A Survey and Experimental Study of Real-Time Scheduling Methods for 802.1Qbv TSN Networks

CHUANYU XUE, University of Connecticut, Storrs, United States
TIANYU ZHANG, University of Iowa, Iowa City, United States
YUANBIN ZHOU, Singapore University of Technology and Design, Singapore, Singapore
MARK NIXON, R&D, Emerson Automation Solutions, Round Rock, United States
ANDREW LOVELESS, NASA Johnson Space Center, Houston, United States
SONG HAN, University of Connecticut, Storrs, United States

Time-sensitive networking (TSN) has been recognized as one of the key enabling technologies for Industry 4.0 and has been deployed in many mission- and safety-critical applications e.g., industry automation, automotive and aerospace systems. Given the stringent real-time requirements of these applications, the Time-Aware Shaper (TAS) draws special attention among TSN's many traffic shapers due to its ability to achieve deterministic timing guarantees. Many scheduling methods for TAS shapers have been recently developed that claim to improve system schedulability. However, these scheduling methods have not yet been thoroughly evaluated, especially through experimental comparisons, to provide a systematical understanding of their performance using different evaluation metrics in diverse application scenarios. In this article, we fill this gap by presenting a systematic review and experimental study on existing TAS-based scheduling methods for TSN. We first review and categorize the system models employed in these works along with the specific problems they aim to solve, and outline the fundamental and additional considerations in the designs of TAS-based scheduling methods. We then perform an extensive evaluation on 17 representative solutions using both high-fidelity simulations and a real-life 16-node TSN testbed, and comparing their performance in terms of schedulability, scalability, and schedule quality. Through these experimental studies, we identify the limitations of individual scheduling methods and highlight important findings. We also summarize the open issues and future research directions in this area. We expect this work will provide foundational knowledge and performance benchmarks for future studies on real-time TSN scheduling and beyond.

CCS Concepts: • **General and reference → Surveys and overviews**; • **Networks → Network design and planning algorithms**; **Network performance analysis;**

Additional Key Words and Phrases: Time-sensitive networking, real-time scheduling, time-aware shaper, experimental study

The work is supported in part by the NSF Grant CNS-1932480, CNS-2008463, CCF-2028875, CNS-1925706, and the NASA STRI Resilient Extraterrestrial Habitats Institute (RETHi) under grant number 80NSSC19K1076.
Authors' Contact Information: Chuanyu Xue, University of Connecticut, Storrs, Connecticut, United States; e-mail: chuanyu.xue@uconn.edu; Tianyu Zhang, University of Iowa, Iowa City, United States; e-mail: tianyu-zhang@uiowa.edu; Yuanbin Zhou, Singapore University of Technology and Design, Singapore, Singapore; e-mail: yuanbin_zhou@sutd.edu.sg; Mark Nixon, R&D, Emerson Automation Solutions, Round Rock, Texas, United States; e-mail: mark.nixon@emerson.com; Andrew Loveless, NASA Johnson Space Center, Houston, Texas, United States; e-mail: andrew.loveless@nasa.gov; Song Han, University of Connecticut, Storrs, Connecticut, United States; e-mail: song.han@uconn.edu.

ACM 0360-0300/2025/09-ART46
https://doi.org/10.1145/3736715

ACM Reference Format:
Chuanyu Xue, Tianyu Zhang, Yuanbin Zhou, Mark Nixon, Andrew Loveless, and Song Han. 2025. A Survey and Experimental Study of Real-Time Scheduling Methods for 802.1Qbv TSN Networks. *ACM Comput. Surv.* 58, 2, Article 46 (September 2025), 37 pages. https://doi.org/10.1145/3736715

1 Introduction

Time-sensitive networking (**TSN**), as an enhancement of Ethernet, has quickly become the **local area network** (**LAN**) technology of choice to enable the co-existence of **information technology** (**IT**) and **operation technology** (**OT**) in the **industrial Internet-of-Things** (**IIoT**) paradigm. TSN aims to provide deterministic Layer-2 communications, which are highly desirable for many real-time industrial applications, such as process automation and factory automation [51, 88, 98]. To enable such communication capabilities, the TSN **Task Group** (**TG**) has developed a suite of traffic shapers in the TSN standards, including the **Credit-Based Shaper** (**CBS**) [2], **Asynchronous Traffic Shaper** (**ATS**) [5], and **Time-Aware Shaper** (**TAS**) [3], to handle different traffic types and satisfy communication requirements at different levels. In terms of providing strict real-time performance guarantees, TAS stands out by leveraging network-wide synchronization and time-triggered scheduling mechanisms, making it a critical technology to support deterministic traffic.

Although the scheduling mechanism of TAS is clearly defined in the IEEE 802.1Qbv standard, the configuration of TAS has no clear-cut best practice. Specifically, the fundamental question for TAS-based real-time scheduling in TSN is how to generate a network-wide schedule to guarantee the timing requirements of all **time-triggered** (**TT**) traffic [91]. Given that applications that employ TSN as the communication fabric can be diverse from different perspectives (e.g., traffic patterns, topology, deployment environments, and QoS requirements), the specific scheduling problem to be studied may vary significantly. This results in a large amount of effort from both researchers and practitioners to study various system models and develop corresponding algorithms to address specific TAS-based TSN scheduling problem. These studies considerably enrich the literature, paving the way for improving TSN network performance.

There have been several recent survey works on real-time scheduling in TSN networks (e.g., [24, 39, 63, 68, 86, 91, 103]). These studies provided a broad overview of the TSN standards, identified the limitations of existing TSN scheduling methods, and outlined future research directions. In addition, [67, 91] provided comparisons among various TSN scheduling approaches, with [91] primarily focusing on TAS-based studies and [67] extending the comparisons to all TSN shapers. However, all the aforementioned works suffer from the following two significant limitations. First, they do not provide a thorough model-based categorization for the TSN scheduling methods considering the used network models, traffic models, and scheduling models. Second, for the few existing works conducting comparisons of TSN scheduling methods, their comparisons are all conceptual in nature, which are far from sufficient for determining the effectiveness of individual methods under diverse scenarios.

To address the above limitations, this article summarizes the network models, traffic models, and scheduling models used in the literature for real-time scheduling in TSN. Based on the summarized models, we categorize 17 representative TAS-based scheduling methods proposed since 2016 (i.e., [10, 16, 22, 27, 29, 30, 41, 46, 47, 69, 70, 83, 85, 95, 97, 110]). To perform realistic experimental comparisons among these methods, we establish an 8-bridge TSN testbed to obtain quantitative measurement results of several key parameters commonly used in TSN models (e.g., propagation delay, processing delay, and synchronization error). Relying on the TSN testbed, we further conduct performance validation for all the scheduling methods to ensure the consis-

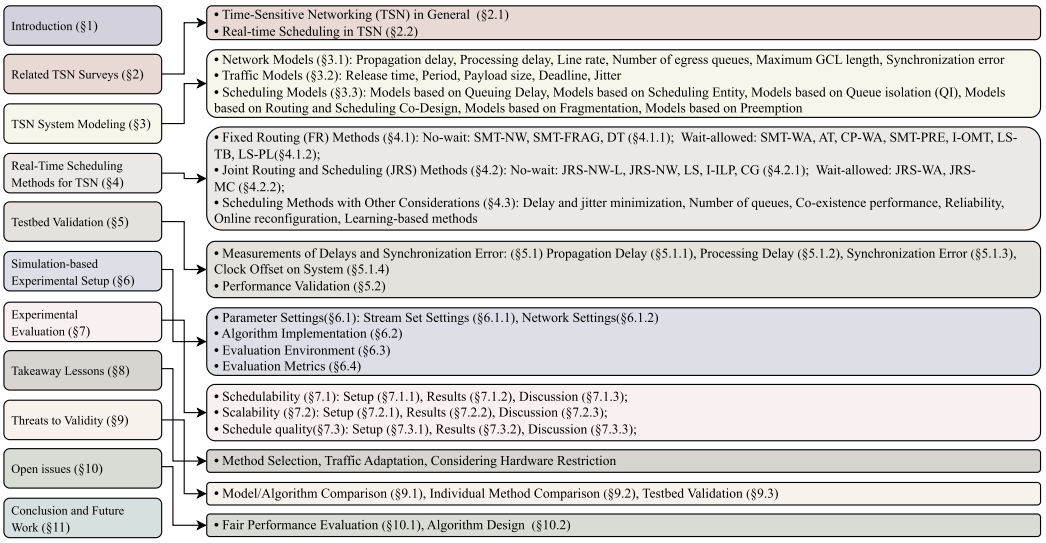


Fig. 1. High-level outline of the article.

tency between testbed results and analytic results derived from simulations. Based on all these preliminary outcomes, we perform extensive experimental studies for the 17 TSN scheduling methods under various stream sets and network settings covering a broad range of industrial application scenarios. Benefiting from our model-based categorization, we are able to perform experimental comparisons not only among individual scheduling methods but also across different system models.

Based on these experimental results, we are able to highlight a set of interesting observations. In general, our study shows that there is no one-size-fits-all solution that can achieve dominating performance in all scenarios while individual scheduling method/model may demonstrate superiority under certain setting(s). We also demonstrate that diverse experiment settings complicate the fair evaluation of scheduling methods without introducing bias. This makes conclusions from previous studies only valid under specific settings. We expect that our findings will help the community better understand the benefits and drawbacks of existing TSN scheduling methods and provide valuable insights for the development of future TSN scheduling methods. In summary, this work makes the following contributions, and the high-level outline of the article is summarized in Figure 1.

(1) We provide a comprehensive, up-to-date review of various TAS-based TSN system models and categorize 17 representative TAS-based TSN scheduling methods accordingly.
(2) We establish a 16-node real-world TSN testbed and perform quantitative parameter measurement and performance validation for the studied TSN scheduling methods.
(3) We perform extensive experimental evaluations on the 17 TSN scheduling methods under a wide range of industrial scenarios.
(4) We summarize the findings obtained from the evaluation and provide takeaway lessons for future research and development on TSN real-time scheduling method design and beyond.
(5) We provide an open-source repository that not only ensures the reproducibility of our results but also offers a tool for the community to extend the experimental research on TSN scheduling.

2 Related Surveys on TSN

2.1 Time-Sensitive Networking (TSN) in General

Existing survey articles have broadly introduced the fundamental concepts, protocols, and applications of TSN. Refs. [59, 68, 86] provided a comprehensive overview of TSN standards, usages, and related research, serving as the foundation for subsequent studies. Refs. [13, 31, 103] focused on the applications of TSN's applications in industrial automation and distributed systems, emphasizing on how TSN is used to ensure real-time communications in those application scenarios. Beyond the above general overviews, some works studied specific topics in TSN. For example, [48, 81, 82, 89] discussed the integration of TSN in 5G networks, identifying open challenges and research opportunities. Adame et al. [7] reviewed TSN's capabilities to support low latency and ultra-reliable services and discussed how to implement those functions in WiFi-7. Peng et al. [76] provided a comprehensive review of in-vehicle TSN, highlighting its role in meeting the demands of autonomous driving and connected vehicle applications. Ergenc et al. [28] reviewed TSN security, detailing potential vulnerabilities and mitigation strategies. Kehrer et al. [50] discussed TSN fault-tolerance, focusing on improving the reliability of these networks. Silva et al. [87] investigated **Software-Defined Networking** (**SDN**)-enabled TSN, highlighting its potential in industrial applications given the programmability of SDN.

2.2 Real-Time Scheduling in TSN

Real-time scheduling is one of the most important topics in TSN since it enables the network to meet stringent real-time requirements. Several surveys have explored TSN scheduling, focusing on individual TSN traffic shapers. Chahed et al. [18] provided an overview of TSN scheduling, covering key topics such as traffic types, shapers, scheduling approaches, problem formulations, and performance metrics. Zhao et al. [107] focused on the performance comparison among various traffic shapers, offering a comprehensive review and quantitative comparison of their performance through experiments, highlighting tradeoffs in latency, jitter, and throughput. Maile et al. [58] dived into network calculus-based response analysis used in CBS, Priority Shaper, and TAS, providing a unified perspective on their dependencies and common assumptions. Wang et al. [99] conducted a systematic literature review, identifying research trends and emphasizing on future challenges in TSN schedulability analysis. Finally, Nasrallah et al. [67] concentrated on CQF-based scheduling, detailing related protocols and approaches.

Among the various TSN traffic shapers, TAS is commonly used in various applications. Several surveys have focused on TAS scheduling, providing a comprehensive overview of the scheduling mechanisms and individual scheduling methods. Hellmanns et al. [39] discussed different TAS scheduling approaches, classifying them into stream-based, class-based, and frame preemption-enabled methods. It proposed a method to formulate and solve the class-based TAS scheduling problem, and discussed the selection of the three approaches. Kist et al. [52] conceptually compared two TAS-based scheduling approaches: one leverages the hierarchical structure of factory automation networks to simplify the creation of schedules; the other aims to improve the performance of BE traffic. Stüber et al. [91] provided a comprehensive survey of TSN scheduling, with a focus on TAS-based scheduling. It classified the TAS-based scheduling methods into joint-routing, fixed-routing, and their subclasses. It also provided suggestions and pointed out the open issues in TSN scheduling. Nevertheless, these existing works are constrained by two notable limitations. Firstly, a comprehensive categorization of TSN scheduling methods based on their underlying network, traffic, and scheduling models is missing. Secondly, the comparisons drawn between TSN scheduling methods in the literature are largely conceptual, failing to provide sufficient insight into their practical effectiveness under varied conditions.

