production lines of a glass factory where there is a frequent need to switch production types, it is crucial to obtain a scheduling solution as swiftly as possible [13]. This approach enables more efficient and adaptable operations in dynamic industrial environments.

B. Comparative Algorithms.

We focus on the impact of gating on scheduling performance. The following typical gating strategies are used.

1) *AllGate:* This strategy utilizes the traditional TAS gating model, applying gating to all streams. It represents the majority of existing TAS-based solutions [5], [6], [11],

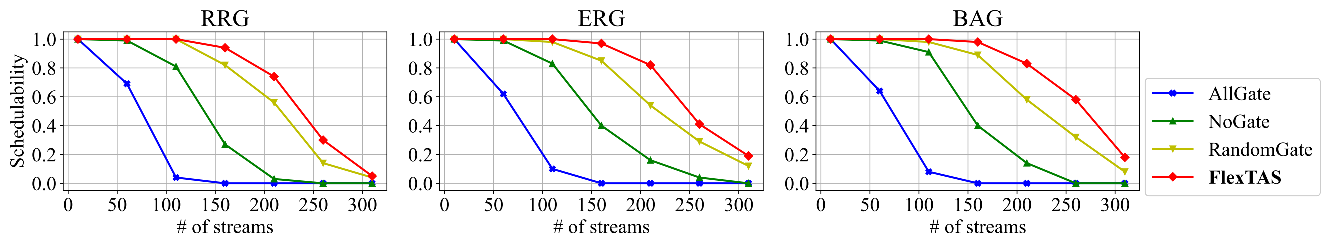


Fig. 15. Performance on different network topologies.

[13], which require a distinct mapping between GCL windows and frames.

2) *NoGate:* No gating is deployed. The isolation between TT streams is achieved by controlling the TT frame injection time, similar to the NGC method in [3].
3) *RandomGate:* This strategy uses the `close-open` gating model proposed in this study, but applies gating to streams randomly.
4) *FlexTAS:* This approach utilizes the DRL-based scheduling engine with the `close-open` gating model. A *single* pretrained agent is used across *ALL* experiments thanks to its generalization design.

As our study focuses on the impact of gating on scheduling performance, the choice of time slot scheduling strategies has a limited effect in this context, since the GCL capacity constraint is the primary limiting factor. Therefore, we employ the classical TimeTabling algorithm [11] for time-slot scheduling in the AllGate, NoGate, and RandomGate strategies.

C. Results

1) Overall Performance: We first evaluate the schedulability of the scheduling algorithms. For each experiment, a set of different inputs are generated and fetched to each scheduling algorithm. Finally, the percentage of success is measured, i.e., all streams are successfully scheduled.

Fig. 15 shows the scheduling performance of the algorithms across different network topologies, which show a similar trend. As the number of streams requiring scheduling increases, the performance of all algorithms gradually declines. Among these, the decline is most pronounced with the AllGate algorithm, due to its requirement to apply gating to all streams. This leads to a severe GCL expansion problem, swiftly surpassing the device's GCL capacity and resulting in scheduling failures. In contrast, the NoGate approach, which operates without gating, is not constrained by GCL limitations. However, it faces scheduling challenges due to accumulated jitter and high link utilization rates, which fail to meet isolation constraints and jitter constraints. The RandomGate serves as a compromise between the two, utilizing flexible gate control from this study to mitigate significant GCL expansion while moderately reducing accumulated jitter and link utilization. Consequently, it performs slightly better than the previous two. Lastly, our proposed algorithm, FlexTAS, benefits from a pretrained agent that intelligently determines whether and where to enable gate control for various TT streams at each node, resulting in enhanced performance.

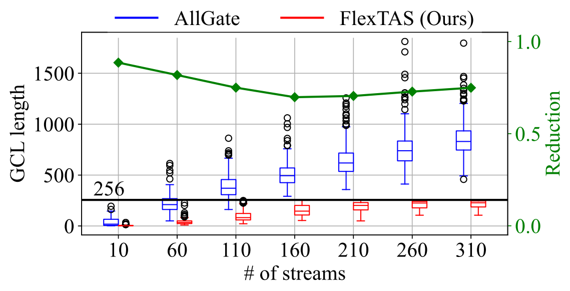


Fig. 16. Required GCL length for scheduling TT streams.

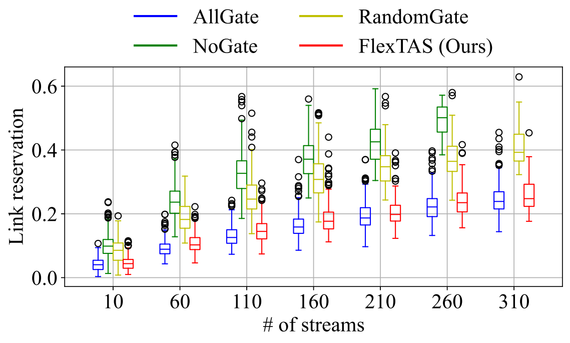


Fig. 17. Link reservation for scheduling TT streams.