Recently, two surveys have attempted to bridge the gap between conceptual comparisons and experimental studies in TAS scheduling. Schweissguth et al. [84] benchmarked several TAS scheduling approaches through experimental comparisons. However, their study is limited to six configurations (e.g., cut-through capability, differing optimization objectives) and does not cover individual scheduling methods. Stüber et al. [90] provided an experimental evaluation of TAS scheduling algorithms by comparing their schedule quality and runtime under various network conditions. However, this study does not consider several critical models and scheduling methods, including the preemption model, fragmentation model, and **constraint programming** (**CP**) methods, which have a significant impact on the performance of TAS-based scheduling. In addition, several modeling elements, e.g., queuing delay, scheduling entities, and **queue isolation** (**QI**), have also been overlooked. Our work, instead, takes a more comprehensive approach, incorporating these potentially influential factors into the evaluation. We also construct a real-world testbed to validate our findings, providing a robust foundation for generating a series of insights, open issues, and takeaway lessons. In addition, we provide an open-source repository that not only ensures the reproducibility of our results but also offers a comprehensive tool to facilitate the community in designing and evaluating new TSN scheduling methods and beyond.

3 TSN System Modeling

This section presents an overview of the network models, traffic models, and scheduling models for real-time scheduling in TSN. It provides the foundation for the categorization of TAS-based scheduling methods in Section 4.

3.1 Network Models

A TSN network consists of two types of devices: bridges and **end stations** (**ESs**). A bridge can forward Ethernet frames for one or multiple TSN streams according to a schedule constructed based on the IEEE 802.1Q standard [45]. Each ES can be a *talker*, acting as the source of TSN stream(s), a *listener*, acting as the destination of TSN stream(s), or both.

Each full-duplex physical link connecting two TSN devices (either bridge or ES) is modeled as two directed logical links. Each logical link is associated with the following attributes:

- **Propagation delay** refers to the time duration of a signal transmitting on the physical link (i.e., Ethernet cable). This delay is solely dependent on the length of the cable and the type of physical media used.
- **Processing delay** refers to the time a frame takes from the moment it reaches the ingress port to the moment it is fully stored in the egress queue. The delay is determined by the bridge's processing capability. In the literature, it is typically modeled as a constant or a boundable value.
- **Line rate** refers to the data rate that frames can be transmitted over a logical link within a given time interval. There are three line rates commonly employed for TSN (10/100/1000 Mbps), while higher-speed TSN bridges with 10/100 Gbps line rates have also been developed recently.
- **Number of egress queues** refers to the available egress queues dedicated to TT traffic. The IEEE 802.1Q standard sets a max of eight queues per egress port for a TSN bridge [45].
- **Maximum GCL length** indicates the maximum allowed number of entries in the GCL of a logical link, and this is determined by the specific bridge implementation (e.g., typically between 8 and 1024 [69]).
- **Synchronization error** is typically defined as the maximum time offset between any two non-faulty logical clocks in the network and is shared across all nodes and links. However, the recent IEEE 802.1AS-rev standard introduces a more precise synchronization error,

enabling individual error for each node based on specific network configurations, roles of nodes, and hop distance to the grandmaster [6].

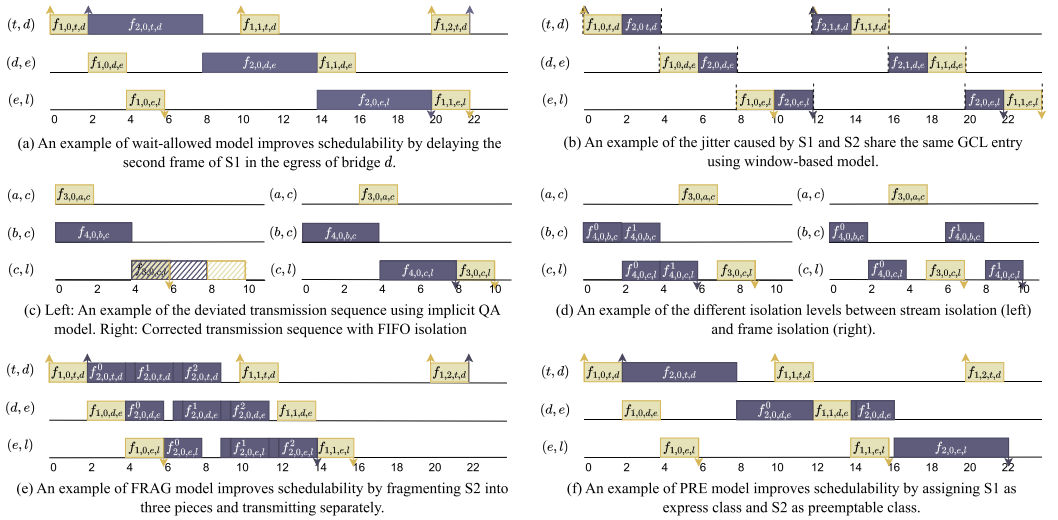
3.2 Traffic Models

In TSN networks, a traffic stream refers to a unidirectional flow of data transmitted from a single talker to one or multiple listeners, passing through bridges over multiple logical links. In addition to TT traffic, TSN can also accommodate lower-criticality asynchronous traffic, such as standard Ethernet and audio and video (AVB) traffic. Driven by the determinism requirements posed by real-time industrial applications, the literature mainly focuses on the TT traffic scheduling problem, which is also the emphasis of this article.

Each TSN stream can be characterized by five parameters: release time, period, payload size, deadline, and jitter. Each of these parameters can be modeled individually in order to capture the specific characteristics of the targeted traffic type, based on the application scenario under study.

- **Release time:** The release time of a stream is defined as the time when its first frame is dispatched on the physical link by the talker. Depending on whether the talker can determine the release time of its stream(s), the traffic model can be classified into *fully schedulable* traffic and *partially schedulable* traffic. The fully schedulable traffic model allows the scheduler to configure the release time of each stream and thus yield higher schedulability, while the latter assumes that the release time of each stream is given by the application.
- **Period:** The period of a stream defines the inter-arrival pattern of its frames. It can be classified into *strictly periodic* model and *non-strictly periodic* model based on the determinism of their arrival times. In a strictly periodic model, each frame must follow the same release offset. By contrast, in the non-strictly periodic model, frames from the same stream can be released with varied but bounded offsets.
- **Payload size:** Payload size refers to the size of the application data to be transmitted within a stream. When the payload size is larger than 1500 bytes, a stream may contain multiple frames in the same period.[1] There are two main strategies for transmitting multiple frames from the same stream in the same period. The first approach schedules each frame individually while preserving the frame order by introducing additional constraints. Alternatively, the second approach schedules all frames from the same stream successively within an extended time duration on each link.
- **Deadline:** The deadline of a stream defines the time by which the released frame(s) must be received by the listener, such that the release time from the talker plus the end-to-end delay does not exceed this deadline. The deadline of a stream can be modeled as *implicit* (equal to the period), *constrained* (less than the period), or *arbitrary*, according to the application scenario. Note that, under the arbitrary deadline model, a frame released in its period can be scheduled into the next period interval and such cases should be carefully handled in the scheduling to avoid potential conflicts.
- **Jitter:** The jitter captures the variation of end-to-end stream delay (i.e., the difference between the minimum and maximum delays of frames transmitted from the same stream), following the definition in IEC/IEEE 60802 TSN Profile [26]. The *zero-jitter* model enforces fully deterministic traffic behavior for each frame, whereas the *jitter-allowed* model permits limited conflicts from other traffic on delay, subject to a user-defined jitter upper bound. Figure 2(a) illustrates a jitter-allowed scheduling example for stream S1, with a delay of 6 for the first instance and 12 for the second. Figure 2(b) demonstrates how jitter arises from

[1]According to the IEEE 802.1Q standard, the maximum frame size is 1522 bytes, including a frame payload and a 22-byte frame header containing the VLAN Tag, Ethernet header, and Frame Check Sequence.



(a) An example of wait-allowed model improves schedulability by delaying the second frame of S1 in the egress of bridge d.

(b) An example of the jitter caused by S1 and S2 share the same GCL entry using window-based model.

(c) Left: An example of the deviated transmission sequence using implicit QA model. Right: Corrected transmission sequence with FIFO isolation

(d) An example of the different isolation levels between stream isolation (left) and frame isolation (right).

(e) An example of FRAG model improves schedulability by fragmenting S2 into three pieces and transmitting separately.

(f) An example of PRE model improves schedulability by assigning S1 as express class and S2 as preemptable class.

Fig. 2. An illustration of various scheduling models. $f_{i,k,a,b}$ indicates the kth frame of stream-i scheduled on link (a, b), $f^r_{i,k,a,b}$ indicates the rth fragment if fragmentation or preemption is applied. To simplify the notations, we use t to denote the talker and l to denote the listener associated with each path. The up arrow indicates the frame release time at the talker, and the down arrow indicates the frame reception time at the listener. The solid area indicates the actual transmission pattern and the shallow area indicates the traffic planning result. The dash line indicates the border of a GCL entry.

interference when both S1 and S2 share a GCL entry along the path, resulting in a bounded delay between 10 and 12.

3.3 Scheduling Models

Based on the network and traffic models described above, the TAS-based real-time scheduling problem in TSN aims to construct feasible communication schedules to satisfy the temporal constraints imposed by the streams deployed in the TSN network. For this aim, a range of scheduling models have been proposed to define the constraints made on end-to-end delay, jitter, queuing assignment, routing path, fragmentation, and preemption. We summarize these scheduling models below and categorize them according to their unique features.

- **Models based on Queuing Delay:** Queuing delay (i.e., the amount of time that a frame spends waiting in the egress queue) is decided by the constructed communication schedule and has the most impact on the end-to-end (e2e) delay of a stream. Based on the assumptions on queuing delay, the scheduling models can be classified into *no-wait* and *wait-allowed* models. The no-wait model requires consecutive frame transmissions along the path, i.e., frames should be forwarded without queuing delay. As a result, the no-wait scheduling model focuses on planning the release times of the streams on the talkers, which typically results in reduced scheduling effort but a smaller solution space. On the other hand, the wait-allowed model is more general. It allows frames to be stored in the queue and forwarded at a later time on bridges, therefore having a larger solution space. For example, Figure 2(a) follows the wait-allowed model as the second frame from stream S1 waits 2 time units on link (d, e) and 4 time units on link (e, l). The no-wait model cannot find a feasible solution in this case.
- **Models based on Scheduling Entity:** Depending on the objects used for the allocation of GCL entry [39], the scheduling models can be classified into *frame-based* model, *window-based* model,

and *class-based* model. The frame-based model schedules the transmission time of each frame in a per-stream fashion and directly maps it to a dedicated GCL entry. By constraining that no overlap exists for any two frames' transmission time, the frame-based model guarantees that there is no interference between any two streams. The window-based model partitions frames into different groups and jointly schedules the transmission time of the frames in each group. As each GCL entry is shared by a group of frames, the transmission order of each frame can be interfered with by other frames within the same group, resulting in jitter. For example, in Figure 2(b), streams S1 and S2 are assigned to the same egress queue and share the same GCL entry according to a window-based model. Their transmission orders in two consecutive periods are different within the window, which causes jitter. In general, the window-based model can be further classified as assigned, partially assigned or non-assigned based on different frame-to-window allocations [11]. The class-based model allocates resources for each traffic class and guarantees the same deadline and jitter requirements for a whole class. Each traffic class is mapped to a dedicated egress queue. Because the class-based model is mainly employed for asynchronous traffic classes (e.g., AVB and BE traffic), it is out of the scope and not discussed in this article.

- **Models based on Queue isolation (QI):** Scheduling models based on queue assignment can be categorized into *unrestricted QI* models and *explicit QI* models. Unrestricted QI models, derived from TTEthernet scheduling, assume a global schedule that defines the temporal behavior of all frames without considering QI [21]. However, applying this to TSN may violate FIFO properties, leading to deviations from the intended transmission schedule. For example, in Figure 2(c), the unrestricted QI model allows S4 to be scheduled before S3 on link (c, l) without conflict. However, if both streams share a queue, S3 may occupy S4's slot at runtime, causing S4 to be delayed and miss its deadline. This inconsistency occurs only under the wait-allowed model, as the no-wait model forwards frames immediately.

  To avoid such schedule inconsistency, explicit QI models isolate streams into different queues by jointly computing the queue assignment along with the schedule. For this aim, several isolation constraints are proposed which can be categorized into three levels: *FIFO*, *frame-based*, and *stream-based* isolation. The basic idea of FIFO isolation is to prevent reordering the forwarding sequence of frames when they are assigned to the same queue. For instance, the schedule on the right side of Figure 2(c) enforces stream S4 to arrive earlier than S3, so that the forwarding order matches the predefined schedule, even if they are within the same queue.

  Frame-based isolation and stream-based isolation are comparatively more realistic as they take into account frame loss, unbounded processing jitter, and interleaving caused by fragmentation when the payload size is larger than the MTU. For example, as shown on the right of Figure 2(c), under the FIFO isolation model, stream S3 will forward earlier than expected by using the slot allocated for S4 on link (c, l) if the frame of S4 is lost. Frame-based isolation ensures that at one time, only one frame can exist in the queue so that it is not interfered by other frames' fault conditions. For example, as illustrated on the right side of Figure 2(d), the frame of stream S3 can only arrive at link (c, l) after the first fragment of stream S4 is dispatched. The second fragment of S4 can only arrive at link (c, l) after the frame of S3 is dispatched. Compared to frame-based isolation, stream-based isolation is more stringent. It requires that the frame of the current stream can only be enqueued after all frames of the previous stream(s) have been fully dispatched, as shown on the left side of Figure 2(d). The frame of S3 must wait until all the fragments of S4 are dispatched.

- **Models based on Routing and Scheduling Co-Design:** Depending on whether the routing path of each stream is given or needs to be determined, the scheduling models can be categorized as ***fixed routing*** (**FR**) model and ***joint routing and scheduling*** (**JRS**) model. The JRS model allows the co-design of route selection and schedule construction, while the FR model focuses

on schedule construction only, assuming that the routes are pre-determined. By optimizing the routing and scheduling decisions in a joint fashion, the JRS model could offer better resource utilization and schedulability in general when compared to the FR model. However, the JRS model may also incur much higher computational overheads and may not be able to find feasible solutions if the computing resource is constrained.

• **Models based on Fragmentation:** In the network layer, fragmentation occurs when a packet is split into smaller fragments to fit the **maximum transmission unit** (**MTU**) size of the network. However, default fragmentation may result in high latency due to the large size. To address this issue, the *fragmentation* (FRAG) model is proposed to determine the number and size of fragments along with the schedule construction. The FRAG model, to some extent, can improve schedulability, especially in cases when the deadline is exceeded due to the large frame size. For example, consider stream S2 in Figure 2(e) with a transmission duration of 6 time units, the minimum end-to-end delay for an S2 frame to travel through the three hops is 18 time units. However, the FRAG model fragments the original frame into three small fragments and starts forwarding immediately after the first fragment is received on each hop. In this case, if the deadline is set to 12, S2 can only be scheduled for transmission under the FRAG model. It is worth noting that fragmentation comes with extra header overhead, which may negatively impact link utilization.

• **Models based on Preemption:** The IEEE 802.1Qbu standard defines frame preemption allowing an express frame to interrupt the transmission of a preemptable frame, and subsequently resume the preempted frame at the earliest available opportunity [44]. In the *preemption* (PRE) model, frames may be assigned with varied preemption classes at different hops, with only express frames being able to interrupt preemptable frames. The preemptable frame can thus be broken into two or more fragments. For instance, in Figure 2(f), the frame of stream S2 on link (d, e) is designated as a preemptable class and is consequently preempted by the frame of S1, classified as an express class, at time 12 during the transmission. Following the completion of stream S1's transmission, the second fragment of S2 resumes its transmission. It is worth noting that the *preemption* (PRE) model does not reduce the transmission delay compared to FRAG, as frames can only be forwarded after all fragments are fully received and reassembled. For example, in Figure 2(f), link (e, l) cannot forward at time 12 when the first fragment is received but has to wait until time 16 when both fragments have been received. Nonetheless, if preemption is disabled, S1 fails to meet its deadline, resulting in an unschedulable stream set. In addition, as fragmentation only occurs when necessary, the PRE model offers better bandwidth conservation when compared to the FRAG model due to the reduced number of generated headers. For example, there is only one additional header created for the second fragment of the frame of S2 along its path in Figure 2(f). By contrast, a total number of six headers are generated under the FRAG model in Figure 2(e).

4 Real-Time Scheduling Methods for TSN

Based on the system models discussed above, we now delve into a detailed review of 17 TAS-based real-time scheduling methods published since 2016. Following the standard protocol of systematic review outlined in [53], we select these methods based on two main criteria. **(1) Breadth:** to give a comprehensive review and experimental study, we aim to include as diverse a set of models and algorithms as possible; **(2) Relevance:** to focus on real-time scheduling of TT traffic in TSN, approaches aiming at enhancing AVB or BE traffic in mixed-criticality scenarios or improving reliability are not included. We also exclude learning-based methods that cannot provide deterministic results.

In the following, we categorize the 17 scheduling methods and highlight their specific optimization objectives in addition to generating feasible schedules. We first classify all the methods into

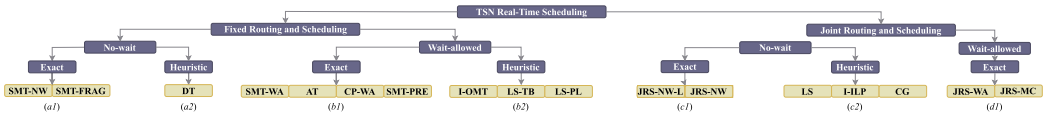


Fig. 3. Classification of the TSN real-time scheduling methods based on the employed system models. The first layer categorizes routing models, the second layer categorizes queuing delay models, and the final layer categorizes algorithms. We tag each leaf category by {*a1, a2, b1, b2, c1, c2, d1*} for cross-referencing.

two categories, FR-based methods or JRS-based methods, depending on whether the routing path of each stream is given or to be determined. The methods in each category are further divided into no-wait-based methods and wait-allowed-based methods according to their employed delay models. Finally, each method is classified as either an exact solution or a heuristic solution based on whether the method can yield an optimal schedule or not.[2] Figure 3 summarizes the categorization.

4.1 Fixed Routing (FR) Methods

FR-based methods assume that the routing paths of individual streams are pre-determined and provided as input for the scheduling algorithms. Based on the IEEE 802.1Qca standard, the default routing paths generated by the Shortest Path Bridging Protocol ensure that all streams are routed along their shortest paths between the talkers and listeners [4].

4.1.1 No-wait. The no-wait model requires all frames to be forwarded along their routing paths without any queuing delay. The following methods employ a combination of the FR model and no-wait model to minimize the e2e latency.

- **SMT-NW (*a1*):** Durr et al.[27] addressed the crucial problem of reducing the end-to-end latency of TT traffic and increasing the available bandwidth for BE traffic. The key idea is to adapt this problem to the no-wait job-shop scheduling problem [60]. The authors proposed an exact solution using the CPLEX **Integer Linear Programming** (**ILP**) solver with Big-M logical expressions and a compression algorithm in post-processing to minimize the guard bands. To tackle the scalability issue, a heuristic approach based on Tabu search was also proposed.
- **SMT-FRAG (*a1*):** Jin et al. [47] proposed a no-wait-based approach to improving schedulability by introducing a joint scheduling with fragmentation framework. The key idea is to jointly determine the number and size of fragments for individual streams during the schedule construction. The scheduling problem was formalized using a **Satisfiability Modulo Theories** (**SMT**) formulation and solved using the Z3 solver. To address the scalability issue, the authors also proposed a heuristic solution based on fixed-priority scheduling.
- **DT (*a2*):** Zhang et al. [105] addressed the high computational overhead issue associated with the non-overlap constraint checking during the scheduling process. The authors proposed a stream-aware model conversion algorithm that accelerates the scheduling process for the zero-jitter model. It employed the divisibility theory to detect collision and speeded up this process by only comparing the first instances of two streams. In addition, this work introduced an efficient incremental searching strategy without traceback, further reducing the runtime overhead.

4.1.2 Wait-allowed. In the wait-allowed model, frames can be stored in the egress queue and forwarded at a later time. It thus has a larger solution space compared with that of the no-wait

[2]Some selected works proposed both exact and heuristic solutions. In this article, we only evaluate one of them based on their key contributions to make the review and performance comparison more concise and informative.

scheduling model. We evaluate the following seven works that employ a combination of the FR and wait-allowed model to solve the real-time scheduling problem in TSN.

• **SMT-WA (*b1*):** Craciunas et al. [22] focused on providing accurate modeling for the behavior of TAS-based scheduling mechanisms in wait-allowed scenarios. They first formalized the constraints for constructing valid schedules using SMT formulation based on the wait-allowed model. To ensure correctness, a key contribution of their work is to first introduce the queuing isolation model such as frame- and stream-based isolation in the TSN field.

• **AT (*b1*):** Oliver et al. [69] addressed the challenge that GCLs only support a limited number of entries in practical implementations. Their study introduced a window-based scheduling method that applied array theory to an SMT solver, taking the maximum number of GCL entries as the algorithm input. In addition, this work incorporated the queue assignment and isolation model into the solution.

• **CP-WA (*b1*):** Vlk et al.[97] focused on modeling deterministic TT traffic through CP. CP provides a more efficient solution to the TSN real-time scheduling problem with an All-different Constraint compared to other formalizations such as SMT and ILP. This work also proposed a decomposition optimization to enhance the scalability, which alternated between routing and scheduling searches.

• **SMT-PRE (*b1*):** Zhou et al. [110] aimed to increase schedulability by integrating the preemption feature. An SMT-based approach was proposed to assign streams to express class and preemptable class while jointly determining the schedule. The allowed maximum number of preempted fragments was taken as the algorithm input.

• **I-OMT (*b2*):** Jin et al. [46] also addressed the practical limitation of GCL length. Instead of setting a hard constraint on the GCL length, this work minimized the GCL length by proposing an iterative-**Optimization Modulo Theories** (**OMT**)-based approach to scheduling streams by group. The queue assignment is calculated for each stream based on its payload size, shortest routing path, and deadline. Then it is taken as the input of the iterative OMT solver. The stream-based isolation is formulated as a constraint to guarantee the correctness of the queue assignment.

• **LS-TB (*b2*):** Vlk et al. [95] addressed the challenge of poor scalability in scheduling large-scale TT traffic in TSN, an inherent problem when using general third-party solvers. Their proposed scheduler can revert to a previous search stage and modify the timing and queue assignment if the current frame conflicts with any other scheduled frames.

• **LS-PL (*b2*):** Bujosa et al. [16] focused on improving scalability in scheduling large-scale TT traffic in TSN. The authors proposed a heuristic algorithm that allocates links into phases based on their dependency and then schedules these links phase by phase. The algorithm also determines the queue assignment during the heuristic search. This work covered both the no-wait and wait-allowed model, offering flexible jitter control under different scenarios.

4.2 Joint Routing and Scheduling (JRS) Methods

A feasible schedule may not be found when the routing paths of the streams are pre-determined under the FR model. By contrast, the JRS-based methods allow the scheduler to jointly determine the routes and schedules for each stream, thus providing opportunities to offer better network resource utilization and schedulability.

4.2.1 No-wait. The following five methods employ a combination of the JRS model and the no-wait model.

• **JRS-NW-L (*c1*):** Falk et al. [29] proposed an ILP-based approach to determine the routing path of each stream and the schedule of the stream set. Different from using the Big-M formulation

commonly employed by other ILP-based models (e.g., [27, 29, 83, 85]), the authors used the indicator constraints to address the logical constraints.

• **JRS-NW (*c1*) :** Hellmanns et al. [41] tackled the high computational complexity of the JRS-based scheduling problem. The authors first evaluated the impact of stream set scale and network scale on the schedulability performance and then provided an optimization framework to reduce computational overhead. The framework includes three components: input optimization, model generation optimization, and solver parameter tuning.

• **LS (*c2*) :** Pahlevan et al. [70] proposed a heuristic-based list scheduling algorithm to address the scalability issue in JRS-based scheduling methods. The proposed heuristic-based list scheduling algorithm searches all potential release times on a route and only moves to the next route when no available release time remains on the current route. The search order is governed by the hops of each stream's shortest path. Notably, this algorithm has no backtracking mechanism, thereby it returns infeasible once the algorithm traverses all paths of one flow without finding a solution.

• **I-ILP (*c2*) :** Atallah et al. [10] aimed to design an efficient framework to compute no-wait schedules and multicast routing in large-scale TSN. Their solution has three key techniques: iterative ILP-based scheduling for enhanced scalability, **Degree of Conflict** (**DoC**)-aware partitioning for stream grouping, and **DoC-aware multicast routing** (**DAMR**).

• **CG (*c2*) :** Falk et al. [30] tackled the critical issue of computational overhead in existing FR and no-wait based methods. Their approach constructs a conflict graph to capture the collision between individual stream's transmission time, and thus accelerate the solving process. The key idea is to identify an independent set within this graph and gradually expand it to obtain a valid schedule. Based on the search state, the solution automatically selects between a quick algorithm or ILP solver, strategically combining heuristic and exact methods.

4.2.2 Wait-allowed. The following two methods use a combination of JRS and wait-allowed models.

• **JRS-WA (*d1*) :** Schweißguth et al. [83] first proposed the JRS framework and addressed the issue that FR-based methods may exclude feasible solutions without considering routing in the design space. The authors introduced an ILP-based approach that simultaneously decides the routing path and schedule. In addition, the authors improved the search speed by excluding infeasible routing paths during pre-processing, without hurting the schedulability.

• **JRS-MC (*d1*) :** In this work, Schweißguth et al. [85] further extended the above ILP-based JRS scheduling approach to support multicast traffic streams. The authors argued that including multicast features requires more than a trivial extension from the unicast model, necessitating additional scheduling constraints to prevent loops and negative latency. The authors investigated various objectives, examining the improvement of schedule quality and tradeoffs between schedule quality and runtime overhead. The authors also introduced optimization techniques for preprocessing and model generation while demonstrating that these enhancements significantly reduce the solver's runtime.

Table 1 summarizes the 17 scheduling methods in chronological order. A (✓) symbol indicates that the method was presented in the original paper but we do not implement it.

4.3 Extended Constraints and Alternative Approaches in TSN Scheduling

In addition to the fundamental feasibility requirement, practical deployments of TSN systems must often satisfy several other system-level constraints. Moreover, beyond traditional centralized scheduling approaches, several alternative scheduling mechanisms have been proposed. This

Table 1. A Summary of the System Models and Scheduling Approaches in the Studied TSN Scheduling Methods

Article	Year	Fully schedulable	Strictly periodic	No-wait	Window -based	Queueing	Routing	Multicast	Heuristic	Exact	Algorithms	Enhancements	Main focus
Dürr et al. (SMT-NW) [27] (*a1*)	2016	✓	✓	✓					(✓)	✓	ILP (Tabu)		Scalability
Craciunas et al. (SMT-WA) [22] (*b1*)	2016	✓	✓			✓				✓	SMT		Schedulability
Schweissguth et al. (JRS-WA) [83](*d1*)	2017	✓	✓				✓			✓	ILP	Paths reduce	Schedulability
Oliver et al. (AT) [69] (*b1*)	2018	✓			✓	✓				✓	SMT	Array-theory	GCL length
Falk et al. (JRS-NW-L) [29] (*c1*)	2018	✓	✓	✓			✓			✓	ILP	Logic indicator	Schedulability
Pahlevan et al. (LS) [70] (*c2*)	2019	✓	✓	✓			✓		✓		List scheduler		Scalability
Schweissguth et al. (JRS-MC) [85](*d1*)	2020	✓	✓				✓	✓		✓	ILP	Paths reduce	Multicast
Atallah et al. (I-ILP) [10] (*c2*)	2020	✓	✓	✓			✓	✓	✓		Iterative-ILP		Scalability, multicast
Jin et al. (I-OMT) [46] (*b2*)	2020				✓	✓			✓	(✓)	Iterative-OMT		Scalability, GCL length
Falk et al. (CG) [30] (*c2*)	2020	✓	✓	✓			✓		✓	(✓)	Conflict-graph		Scalability
Hellmanns et al. (JRS-NW) [41] (*c1*)	2021	✓	✓	✓			✓			✓	ILP	Path cut-off	Scalability
Jin et al. (SMT-FRAG) [47] (*a1*)	2021	✓	✓	✓					(✓)	✓	SMT (WCRT)	Fragmentation	Schedulability
Vlk et al. (CP-WA) [97] (*b1*)	2021	✓	✓			✓			(✓)	✓	CP (Decompose)		Scalability
Vlk et al. (LS-TB) [95] (*b2*)	2022		✓			✓			✓	(✓)	List scheduler	Traceback	Scalability
Bujosa et al. (LS-PL) [16] (*b2*)	2022		✓			✓			✓		List scheduler	Per-link search	Scalability
Zhou et al. (SMT-PRE) [110] (*b1*)	2022	✓								✓	SMT	Preemption	Schedulability
Zhang et al. (DT) [105] (*a2*)	2022	✓	✓	✓					✓	(✓)	Divisibility		Scalability

body of work is not included in our experimental evaluation, but is summarized in this section for the completeness of the systematic review.