2) GCL Requirements: Fig. 16 depicts the GCL lengths required for scheduling TT streams in our experiments across various network topologies. The AllGate strategy, which employs complete gating, leads to a significant GCL expansion problem, which is precisely the factor that limited its scheduling performance in the previous experiment. As the number of TT streams increases, the *GCL expansion* issue (see Section II) becomes more pronounced, with the required GCL length rapidly exceeding the capacity. In contrast, our proposed FlexTAS algorithm takes into account the device's GCL capacity during scheduling, consistently keeping the required number of GCL entries within the device's capabilities. The green line in the diagram demonstrates the percentage reduction in GCL entries achieved by FlexTAS compared to AllGate, showing a substantial reduction of 60% to 80%.

3) Link Reservation: Fig. 17 shows the link reservation for scheduling TT streams in our experiments. Lower link reservation values are desirable, as higher values suggest that more time slot resources are exclusively allocated to TT streams, making it more challenging to schedule new TT streams due to

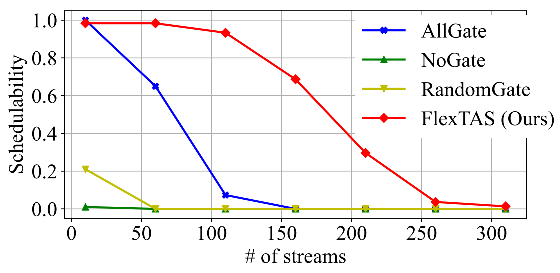


Fig. 18. Schedulability under strict jitter requirements.

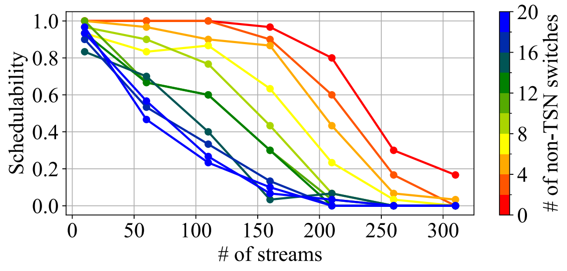


Fig. 19. FlexTAS under heterogeneous network conditions.

the stringent isolation constraints. The AllGate strategy, using complete gating, maintains the lowest link reservation, while the NoGate strategy significantly increases link reservation by 1× to 3×, since it introduce additional jitter at each hop, thereby increasing the required reserved time slot. FlexTAS, with its intelligent gating, controls link reservation levels close to AllGate, demonstrating efficient resource management and reliable TT stream scheduling.

4) Jitter Impact: We further tightened the jitter requirements for streams, including scenarios demanding zero jitter, to assess the performance of FlexTAS under extremely stringent conditions. Specifically, the jitter requirement percentage θ is randomly chosen from the set $\{0, 0.1, 0.2\}$. Fig. 18 illustrates the performance of various algorithms under these conditions. Both RandomGate and NoGate, lacking effective gating to mitigate jitter, are completely unstable in this scenario. In contrast, AllGate is insensitive to jitter variations, and its performance remains unchanged from less strict scenarios. Ultimately, our proposed algorithm, FlexTAS, continues to outperform other methods, with its agent effectively managing gate control to meet the stringent jitter demands of the streams.

5) Heterogeneous Network: In this experiment, a certain number of nodes are configured as non-TSN switches, which can only forward traffic based on SP. Fig. 19 shows the performance of FlexTAS as the number of non-TSN switches in the network increases. As the proportion of non-TSN switches grows, the scheduling performance of FlexTAS gradually declines. Eventually, when all switches in the network are non-TSN, the performance of FlexTAS degrades to that of the NoGate strategy.

VI. Related Works

GCL management: Fig. 20 illustrates a comparison between FlexTAS and other TSN mechanisms. Specifically, traditional TAS-based approaches [5], [6], [7] enable precise control over

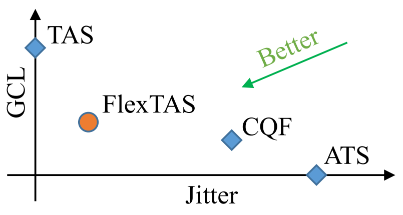


Fig. 20. Comparison with other TSN mechanisms.

TT frames, achieving zero jitter. However, they suffer from the GCL expansion issue, which limits scalability. On the other hand, more recent cyclic queuing and forwarding-based and asynchronous traffic shaping-based variants [7], [17], [18], [19] mitigate the GCL expansion problem. Nevertheless, their coarse-grained time slicing is often insufficient to accommodate the fine-grained and flexible scheduling requirements of TT streams. FlexTAS strikes a balance between precision and configurability, offering a more adaptive and scalable solution for industrial networks.

DRL for TSN: Recent years have seen rapid advancements in DRL, with its applications in TSN also being demonstrated [7], [13], [14]. For example, DeepScheduler [13] trains an agent to learn the dependencies between TT streams, while Yu et al. [7] aim to solve the joint routing and scheduling problem. However, these studies often overlook the management of GCL and lack a generalized design that can be adapted to different network configurations and demands. FlexTAS tailors the DRL algorithm for a more generalized application, thus advancing the application of DRL in TSN settings.

Heterogeneous networks: [20] specifically tackles the scheduling problem in heterogeneous networks where end-systems are unsynchronized, a critical consideration for practical network applications. Our study focuses on the diversity of switches within the network, particularly the coexistence of legacy switches with TSN-capable devices, which is orthogonal to their work, exploring a different facet of network heterogeneity.

VII. Conclusion

In this article, we introduced FlexTAS, a practical solution for TSN, which comprises a novel gating model and a DRL-based scheduling algorithm. FlexTAS reduces GCL requirements while ensuring QoS for TT streams. Its unique gating approach and general DRL-based scheduler address the challenges of joint scheduling and gating problem, making it an effective tool for transitioning to deterministic networks in heterogeneous environments.

References

[1] "Use cases IEC/IEEE 60802 V1.3," 2018. [Online]. Available: https://www.ieee802.org/1/files/public/docs2018/60802-industrial-use-cases-0918-v13.pdf

[2] R. S. Oliver, S. S. Craciunas, and W. Steiner, "IEEE 802.1Wbv gate control list synthesis using array theory encoding," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp.*, 2018, pp. 13–24.

[3] X. Jin et al., "Real-time scheduling of massive data in time sensitive networks with a limited number of schedule entries," *IEEE Access*, vol. 8, pp. 6751–6767, 2020.

[4] "Hirschmann user manual industrial hivision release 8.6." Accessed: Nov. 2024. [Online]. Available: https://www.doc.hirschmann.com/pdf/UM_IndHiVision_08600_en.pdf

[5] S. S. Craciunas, R. Serna, O. Martin, and C. W. Steiner, "Scheduling real-time communication in IEEE 802.1Qbv time sensitive networks," in *Proc. 24th Int. Conf. Real-Time Netw. Syst.*, 2016, pp. 183–192.

[6] J. Lin et al., "Rethinking the use of network cycle in time-sensitive networking (TSN) flow scheduling," in *Proc. IEEE/ACM 30th Int. Symp. Qual. Serv.*, 2022, pp. 1–11.

[7] H. Yu, T. Taleb, and J. Zhang, "Deep reinforcement learning based deterministic routing and scheduling for mixed-criticality flows," *IEEE Trans. Ind. Informat.*, vol. 19, no. 8, pp. 8806–8816, Aug. 2023.

[8] *IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks—Amendment 25: Enhancements for Scheduled Traffic*, IEEE Std 802.1Qbv-2015, 2016.

[9] *IEEE Standard for Local and Metropolitan Area Networks—Timing and Synchronization for Time-Sensitive Applications*, IEEE Std 802.1AS-2020, 2020.

[10] *IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks—Amendment 26: Frame Preemption*, IEEE Std 802.1Qbu-2016, 2016.

[11] F. Dürr and N. G. Nayak, "No-wait packet scheduling for IEEE time-sensitive networks (TSN)," in *Proc. 24th Int. Conf. Real-Time Netw. Syst.*, Oct. 2016, pp. 203–212.

[12] X. Jiang, Y. Zhang, W. Fu, X. Yang, Y. Sun, and Z. Sun, "TASP: Enabling time-triggered task scheduling in TSN-based mixed-criticality systems," in *Proc. IEEE/ACM 30th Int. Symp. Qual. Serv.*, 2022, pp. 1–11.

[13] X. He, X. Zhuge, F. Dang, W. Xu, and Z. Yang, "DeepScheduler: Enabling flow-aware scheduling in time-sensitive networking," in *Proc. IEEE Conf. Comput. Commun.*, 2023, pp. 1–10.

[14] X. Wang, H. Yao, T. Mai, S. Guo, and Y. Liu, "Reinforcement learning-based particle swarm optimization for end-to-end traffic scheduling in TSN-5G networks," *IEEE/ACM Trans. Netw.*, vol. 31, no. 6, pp. 3254–3268, Dec. 2023.

[15] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–17.

[16] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.

[17] Y. Zhang, Q. Xu, L. Xu, C. Chen, and X. Guan, "Efficient flow scheduling for industrial time-sensitive networking: A divisibility theory-based method," *IEEE Trans. Ind. Informat.*, vol. 18, no. 12, pp. 9312–9323, Dec. 2022.

[18] L. Zhao, P. Pop, and S. Steinhorst, "Quantitative performance comparison of various traffic shapers in time-sensitive networking," *IEEE Trans. Netw. Service Manage.*, vol. 19, no. 3, pp. 2899–2928, Sep. 2022.

[19] D. Yang, Z. Cheng, W. Zhang, H. Zhang, and X. Shen, "Burst-aware time-triggered flow scheduling with enhanced multi-CQF in time-sensitive networks," *IEEE/ACM Trans. Netw.*, vol. 31, no. 6, pp. 2809–2824, Dec. 2023.

[20] M. Barzegaran, N. Reusch, L. Zhao, S. S. Craciunas, and P. Pop, "Real-time traffic guarantees in heterogeneous time-sensitive networks," in *Proc. 30th Int. Conf. Real-Time Netw. Syst.*, 2022, pp. 46–57.