Delay and jitter minimization. Minimizing the end-to-end delay of the streams is one of the most critical design objectives of TSN schedulers. Scheduling methods based on the no-wait model (e.g., [10, 27, 29, 30, 41, 47, 70]) aimed to reduce the end-to-end delay by eliminating the queuing delay at each TSN bridge. On the other hand, [83, 85] achieved this objective by incorporating additional objective functions (e.g., minimize the average delay among streams), and [47] leveraged stream fragmentation to reduce the transmission delay by allowing more parallel transmissions. The experienced maximum jitter is another critical metric to evaluate the performance of TSN-based applications [12]. In [69], the authors minimized the maximum jitter of TSN streams by controlling the window size. Zhang et al. [104] conducted a case study in an underground mining scenario and proposed a heuristic solution to meet the designated delay and jitter requirements.

Number of queues minimization. Minimizing the number of queues used per hop is another important design objective. Since TT traffic must have exclusive queue access to ensure determinism, it is crucial to limit the number of utilized queues, reserving the remaining ones for asynchronous traffic. As discussed in Section 3.3, no-wait based scheduling methods such as [10, 27, 29, 30, 41, 47, 70] utilized only one queue. Furthermore, scheduling methods based on the wait-allowed model could incorporate an additional objective function to minimize the queue usage [22, 97].

Heterogeneous traffic. Several studies [9, 27, 37, 43, 78, 101] investigate how to enhance the performance of non-TT traffic (e.g., AVB and BE) while guaranteeing the timing constraints of TT traffic. For instance, Reusch et al. [78] proposed a scheduling framework to support the co-existence of asynchronous traffic and TT traffic using a network calculus-based approach. Yao et al. [101] aimed to optimize the network utilization for other traffic by introducing network remaining time as the objective. Han et al. [37] proposed a traffic scheduling algorithm combined with ingress shaping to reduce the e2e delay of BE messages.

Reliability. Ensuring reliable frame transmissions in TSN networks is another crucial research topic and has been extensively studied, e.g., [20, 25, 32, 56, 61, 79, 93, 96, 109]. For example, Mahfouzi et al. [56] introduced an SMT-based scheduling framework that incorporates a stability condition derived from the jitter and delay of control traffic. Reusch et al. [79] proposed a dependability-aware JRS framework that uses redundant disjoint routes to tolerate link failures. Craciunas et al. [20] proposed a robust out-of-sync scheduling approach for TSNs that can accommodate synchronization failures. Min et al. [61] proposed a JRS-based scheduling method to incorporate in-frame replication and elimination for reliability (FRER). Vlk et al. [96] proposed

a hardware enhancement that eliminates the need for a guard band to protect TT traffic, thereby improving the bandwidth utilization for other types of traffic.

Online reconfiguration. A range of studies (e.g., [8, 33, 35, 55, 71, 74, 77, 102]) have been conducted to address the online reconfiguration problem in TSN networks. For example, Raagaard et al. [77] proposed a heuristic algorithm for schedule reconfiguration in fog computing platforms, handling newly added and removed streams. Yu et al. [102] proposed an online scheduling approach to deal with dynamic virtual machine migrations in multicast TSN networks, including offline schedule construction and online rescheduling. Pang et al. [71] identified a deadlock issue during schedule updates, and an online algorithm is proposed to generate conflict-free schedules for the update. Patti et al. [74] proposed an online EDF-based scheduling algorithm in TSN to provide support for event-driven real-time traffic by dynamically updating the priority mapping on TSN switches. Gartner et al. [35] developed a flexibility-aware heuristic scheduling algorithm to enhance the probability of successful reconfiguration.

Distributed scheduling. While distributed scheduling algorithms are commonly associated with asynchronous shapers, e.g., ATS and CQF, without relying on network-wide synchronization, several studies have explored distributed approaches specifically for the TAS shaper. These efforts often aim to enhance scalability, efficiently handle dynamic traffic changes, or improve fault tolerance compared to centralized methods. For instance, Hellmanns et al. [40] proposed a hierarchical scheduling method to locally compute schedules within sub-networks and handle inter-sub-network traffic scheduling using simulation traces, significantly reducing complexity. Bush et al. [17] focused on efficient dynamic reconfiguration by pre-computing and storing partial schedules on local switches to allow faster generation of valid schedules upon changes. Zhou et al. [108] introduced TSN-VM, a distributed framework designed to mitigate scheduling violations stemming from synchronization errors or jitter. Besides, it is worth noting that some reinforcement learning-based scheduling approaches [62, 80, 100] inherently operate in a distributed manner, making local configuration decisions based on the observed state at individual switches.

Learning-based methods. In recent years, an increasing number of learning-based methods [38, 42, 57, 62, 80, 92, 100, 106] have been proposed to solve the TSN scheduling problem. For instance, Yang et al. [100] proposed a graph convolutional network-based deep reinforcement learning solution for the joint optimization of TT and other traffic types. He et al. [38] introduced a method based on deep reinforcement learning to enhance the scalability of TSN scheduling. Roberty et al. [80] focused on a case study that explores the application of deep reinforcement learning to reduce the scheduling time in IEEE 802.1Qbv networks. Min et al. [62] proposed a JRS-based scheduling method using deep reinforcement learning that improves schedulability and reduces maximum link utilization. Zhao et al. [106] utilized a Q-learning approach to optimize average e2e delay, addressing the time overhead and adaptivity issues of traditional methods. Sun et al. [92] employed a deep reinforcement learning framework based on the **Proximal Policy Optimization** (**PPO**) algorithm to minimize TT traffic delay. Hong et al. [42] introduced an online deep reinforcement learning framework using a Convolutional Neural Network for adaptive scheduling and resource allocation by optimizing the allocation of sending time slots.

5 Testbed Validation

To validate the correctness and effectiveness of the studied TSN scheduling methods on **Commercial Off-the-Shelf** (**COTS**) TSN hardware, we set up a TSN testbed and implemented all the scheduling algorithms. The testbed consists of 8 bridges and 8 ESs organized in a ring topology as shown in Figure 4(a). Each bridge is an FPGA hardware-based TTTech TSN evaluation board, and each ES is implemented using the Linux Ethernet stack with an external **Network Interface**

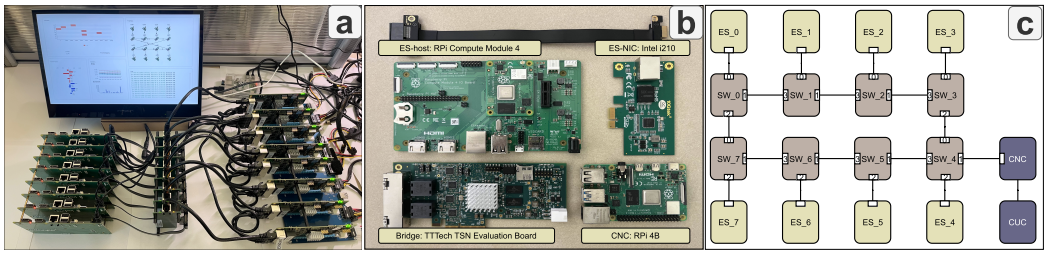


Fig. 4. Overview of the TSN testbed with 8 bridges and 8 ESs.

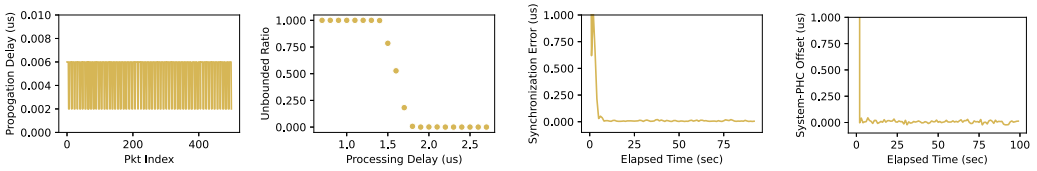
Controller (**NIC**) Intel i210 as shown in Figure 4(b). The network is set up following the ring topology as shown in Figure 4(c), which is commonly applied in industrial scenarios [73]. We use the Linux PTP stack [19] with the gPTP profile for synchronization on ESs, and the bridge implements its own synchronization stack. The synchronization traffic is set with a priority higher than the best-effort traffic and lower than the critical traffic.

There are two main objectives of this testbed: (i) to validate and calibrate the parameters commonly used in the literature's model assumptions, and (ii) to validate if the performance of the scheduling methods on the testbed is consistent with that derived through analysis. Given the limited scale of our testbed, we focus on functional validation rather than performance comparison using the testbed.

5.1 Measurements of Delays and Synchronization Error

As mentioned in Section 3, most existing TAS-based scheduling methods assume that the processing delay, propagation delay, synchronization error, and clock offset on ESs are constant or can be bounded. *To the best of our knowledge, however, there is no existing study validating these assumptions through experimental measurements in real-world TSN testbeds.* While some studies have conducted testbed measurements, they typically focus on end-to-end performance instead of measuring each delay component [64, 65, 94]. Others, such as [54], only focus on measuring the delay of the synchronization function without considering general frame forwarding. We argue that such validation is critical as it provides the foundation for both existing and future TAS-based scheduling method designs.

- **Propagation Delay:** To measure the propagation delay, we directly connected one talker and one listener with a CAT7 cable while measuring the **round-trip time** (**RTT**) of a stream using the hardware timestamping function supported by the NIC. As shown in Figure 5(a), the propagation delay in this one-hop setting is bounded between 2 ns and 6 ns, with a 4 ns jitter due to the measurement inaccuracy.
- **Processing Delay:** Since we cannot measure the processing delay on the TTTech evaluation board directly, we infer its upper bound by observing the e2e delay of a stream. Specifically, we gradually increased the potential upper bound of the processing delay in the TAS configuration until all frames' e2e delay can be statistically bounded within the test duration. Figure 5(b) shows that the one-hop processing delay can be bounded within 1.9 μs in our testbed.
- **Synchronization Error:** Figure 5(c) shows the synchronization error measured on the testbed, which is reported by the logs of the Linux PTP stack. It can be observed that the synchronization error becomes stable after 5 seconds. The large values observed in the first 5 seconds are mainly due to the grand master clock election process [6]. After that, the synchronization error can be bounded within 10 *ns*.



(a) Propagation delay (b) Processing delay (c) Synchronization error (d) Clock offset on system

Fig. 5. Testbed measures on the propagation delay, processing delay, synchronization error, and system clock to physical clock offset.

— **Clock Offset on System:** Figure 5(d) shows the clock offset from the system clock in the application to the physical clock in the network card, which is also reported by the Linux PTP stack. Similar to the synchronization error, the clock offset is also large at the beginning, then it is bounded within 50 ns.

The above measurement results provide the calibration values of the propagation delay, processing delay, and synchronization errors from the real-world testbed. Thus, we use these empirical bounds in the subsequent simulation-based evaluation to ensure the robustness of the simulation model. Specifically, we set the propagation delay, processing delay, synchronization error, and clock offset as 6 ns, 1.9 μs, 10 ns, and 50 ns, respectively.

5.2 Performance Validation

To confirm the theoretical performance and ensure the correctness of our implementations, we implement 16 out of the 17 scheduling methods on the testbed where SMT-FRAG is not implemented because its required fragmentation size for each stream is even smaller than the lower bound of the window size on the hardware device.

We conduct the performance validation using a small-scale stream set consisting of 8 streams to simplify the hardware configuration. The stream set includes four streams with a payload size of 100 bytes, two streams of 200 bytes, and two streams of 400 bytes. Each stream has a common period and deadline of 1 ms. Each stream has a unique talker but may have shared listeners. The streams are routed on the same ring topology, with their routing paths determined by the evaluated methods. After deploying the release times, queue assignments, and GCL configurations that are generated from each of the 16 methods on the testbed, we record the e2e delay of 10000 frames for each stream.

Figure 6 compares the measured e2e delays of individual methods on the testbed (yellow line) and the analyzed worst-case delay from the simulation (red line). We present three representative results for conciseness, as other results exhibit similar patterns. **Overall, our testbed results validate the correctness of all the methods** since the corresponding measurement results of each method are always bounded by the analyzed worst-case e2e delays.

Beyond that, we have two important observations. First, most streams experience a relatively stable delay (<100 ns variation), but some streams are observed to have varied delays under certain methods. For example, the delay of Stream S0 under LS-PL gradually increases to around 12 μs. Then it drops to 9.8 μs suddenly. We believe that these drifts are mainly caused by the collisions between synchronization traffic and TT traffic, which increases the clock drift between the talker and listener over time. Subsequent synchronization recovery procedures eliminate such clock drift, restoring the delay to its normal state. Secondly, a large gap can be observed between the testbed measurements and the simulation results across different methods, with a maximum gap of about 5 μs recorded from S2 with I-ILP. This gap primarily stems from two factors: (1) an enforced error

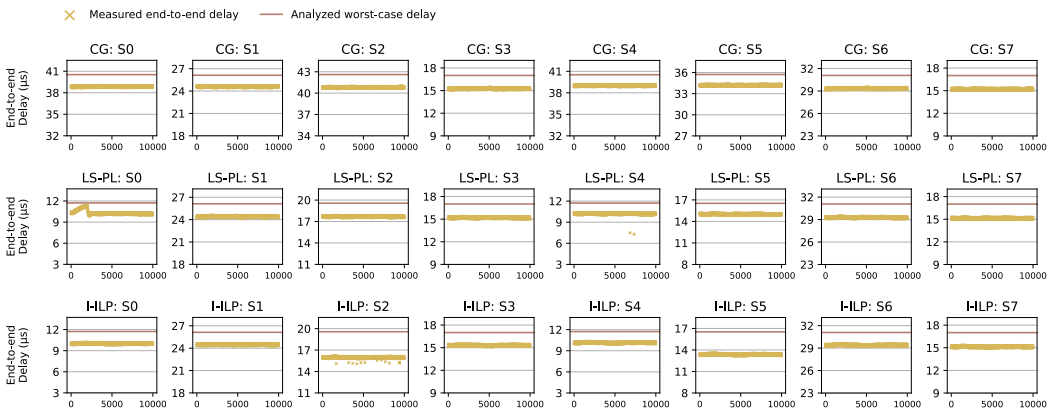


Fig. 6. Performance validation on the e2e delays of eight TT traffic on the real-world TSN testbed.

margin of up to 3.2 μs by the TTTech bridge to accommodate timing errors, and (2) an up to 1.9 μs processing delay on the bridge identified during our measurements.

In addition to the schedulability of stream set under a scheduling method, we are also interested in evaluating the associated schedule quality, including GCL length, delay, jitter, link utilization, and queue utilization, as summarized in Table 3. Since GCL length, link utilization, and queue utilization are determined directly by the schedule, they can be measured from the schedule generated by the simulation. We validate e2e delay and jitter using our testbed here. For e2e delay, we observe that three methods, AT (292.1 μs), JRS-WA (264.3 μs), and JRS-MS (527.1 μs), exhibit significantly higher e2e delays compared to other methods with an average e2e delay of 20.7 μs. These results are consistent with the corresponding simulation outcomes, as discussed in Section 7.3. For jitter, the maximum values observed for JRS-WA at 3.4 μs and LS-PS at 3 μs are slightly higher than the sub-microsecond jitter observed for other methods. However, we believe this is not related to any specific scheduling method but rather caused by variations in processing delay and measurement inaccuracies during the testbed experiments. In addition, unlike the larger maximum jitter predicted by simulations, we did not observe significantly high jitter for window-based methods during the testbed experiments. This can be attributed to the fact that frames were consistently transmitted at specific times within their allocated windows, leaving a substantial portion of the window unused on the real testbed in our test case.

6 Simulation-Based Experimental Setup

6.1 Parameter Settings

To ensure a fair evaluation among the selected TSN scheduling algorithms, we followed the parameter settings below in the experiments, which are summarized in Table 2.

6.1.1 Stream Set Settings. We control the randomly generated TSN stream set by tuning the following parameters: (i) number of streams, (ii) stream period, (iii) number of frames, (iv) stream payload, and (v) stream deadline.

Number of streams. In each randomly generated stream set, the number of streams follows a uniform distribution within the range of [10, 220] with a step size of 30. The maximum number of streams is set to 220 to encompass the settings employed in both simulation-based studies and real-world applications. In our experiments, when the number of streams reaches 220, the average system utilization surpasses the recommended upper bound for industrial applications' critical traffic [1], resulting in a very low schedulability ratio and impractical runtime for most methods.

Table 2. Parameter Settings for the Performance Evaluation

	Parameter	Type	Value
Stream set	Stream period (ms)	Sparse single	{2}
		Dense single	{0.4}
		Sparse harmonic	{0.5, 1, 2, 4}
		Dense harmonic	{0.1, 0.2, 0.4, 0.8}
		Sparse inharmonic	{0.25, 0.5, 1.25, 2.5, 4}
		Dense inharmonic	{0.05, 0.1, 0.25, 0.5, 0.8}
	Stream payload (bytes)	Tiny size	50
		Small size	50 - 500
		Medium size	200 - 1500
		Large size	500 - 4500
		Huge size	1500 - 4500
	Number of streams	-	{10, 40, 70, …, 190, 220}

	Parameter	Type	Value
Stream set	Number of frames	-	{8, 16, 32, …, 2048, 4096}
	Stream deadline (ms)	Implicit deadline	Equal to period
		Relaxed deadline	NW + {0.1, 0.2, 0.4, 0.8, 1.6}
		Normal deadline	NW + {0.01, 0.025, 0.05, 0.1, 0.2, 0.4}
		Strict deadline	NW + {0, 0.01, 0.02, 0.025, 0.05}
		No-wait deadline	NW
Network	Topology	Linear, Ring, Tree, Mesh	-
	Number of bridges	-	{8, 18, 28, …, 78}
	Number of links	-	{30, 32, 36, …, 386}
	Number of queues	-	8

Stream period. Following the TSN profile for industrial automation use cases in IEC/IEEE 60802 [26], we set the range of the stream periods as [50μs, 4ms]. However, randomly generated stream periods are less meaningful as the stream periods in real-world TSN applications typically follow specific patterns in corresponding industrial sectors [66]. Thus, we define 6 stream period types, as shown in Table 2, to include all the commonly employed periodicity settings.

Number of frames. Within a network cycle (i.e., the period that GCL repeats itself), the number of frames is determined by the combination of the number of streams and their periods. In our experiments, considering the network cycle as the least common multiple of stream periods, the number of frames can range from 10 to 7842. Given its exponential and continuous distribution, we sort these values into bins $\geq 8, \geq 16, \ldots, \geq 4096$ as shown in Table 2.

Stream payload. The stream payload size is the amount of data payload (in bytes) carried by one instance of the stream. According to the IEEE 802.1Q standard [45], if the payload size exceeds the MTU size (typically 1500 bytes), the stream instance can be fragmented into multiple fragments, each of which is transported by one frame. In the experiments, we define 5 payload size types (see Table 2) based on the typical configurations in industrial applications.

Stream deadline. Theoretically, the minimum e2e delay experienced by a stream equals the sum of propagation delay, processing delay, and transmission delay along the shortest routing path (i.e., the e2e delay under both FR and no-wait model). Thus, we set the minimum deadline of each stream to its delay under the no-wait model (denoted as NW), which can be calculated according to our hardware-based measurement results in Section 5. We define 5 stream deadline types (see Table 2) to aid the generation of random stream sets in our experiments.

6.1.2 Network Settings. The generation of a TSN network in our experiments is controlled by the following parameters:

Network topology. We employ four commonly used topologies, i.e., linear, ring, tree, and mesh.

Number of bridges and links. The number of bridges in the network ranges from 8 to 78 (with a step size of 10), where the network diameter reaches the synchronization accuracy limitation in IEEE 802.1AS [6] under our topology settings. The number of links is determined accordingly under different network topologies, as detailed in Table 2.

Link rate and number of queues. In our experiments, unless specified otherwise, we employ gigabit bridges with a line rate of 1 Gbps, which is offered by most vendors [15]. The number of queues on each egress port is fixed to 8, also a common setting in TSN bridges. We also assume that all 8 queues are exclusively dedicated to handling TT traffic.

6.2 Algorithm Implementation

We implement all 17 TAS-based scheduling methods in Python3, as some works rely on third-party software, which all provide an interface in Python3. Specifically, for SMT/OMT-based methods, we use the Z3 solver to support the required theories and logical formulas such as array and arithmetic theory [23]. For ILP-based methods, we use the Gurobi optimizer, one of the most advanced ILP solvers [36].[3] For methods without relying on third-party software, we implement them from scratch using native Python. Please note that some works proposed multiple scheduling methods, including both heuristic and exact solutions. For the evaluation efficiency, we only implement the proposed solution claimed to be the main contribution of the article. The specific implementation of each work is described below.

SMT-WA. This work studied both the frame-based model and stream-based isolation model, showing that the frame-based approach can enhance schedulability with only a marginal runtime overhead (up to 13%). Thus, we only implement the proposed frame-based approach in our study.

JRS-NW-L/JRS-MC. Because the model generation optimization techniques proposed in JRS-NW-L and JRS-MC were found to be counter-effective [41], we omit such optimizations in our simulation to reduce the execution time.

SMT-NW. The exact solution in SMT-NW is selected and implemented as it shows better overall performance in our evaluation compared with the proposed heuristic solution.

LS-TB. We omit the “global conflict set" data structure used in the article as it is rarely called (only 0.96%) in the problem-solving process.

LS. The FINDIT function is not described in detail, and thus we implement it using a binary search-based strategy.

SMT-FRAG. We only implement the exact solution in SMT-FRAG, as the proposed heuristic-based fixed-priority scheduling method involves complex worst-case delay analysis, which is challenging to implement and verify for correctness.

CP-WA/LS-TB. We omit the “presence" decision variable used to select streams in CP-WA and LS-TB to optimize the number of scheduled streams. We consider a set of streams to be schedulable only when all streams are scheduled.

I-OMT. In OMT-based methods, we introduce an indicator variable to map each frame to only one window. This is to simplify the *time validity constraint* to make the original formulation practical without sacrificing schedulability.

For methods that require additional parameters, we directly follow their default settings in the original papers. For example, we set the maximum number of windows to 5 for AT, the maximum fragment count to 5 for SMT-FRAG, the maximum iteration number to 100 for I-ILP, the maximum preemption count to 5 for SMT-PRE, and assume a release point at 0 for all streams in LS-TB. In addition, as suggested in the IEEE 802.1Qcc standard [4], we apply the shortest path routing algorithm to construct the routing path for each stream in FR-based methods.

6.3 Evaluation Environment

Our experiments are conducted on Chameleon Cloud, an NSF-sponsored public cloud computing platform [49]. We utilized 8 nodes equipped with 2x AMD EPYC® CPUs, 64 cores per CPU with a clock speed of 2.45 GHz, and 256 GB DDR4 memory. The operating system used was Ubuntu

[3]Following the original papers, we use the CPLEX ILP solver for JRS-NW-L for the logical indicator, and the IBM CP Optimizer for CP-WA, and the Sklearn library [75] to implement the spectral clustering based stream set partition algorithm for I-ILP.

Table 3. A Summary of the Evaluation Metrics in the Study

Evaluation	Metric	Definition
Schedulability	Schedulable ratio	The ratio of schedulable stream sets
	Schedulability advantage	The pair-wise comparison on same stream sets
Scalability	Running time	Total running time of an algorithm
	Memory usage	Peak memory usage of an algorithm
Schedule quality	GCL length	Maximum GCL length across all links
	Overall delay & jitter	Average end-to-end delay and jitter across all streams
	Link utilization	Maximum bandwidth utilized on links across all links
	Queue utilization	Maximum number of utilized queues across all links
Fault tolerance	Reliability*	Robustness to meet traffic characters when faults occur
	Integrity*	Correctness of payload information protected during faults

20.04 LTS. To make the benchmark robust and representative, we ran a total of 38400 problem instances covering all combinations of our parameter settings in Table 2, with 64 experiments running simultaneously on a single node at any given time. To avoid any interference among experiments and enable concurrency, a single process with a maximum of 4 GB RAM and 4 threads was dedicated to each experiment. We set a 2-hour runtime limit for all the methods where most of them took less than 2 hours according to our evaluation. If any thread of the algorithm exceeded the time threshold, the algorithm was terminated and returned "unknown". We fixed the random seed to 1024.

6.4 Evaluation Metrics

Based on the research objectives and application scenarios discussed in Sections 3 and 4, we summarize the commonly used evaluation metrics in Table 3. As we do not consider network faults in this work, we mainly focus on the first eight metrics in our performance evaluation.

7 Experimental Evaluation

In this section, we perform a comprehensive simulation-based evaluation for the 17 TAS-based scheduling methods by comparing their *schedulability*, *scalability,* and *schedule quality* using the evaluation metrics presented in Table 3.

7.1 Schedulability

7.1.1 Setup. As discussed in Section 6.3, we set a 2-hour timeout and 4 GB RAM limit for each method. Therefore, each method in our evaluation outputs one of the three results for each randomly generated stream set: schedulable, unschedulable, and unknown. Due to the presence of unknown results, we cannot precisely quantify the schedulability performance of each method. To overcome this issue, we devise two evaluation scenarios to ensure a fair comparison.

Evaluation Scenario 1 (ES1). In ES1, we conduct a comprehensive cross-evaluation of all 17 methods by employing a conservative statistical strategy to calculate ***schedulable ratio (SR)***. Specifically, the SR of each method is defined as the ratio of schedulable stream sets to all the generated stream sets. Such SR plays as the schedulability lower bound because all the unknown results are deemed as unschedulable.

Although SR can to some extent reflect the schedulability of the studied methods, it can be unfair to those methods requiring higher resource consumption where a considerable portion of the stream sets with unknown results might be schedulable. To mitigate the influence of unknown results on the performance comparison, a straightforward solution is to only consider the experimental settings where all methods produce known results, i.e., schedulable or unschedulable.

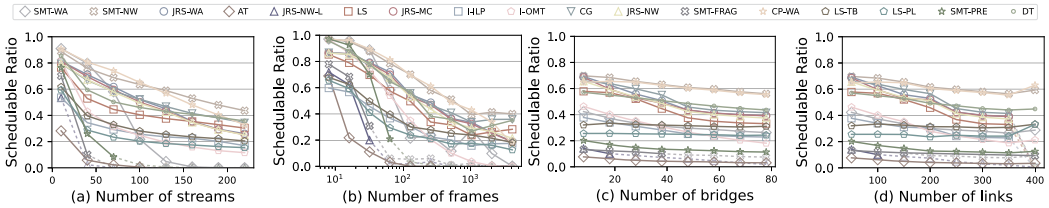


Fig. 7. SR comparison under different stream set and network settings by varying the parameter values.

However, the experimental settings that yield known results for all methods could be very small, making the performance comparison statistically insignificant.

Evaluation Scenario 2 (ES2). To tackle this issue, in ES2, we conduct a pairwise performance comparison between any two methods by developing a novel metric, called ***schedulability advantage (SA)***, which is calculated only based on the known results for both methods. SA of A to B, denoted as $\Phi(A, B)$, quantifies the degree to which method A outperforms method B. Specifically, $\Phi(A, B)$ represents the ratio of the number of stream sets where method A returns schedulable while method B returns unschedulable to the number of stream sets where both methods A and B return known results. Therefore, if $\Phi(A, B) > \Phi(B, A) = 0$, we say that method A dominates method B as there does not exist any stream set where method B can find a schedulable solution but method A cannot.

7.1.2 Results. We conduct extensive experiments based on the above two evaluation scenarios.

In the first set of experiments, we evaluate the SRs of all the methods by varying the parameter settings summarized in Table 2. Specifically, Figure 7 shows the SR as functions of the number of streams, number of frames, number of bridges, and number of links, respectively. In each subfigure, only one parameter is varied with all other parameters being fixed. We use dashed lines to denote data points comprising over 90% unknown results. Figure 9 shows the SR of each method under different stream period types, stream payload types, stream deadline types, and topologies, respectively.

The second set of experiments performs the pairwise comparison using the SA metric and the results are shown in a heatmap in Figure 8. Specifically, each cell represents the $\Phi(A, B)$ value, where the row-index represents method A and the column-index represents method B. Darker cells signify higher SA values, while light yellow represents a zero SA value. We use ✗ to indicate that the method in a given row dominates the method in the corresponding column. Moreover, on the vertical axis, methods are sorted from high to low based on their average SA values.

7.1.3 Discussion. Based on the obtained experimental results, we now present our discussion across two dimensions of granularity. First, we perform evaluation comparisons among different scheduling models discussed in Section 3.3 to show their pros and cons. Next, we delve into the performance evaluation of individual scheduling algorithms to discuss their advantages and limitations.

Model comparisons. We discuss the model comparison by categorizing two sets of different scheduling models: (a) models with varied performance, and (b) models with stable performance between SR and SA. This classification is based on their observed trends in our experiments. The first set of comparisons include: (1) JRS model and FR model, (2) FRAG/PRE model and non-FRAG/non-PRE model, and (3) no-wait model and wait-allowed model. The second set of comparisons includes: (1) fully and partially schedulable models, (2) frame-based model and window-based model.

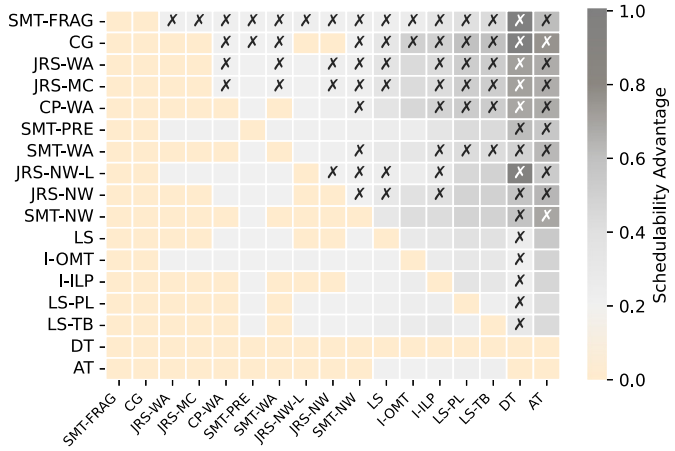
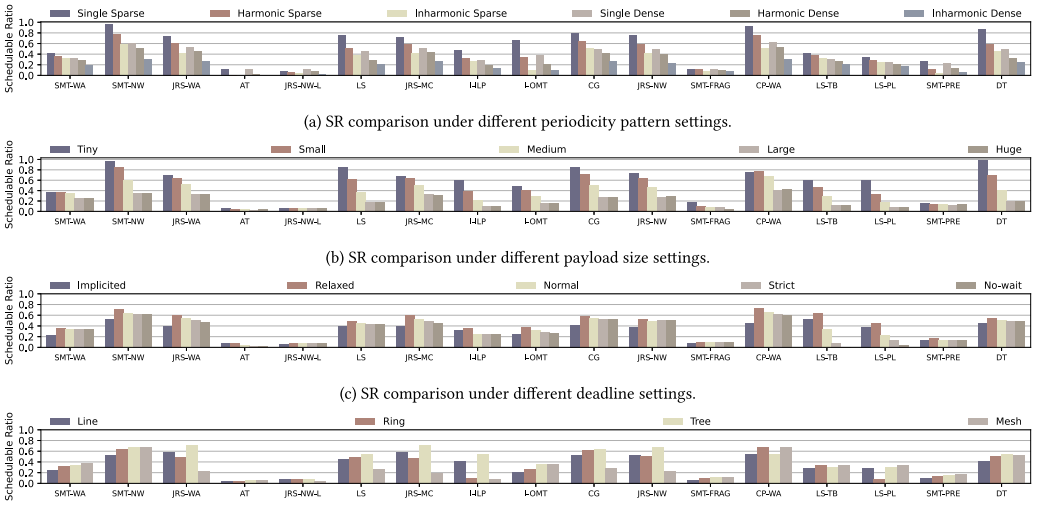


Fig. 8. Pairwise SA comparison among the studied scheduling methods.



(a) SR comparison under different periodicity pattern settings.

(b) SR comparison under different payload size settings.

(c) SR comparison under different deadline settings.

(d) SR comparison under different network topology settings.

Fig. 9. SR comparison under different stream set and network settings by varying the parameter types.

(a) Models with inconsistent performance on SR and SA. The results show that a complex model can achieve higher SA. However, it also incurs higher computation overhead, which may significantly limit its performance on SR.

• **JRS vs. FR model.** JRS model dominates FR model on SA, but it may cause lower performance on SR. For example, comparing in Figure 8, JRS-WA and JRS-MC dominate their counterparts under the FR model (SMT-WA and CP-WA), and JRS-NW, JRS-NW-L dominates its counterpart SMT-NW. However, the JRS model usually leads to lower SR compared to the FR model due to their incurred computation overhead. For example, as shown in Figure 7(c)–(d), along with the increase of the network scale, the methods with exact solutions under the JRS model (JRS-NW, JRS-NW-L, JRS-WA, JRS-MC) suffer larger performance degradation by 41.8% on average compared to that of the

methods under FR model (SMT-WA, SMT-NW, CP-WA) by 10.9%. The side effects of JRS model on SR can also be validated by comparing its performance under different topologies as shown in Figure 9(a). All JRS-based methods with exact solutions (JRS-WA, JRS-MC, JRS-NW, and JRS-NW-L) show significantly degraded SR under ring and mesh topologies than line and tree topologies, while most FR-based methods have improved SR on mesh topology.

- **FRAG/PRE vs. non-FRAG/non-PRE model.** In Figure 8, SMT-FRAG and SMT-PRE show the average SA values of 14.7% and 14.0%, respectively, outperforming the average of other methods at 10.4%. However, these methods experience significantly reduced schedulability due to their larger computational overhead. For example, in Figure 7(a)(b), although SMT-FRAG and SMT-PRE start with very high SR (70.1% and 76.4% respectively), their SR degrades sharply to below 10.0% after the number of streams and frames are increased to 70 and 128, respectively. This poor SR performance can be consistently observed by varying other parameters in Figures 7(c)(d) and 9.
- **No-wait vs. wait-allowed model.** The initial comparison between two FR-based methods (SMT-NW and SMT-WA) suggests that the wait-allowed method dominates the no-wait method on SA as expected due to its more flexible delay model from Figure 8. However, on the SR performance, SWT-NW surpasses SWT-WA when the workload is increased to 70 streams or 64 frames in Figure 7(a)(b), and SWT-NW consistently outperforms SWT-WA in Figure 7(c)(d). A similar pattern can be observed under JRS methods (e.g., JRS-NW and JRS-WA) that JRS-WA dominates JRS-NW on SA, but their difference in SR is negligible.

(b) Models with consistent performance on SR and SA. We find that for some methods, the schedulability improvement introduced by applying a complex model outweighs the correspondingly increased computational overhead, which leads to consistent performance improvement on both SA and SR. Such a trend can be found in the comparisons among fully schedulable model vs. partially schedulable model, and frame-based model vs. window-based model.

- **Fully vs. partially schedulable model.** We compare LS with LS-PL and LS-TB methods, all rooted in the LS-based heuristic approach. As shown in Figure 8, the fully schedulable model LS shows higher SA (7.24%) than the partially schedulable model LS-PL (2.8%) and LS-TB (4.29%). The comparison results are retained when evaluating the SR performance. As shown in Figure 7, the fully schedulable model LS also consistently outperforms the partially schedulable model LS-TB and LS-TL under varied workload and network parameters. Combined with the comparison results of no-wait/wait-allowed model, these results imply that enlarging the search space on the ES side (fully schedulable/partially schedulable) is more effective than enlarging the search space on the bridge side (no-wait/wait-allowed).
- **Frame-based vs. window-based model.** We compare AT with SMT-NW and SMT-WA which are all SMT-based exact approaches. As shown in Figure 8, the frame-based methods SMT-NW and SMT-WA dominate the window-based method AT on SA. Consistently, as shown in Figure 7, both SMT-NW and SMT-WA also consistently outperform AT in terms of SR by increasing either the workload or network scale. These results imply that the constraints applied on GCL length may significantly limit the schedulability.

Based on the above results and discussions on different scheduling models, we conclude with the following finding.

Finding 1. *Although complex TSN scheduling models (e.g., JRS, FRAG, PRE, and wait-allowed) can enhance the schedulability in theory, their incurred high computational overhead reduces the performance improvement in practice. They may even have counterproductive effects in resource-constrained systems.*

Algorithm comparisons. We now present the schedulability performance comparison among individual scheduling methods. We first perform comparisons between heuristic approaches and exact solutions. We then delve into heuristic approaches to examine the properties derived by individual heuristic designs.

(a) Heuristic vs. exact solutions. Apparently, although heuristic approaches may not match the performance of exact solutions, they show higher efficiency, especially under heavy workloads and restricted computational resources. Our results align with this expectation. For example in Figure 7, the exact solution SMT-WA outperforms heuristic LS-TB in SR when the number of streams is less than 100. However, when the number of streams keeps increasing, LS-TB remains stable, but SMT-WA rapidly declines to zero. Both methods are under the FR model and wait-allowed model. Similar trends can also be found by comparing other pairs of heuristic and exact solutions, such as SMT-NW vs. DT.

Due to their inherent efficiency, heuristic approaches can also benefit more from complex models compared to exact solutions. For example, comparing SR for the no-wait scheduling methods in Figure 7(a), heuristic CG and exact solution JRS-NW exhibit similar SR when the number of streams is less than 80. However, when the stream set size increases, CG outperforms JRS-NW with a widening gap. Both methods are under JRS model and no-wait model. Similar trends can also be found as the heuristic method I-OMT outperforms the exact method I-ILP consistently in Figure 7.

(b) Comparison among heuristic algorithms. Our experiment results show that the performance of four heuristic algorithms significantly degrades under certain specific scenarios. (1) For networks with routable topologies (i.e., ring and mesh), I-ILP demonstrates lower schedulability due to the inefficiency of its DAMR routing algorithm. For example, as shown in Figure 9(d), SRs of I-ILP drop from 41.6% under line topology and 54.6% under tree topology to 9.6% under ring topology and 7.4% under mesh topology. (2) LS-PL suffers from a notably low SR (7.0%) in networks with ring topology as shown in Figure 9(d). This is mainly due to the high likelihood of cyclic dependencies, causing frequent failures in its phase division algorithm. (3) Under strict deadline settings, both LS-TB and LS-PL show low schedulability due to their partially schedulable traffic model. This deficiency results in a drop in SR from implicit deadline setting (51.9%) to no-wait deadline setting (1.0%) as shown in Figure 9(c). (4) In the presence of inharmonic periodicity, I-OMT exhibits a reduced SR (10.4%), a consequence of its restricted number of GCL entries compared to the single sparse method (66.4%) as shown in Figure 9(a). This decrease is primarily due to scheduling conflicts, where a high volume of frames rapidly exhausts the limited GCL entries.

Based on the above results and discussions, we have the following findings on schedulability optimization.

Finding 2. *Schedulability optimization is highly context-dependent. There doesn't exist a globally optimal scheduling algorithm (neither exact nor heuristic algorithm). In general,*

- Heuristic algorithms demonstrate higher efficiency in large-scale systems, especially under complex models (e.g., with JRS and window-based model); exact solutions show better schedulability in small-scale systems.
- Heuristic algorithms may suffer from low schedulability under certain scenarios, e.g., with tight deadline (LS-TB and LS-PL), inharmonic periodicity (I-OMT), and traffic with cyclic dependencies (LS-PL).

7.2 Scalability

In this section, we compare the scalability of 17 scheduling methods in terms of runtime and memory consumption under different settings. Runtime and memory consumption are both critical performance metrics that evaluate how well the scheduling algorithm will scale in practice [72].

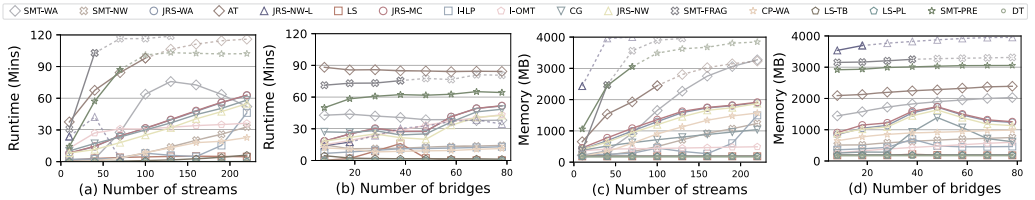
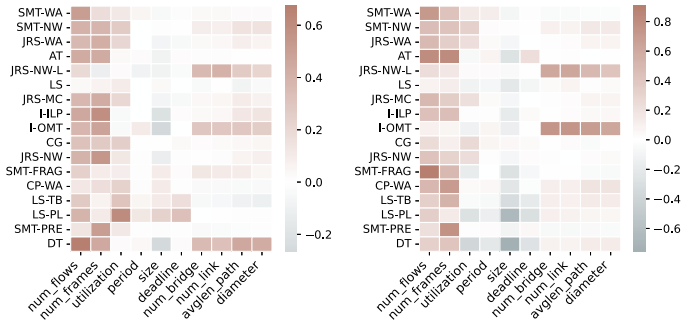


Fig. 10. Runtime and memory consumption comparisons under varied stream set and network settings.



(a) Runtime correlation (b) Memory correlation

Fig. 11. Runtime and memory consumption correlation in diverse scenarios.

7.2.1 Setup. In our experiments, the runtime of a scheduling algorithm consists of the pre-processing time (filtering the invalid solution space), the constraint adding time, and the problem solving time. If a scheduling method follows an objective function, we only measure its runtime of determining a feasible solution, rather than the optimal one to avoid any unfair comparison. For the memory consumption, we track the maximum memory usage for each approach, setting a 4GB threshold to allocate enough RAM and avoid swap space use.

7.2.2 Results. Figure 10 displays how runtime and memory consumption vary with the increase in the number of streams and bridges. Figure 11 presents two separate analyses focusing on the correlation between algorithm runtime and various settings, as well as the correlation between memory consumption and various settings, respectively. The Pearson correlation coefficient is utilized to quantify the correlation in both analyses, indicating a positive correlation through red color and a negative correlation through blue color. A high absolute correlation for a method signifies its sensitivity to that particular factor, implying that targeted scalability enhancements can be explored for that factor.

7.2.3 Discussion. Based on the above experimental results, we have two key findings on how experimental settings and algorithm design affect the runtime and memory usage.

Overall trend. From Figure 10(a)-(b), as the number of streams increases, we observe a significant rise in both the runtime and memory consumption for most methods. Specifically, the average runtime of all methods increases from 9.3 minutes under 10 streams to 48.6 minutes under 220 streams. Likewise, the average memory consumption increases from 476 MB under 10 streams to 1800 MB under 220 streams. Interestingly, adding more bridges to the network has a limited effect on the runtime. Overall, as shown in Figure 10(c), the runtime of most methods slightly increases from 23.6 minutes with 8 bridges to 34.1 minutes with 88 bridges. Among them, FR-based methods

only show a modest increase from 25.4 to 29.2 minutes, but JRS-based methods show a more substantial rise from 19.2 minutes to 41.8 minutes.

Regarding the memory consumption as shown in Figure 10(d), it remains relatively steady for FR-based methods with a slightly average increase of 183 MB when the number of bridges is increased from 8 to 88. As an exception, JRS-based methods peak at an average of 2070 MB with 48 bridges before dropping. These complex trends of memory and runtime along with the increased network scale may be due to compound factors. For example, a larger network can extend routing paths, thereby requiring more scheduling effort, but simultaneously reducing traffic density to lower the chance of collisions. These observations suggest that a larger network size does not necessarily result in a proportionally increased problem size, such as an increase in the number of decision variables or constraints.

Finding 3. *The increased workload poses a significant challenge to TSN scheduling, whereas the increased network scale does not show proportional impact* on the scalability.

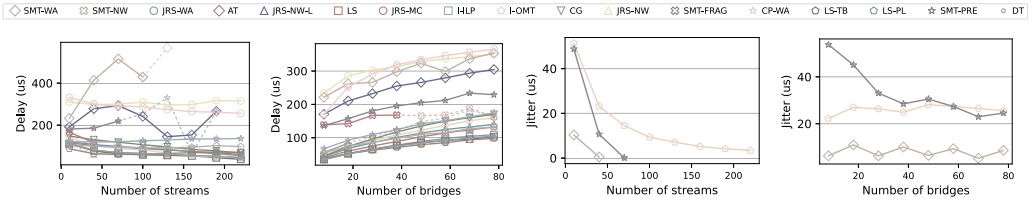
Individual algorithms. We also explore potential scalability patterns and bottlenecks for individual methods by analyzing their runtime and memory consumption correlation. As shown in Figure 11, a common trend observed in all methods is a positive correlation between runtime and memory consumption with the number of flows, frames, and utilization. However, distinct patterns can also be observed for individual methods. For instance, the runtime of I-OMT exhibits higher sensitivity to changes in the number of flows (correlation coefficient of 0.48) than its memory consumption (correlation coefficient of 0.10). In contrast, SMT-FRAG demonstrates a more significant increase in memory consumption when the number of flows is increased (correlation coefficient of 0.90) compared to its runtime (correlation coefficient of 0.24).

7.3 Schedule Quality

In addition to the schedulability and scalability metrics, in this section, we also evaluate the solution quality of the 17 methods in terms of GCL length, end-to-end delay and jitter, link utilization, and queue utilization. The delay and jitter are calculated from the packet-level TSN simulator we implemented in Python.

7.3.1 Setup. In our experiment, we define the GCL length as the peak value, i.e., the number of GCL entries for the port with the maximum value across the entire network. The average end-to-end delay is calculated for all streams, denoted from the moment when its first bit leaves the talker until the last bit is received at the listener. Similarly, we define the jitter as the difference between the maximum and minimum delays for each frame in individual streams, then average these values across all streams. Moreover, the link utilization is computed as the highest observed ratio between the allocated window size and the network cycle across all links. Lastly, we analyze the queue utilization, identifying the maximum required queues across all links.

Because individual methods can yield different feasible solutions due to their inherent schedulability characteristics, it is challenging to compare the schedule quality without any bias. In our evaluation results, we find that the GCL length and link utilization are highly sensitive to the schedulability of the method. To tackle this problem, we define a rank-based metric that aims to generate a more representative comparison utilizing all feasible solutions. The key idea is to leverage the obtained performance of each method and the generality of each problem instance to reduce the bias. Let $\mathbb{M}$ be the set of all methods, and for any given problem instance, $\mathbb{S}$ be the subset of methods that produce feasible results. The rank, $R(M)$, for a method $M \in \mathbb{S}$ is defined as $R(M) = (|\mathbb{M}| - |\mathbb{S}|) + r(M)$, where $r(M)$ is the relative rank of method M within $\mathbb{S}$ based on its performance, $|\mathbb{M}| - |\mathbb{S}|$ is the number of methods that fail to produce a feasible result.



(a) Delay vs. # of streams (b) Delay vs. # of bridges (c) Jitter vs. # of streams (d) Jitter vs. # of bridges

Fig. 12. Delay and jitter vs. # of streams and bridges.

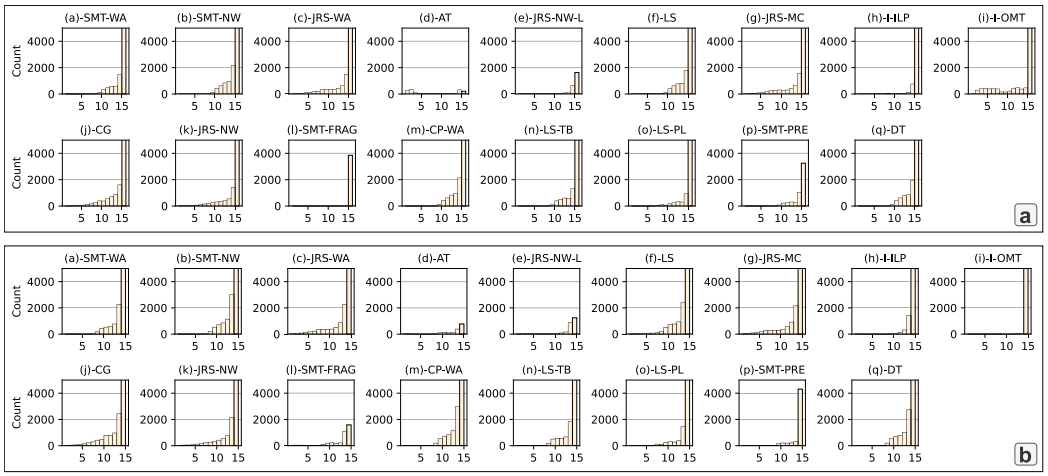


Fig. 13. (a) Rank distribution of the maximum GCL length among all the problem instances. Lower rank indicates better performance (i.e., shorter GCL length); (b) Rank distribution of the maximum link utilization among all the solvable problem instances for each method. Lower rank indicates better performance (i.e., lower link utilization).

Methods with identical performance will receive the same rank. The lower rank value indicates better performance.

7.3.2 Results. Figure 13(a) illustrates the rank distribution of the maximum GCL length for each method. The x-axis represents $R(M)$, and the y-axis denotes the frequency of each rank occurring in their feasible results. Figure 12 shows the average end-to-end delay and jitter values across each method, which vary based on the stream set and network size. Figure 13(b) displays the rank distribution of the peak link utilization of each method.

7.3.3 Discussion. Based on these results, we now discuss the schedule quality of each method. **GCL length.** First of all, we find that methods based on the window-based model require fewer GCL entries, resulting in shorter GCL lengths. For instance, as shown in Figure 13(a), the window-based method AT mostly produces solutions with a low rank (≤ 5), reflecting its strict constraint on the maximum number of windows per link. Another window-based method I-OMT aims to minimize the number of GCL entries, and also shows a relatively low rank compared with others. Secondly, we also observe that the JRS-based methods tend to have a lower rank, especially compared to FRS-based methods, which rarely have instances with a rank of ≤ 8. This is because the routing decisions help balance the workload across the network, avoiding having high-volume

traffic on a single link, which could lead to a large number of windows. In addition, beyond rank distribution, our raw data shows that many frame-based methods, without optimizing GCL length, result in over 2000 entries in some instances. This exceeds the usual maximum GCL length allowed for TSN bridges, which is between 8 and 1024 [69].

End-to-end delay and jitter. For the overall trend, Figure 12(b) shows that the growing network scale significantly increases the e2e delay from an average value of 92 μs to 192 μs when the number of bridges is increased from 10 to 80, but the stream set scale, as shown in Figure 12(a), has limited impact. Additionally, our results also confirm that the no-wait model can significantly reduce the delay as expected compared with the window-based model, whereas the JRS model incurs a higher delay than the FR model as expected. For the jitter, interestingly, Figure 12(c)(d) shows a decreasing jitter when more streams or bridges are added to the network. We believe this is due to the fact that a larger workload and network scale require tighter window size to ensure schedulability, leading to a more deterministic schedule.

Link utilization. We find that the JRS model outperforms the FR model in managing link utilization. In Figure 13(b), JRS-based methods (e.g., JRS-NW, JRS-WA, JRS-MC, and CG) exhibit lower link utilization compared to FR methods due to their inherent load-balancing capacity. In addition, the window-based method I-OMT tends to overcommit resources, as it frequently performs the highest rank in link utilization compared to other methods.

Queue utilization. We also compare the queue utilization of each method. Our results find that methods like JRS-WA, JRS-MC, and SMT-PRE, which are based on the unrestricted queuing model, often require a large number of allocated queues to ensure schedule correctness.[4] This often surpasses the maximum number of available queues in our experimental settings. On the other hand, the explicit queuing model and no-wait model have lower queue utilization because of their explicit queuing constraints.

Finding 4. *Selecting an appropriate scheduling method is crucial for achieving the desired schedule quality, as performance varies across methods. In general,*

- **Minimizing GCL length:** Window-based methods (e.g., AT, I-OMT) tend to produce fewer GCL entries due to their strict window constraints. However, they may use more resources in some cases, which can cause higher link or queue utilization.
- **Minimizing e2e delay:** The no-wait model generally leads to shorter delays but may not be as good at keeping the GCL length small. Besides, the JRS model may increase delay but balances traffic load across multiple paths.
- **Maximizing link utilization:** Methods using JRS (e.g., JRS-WA, JRS-MC, CG) achieve lower link utilization benefiting from load balancing. Using the shortest path routing might allocate more unused bandwidth, leading to higher utilization.
- **Minimizing queue usage:** Methods with unrestricted queuing (e.g., JRS-WA, JRS-MC, SMT-PRE) require more queues when map to TAS queuing model. If queue availability is limited, using the explicit queuing or no-wait model is often more suitable.

8 Takeaway Lessons

We now highlight several key findings from our experiments and how they can guide real-world TSN deployment.

Method Selection. Our experiments identified that selecting an optimal scheduling method in a TSN network is challenging, and there is no single method that fits all scenarios. Based on our

[4]Here we assign each stream to a dedicated queue at each hop for unrestricted queuing, which follows a common practice.

observations, we offer two suggestions: (1) If a specific optimization objective is in place, Finding 2 and Finding 4 can be used to guide algorithm and model selections. (2) If schedulability is the only requirement without any specific optimization consideration, it is recommended to follow a "simple-to-complex" principle when selecting a scheduling model to minimize the deployment costs.

Traffic Adaptation. While selecting the appropriate scheduling method(s) can improve schedulability, it remains difficult to ensure schedulability in all scenarios. For example, with more than 200 streams, all scheduling methods show low schedulability (less than 50%). When a valid schedule cannot be found by any scheduling method, we recommend the following traffic adaptation based on our observations: (1) Reducing the stream set size. This can be done by multiplexing traffic streams into fewer higher rate traffic streams or changing some less critical streams from TT-traffic to AVB or best-effort traffic. (2) Regulating the traffic period to be harmonic. This can be achieved by rounding down the periods to the nearest common divisor. (3) Reducing the payload size. This can be done by using more efficient encoding methods or changing the upper-layer protocols.

Considering Hardware Restriction. Our experiment shows that certain scheduling methods can produce GCLs with over 2,000 entries on a single bridge port, exceeding the typical hardware limits of 1,024 entries [69]. Similarly, methods using unrestricted queuing (e.g., JRS-WA) may require more queues than are physically available. Overlooking these can lead to infeasible schedules in real TSN devices.

9 Threats to Validity

To enhance the applicability of the outcomes in this study, we acknowledge several limitations in the experiments.

9.1 Model/Algorithm Comparison

The primary goal of this study is to provide experimental evaluations for the existing 17 representative TAS-based scheduling methods under various scenarios. The discussions on the model/algorithm comparisons are based on the observations from the evaluation results of individual methods under practical experimental settings (e.g., timeout limit). Providing a thorough independent model/algorithm comparison requires a completely different experiment design to isolate model, algorithm, and implementation, which is beyond the scope of this study.

In addition, since TSN research has been explosive in recent years, we cannot include all the methods in comparison. For example, only two window-based methods [46, 69] are considered, and this is not sufficient to conclude the performance of the window-based model in general. Furthermore, the experimental results of certain individual methods may not be sufficient to represent the performance of the model/algorithm they employ. For example, AT and SMT-PREP are only designed as proof-of-concept without the objective of improving the network performance. Similarly, the efficiency of exact solutions may be improved using incremental scheduling or decomposition approaches [34].

9.2 Individual Method Comparison

Although we employ well-designed experimental setups to ensure fairness, potential issues may still exist.

Additional parameter settings. For the methods that require additional parameters (e.g., max number of windows for AT), we set those parameters using the same settings as in the original papers. However, the performance of individual methods may be further improved through

fine-tuning the parameters, especially for the methods that are sensitive to certain parameter settings, e.g., windows/fragmentations/preemptions setup in [47, 69, 110].

Implementation. For the implementation of each method, we employ the same tools (e.g., selected solver) and follow the settings (e.g., constraints) in the original papers for fairness. However, we identify specific issues that could potentially limit the performance of certain methods: (1) The solver selection and problem formulation significantly affect results. For example, the observed low performance of JRS-NW-L may be attributed to the low efficiency of the Cplex ILP solver in addressing logical constraints. Additionally, our analysis indicates that ILP formulations are generally more efficient than SMT if multiple CPU cores are employed. (2) Some JRS methods (e.g., JRS-WA and I-ILP) spend more time adding constraints on variables rather than searching for solutions, especially with large problem instances.

9.3 Testbed Validation

There are also some potential issues on our testbed validation that may affect the practicality of the results.

Customized Simulator. Our simulator is purpose-built to capture TAS gating mechanisms and model scheduling behavior at a per-slot granularity. This approach ensures full control over implementation details and allows straightforward integration of testbed-derived delay measurements. However, validating a custom simulator can be more challenging than relying on well-established frameworks (e.g., OMNeT++ or NeSTiNg). To validate our simulator, we compared measured end-to-end delays from our testbed to the worst-case delays generated by the simulator. While preliminary results show consistency for the tested scenarios, some overlooked modeling nuances may still remain.

Testbed Calibration. To obtain accurate delay calibration, we measure delays at the MAC layer using hardware timestamping functions on both ESs and TSN switches. Recent studies [14] show that additional system-level jitter (e.g., due to OS or driver activities) can cause sporadic delay spikes. While we have eliminated many sources of jitter in our testbed environment, we acknowledge that deeper system-level optimizations (e.g., low-latency kernels, CPU isolation) fall outside the scope of this study.

Simulation/Testbed Cross-Validation. The delay results measured on the testbed did not exceed the simulated worst-case delays. However, because our testbed scale (8 bridges, 8 ESs) is relatively small, we acknowledge that larger-scale or more diverse scenarios may reveal additional insights, which will be left as our future work.

10 Open Issues

10.1 Fair Performance Evaluation

Through our extensive studies on the 17 TSN scheduling methods, we find that the experimental setting has a significant impact on the evaluation results. Thus we share the following takeaway lessons on fair performance evaluation.

Parameter settings. For the sake of fair comparison, we propose two possible ways to avoid bias. (1) Employ extensive experimental settings to include a broader range of stream set and network settings, and resource constraints. In this way, we can better understand the overall performance of a method and thus improve the fidelity and applicability of the evaluation. (2) If the computational resources are limited for performing extensive experiments, an alternative way is to select representative experiment settings based on real-world industrial scenarios or from standards and profiles. For instance, [1, 26] offer realistic use cases that can serve as common

evaluation scenarios. However, given the early stage of TSN-based research, the availability of real-world industrial scenarios and standardized profiles is still limited.

Evaluation metrics. Another key takeaway is that evaluation metrics can introduce bias. For example, we observed inconsistencies between the SR and SA metrics in our experiments. To reduce bias, we provide the following two suggestions. (1) Use multi-dimensional metrics to assess the algorithm performance, and ensure that these metrics are based on statistically significant data rather than limited or skewed datasets. (2) Since different methods may not produce the same known results (i.e., either schedulable or unschedulable) for the given problem instances, it is important to design metrics that are robust to these unknowns, leading to more accurate evaluation results (e.g., using pairwise or rank-based comparisons metrics).

Experiment description. Another takeaway lesson is to provide a detailed description of the experimental settings used in the evaluation. The absence of such information may lead to inconsistent results for researchers who would like to replicate the method. Below we summarize some experimental settings that have a significant impact on the evaluation results and need to be explained in detail. (1) **Model assumption**. Instead of simply stating that "we compare with [22]", it is better to clarify that "we adapt [22] to a partially schedulable model that incorporates frame-isolation constraints. The Z3 solver is utilized with default configuration and without objective function." (2) **Stream set and network specification**. It is crucial to present the detailed specifications, e.g., the stream period (deadline) range, stream payload, and how the problem instances are sampled from specifications. (3) **Evaluation metrics**. It is important to specify how the evaluation metrics are measured. For example, is the delay measured starting from the time when the frame leaves the ES or the time when it arrives at the first bridge? And, is the jitter calculated as the standard variance in delay across all frames or the maximum difference in delay between any two frames?

10.2 Algorithm Design

We also provide some insights and recommendations to guide future research on TAS-based method scheduling design.

Real-world constraints. In our testbed validation, we identify several issues that prevent existing methods from ensuring e2e delay due to the ignorance of some practice constraints. (1) Co-scheduling of data streams and synchronization messages. Collisions between TT traffic and PTP messages can occur and cause synchronization error out of bound, resulting in network failure or deadline miss of TT traffic. This is due to the fact that a max sync error is included in most TSN network modeling. However, if synchronization cannot be achieved in the pre-defined period due to collisions, a sync error will become larger than the max error during runtime. (2) ES may impose stricter constraints than bridges due to their limited network processing capability. For instance, we need to insert an inter-frame distance (around 50 μs) between TT frames to maintain the packet order on the ESs, which is much larger than that on the bridges. This requirement on ES is overlooked by most existing methods. (3) TSN bridge may be subject to a specific window size bound in GCL; however, only a few methods consider this constraint by adding a granularity variable to their models. If these factors are overlooked during the schedule generation, it may lead to errors when directly deploying them to a real-world testbed. Hence, we suggest including these real-world constraints in future studies to improve the practicality of the proposed scheduling methods.

Performance optimization for specific scenarios. As we point out in the findings described in Section 7, it is important to select the right model and scheduling method for performance optimization under specific scenarios. Below, we summarize the pros and cons of certain models

and methods observed from our experiments. First of all, simple models (e.g., no-wait model) or heuristic approaches (e.g., the list scheduler) can achieve better performance for stream sets with large workload. Second, methods based on the JRS model can be counter-effective in large-scale network settings compared with the FR model due to its low efficiency. Third, enlarging the search space on the ES side (fully schedulable/partially schedulable) could be more effective than on the bridge side (no-wait/wait-allowed). Finally, the no-wait model is preferred if the number of queues is restricted.

Automatic and intelligent model/method selection. Given all the challenges of performance optimization in TSN scheduling as discussed above, it would be helpful to develop an easy-to-use toolkit that is able to perform intelligent model/method selection and automatic schedule generation. Such a tool can significantly reduce the complexity of the scheduling method design for TSN networks.

11 Conclusion and Future Work

The growing R&D interests in TSN aim to achieve ultra-low latency and deterministic communications over switched Ethernet networks. This article examines 17 state-of-the-art real-time scheduling methods based on TAS and establishes a benchmark for performance evaluation using various performance metrics. Comprehensive experiments are conducted using both high-fidelity simulator and real-world testbed to compare these algorithms, highlighting their strengths and weaknesses under various scenarios. This article aims to assist researchers in identifying the current state and open problems in TSN scheduling algorithm design and implementation, offering insights toward future TSN research and development.

For future work, we suggest the following research directions to further advance the field of TSN real-time scheduling. First, the evaluation results suggest that combining heuristic and exact algorithms offers a beneficial tradeoff between schedulability and scalability. This could be a promising research direction, particularly if enhancements like a bounded schedulability ratio or bounded runtime limits are incorporated in future studies. Second, most existing scheduling methods focus on static scenarios, while dynamic scenarios are more common in real-world applications. Future research should investigate dynamic scheduling methods that can adapt to dynamic network conditions and traffic patterns. Last but not least, there is an ever-growing need to incorporate fault tolerance requirements, mixed-criticality task sets, heterogeneous platforms as well as high-level protocols like DDS and OPC-UA, into TSN-enabled applications and thus requires new design methods for TSN real-time scheduling.

References

[1] Wolfgang Fischer, Joseph Gelish, and Michael Hegarty. 2021. Aerospace TSN use cases, traffic types, and requirements. https://www.ieee802.org/1/files/public/docs2021/dp-Jabbar-et-al-Aerospace-Use-Cases-0321-v06.pdf

[2] IEEE. 2010. IEEE Standard for Local and metropolitan area networks - Virtual Bridged Local Area Networks Amendment 12: Forwarding and Queuing Enhancements for Time-Sensitive Streams. *IEEE Std 802.1Qav-2009* (2010), 1–72. https://doi.org/10.1109/IEEESTD.2010.8684664

[3] IEEE. 2016. IEEE Standard for Local and metropolitan area networks - Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic. *IEEE Std 802.1Qbv-2015* (2016), 1–57. https://doi.org/10.1109/IEEESTD.2016.8613095

[4] IEEE. 2018. IEEE standard for local and metropolitan area networks-bridges and bridged networks-amendment 31: Stream reservation protocol (SRP) enhancements and performance improvements. *IEEE Std 802.1 Qcc-2018* (2018).

[5] IEEE. 2020. IEEE Standard for Local and Metropolitan Area Networks -Bridges and Bridged Networks - Amendment 34: Asynchronous Traffic Shaping. *IEEE Std 802.1Qcr-2020* (2020), 1–151. https://doi.org/10.1109/IEEESTD.2020.9253013

[6] IEEE. 2020. IEEE Standard for Local and Metropolitan Area Networks -Timing and Synchronization for Time-Sensitive Applications. *IEEE Std 802.1AS-2020* (2020), 1–421. https://doi.org/10.1109/IEEESTD.2020.9121845

[7] Toni Adame, Marc Carrascosa-Zamacois, and Boris Bellalta. 2021. Time-sensitive networking in IEEE 802.11 be: On the way to low-latency WiFi 7. *Sensors* 21, 15 (2021), 4954.

[8] Abdullah Alnajim, Seyedmohammad Salehi, and Chien-Chung Shen. 2019. Incremental path-selection and scheduling for time-sensitive networks. In *Proceedings of the 2019 IEEE Global Communications Conference (GLOBECOM'19)*. IEEE, 1–6.

[9] Anna Arestova, Kai-Steffen J. Hielscher, and Reinhard German. 2023. Optimization of bandwidth utilization and gate control list configuration in 802.1 Qbv networks. *IEEE Access* 11 (2023), 115076–115090.

[10] Ayman A. Atallah, Ghaith Bany Hamad, and Otmane Ait Mohamed. 2019. Routing and scheduling of time-triggered traffic in time-sensitive networks. *IEEE Transactions on Industrial Informatics* 16, 7 (2019), 4525–4534.

[11] Mohammadreza Barzegaran, Niklas Reusch, Luxi Zhao, Silviu S. Craciunas, and Paul Pop. 2022. Real-time traffic guarantees in heterogeneous time-sensitive networks. In *Proceedings of the 30th International Conference on Real-Time Networks and Systems*. 46–57.

[12] Mohammadreza Barzegaran, Bahram Zarrin, and Paul Pop. 2020. Quality-of-control-aware scheduling of communication in TSN-based fog computing platforms using constraint programming. In *Proceedings of the 2nd Workshop on Fog Computing and the IoT (Fog-IoT'20)*.

[13] Lucia Lo Bello and Wilfried Steiner. 2019. A perspective on IEEE time-sensitive networking for industrial communication and automation systems. *Proceedings of the IEEE* 107, 6 (2019), 1094–1120.

[14] Marcin Bosk, Filip Rezabek, Kilian Holzinger, Angela Gonzalez Marino, Abdoul Aziz Kane, Francesc Fons, Jörg Ott, and Georg Carle. 2022. Methodology and infrastructure for TSN-based reproducible network experiments. *IEEE Access* 10 (2022), 109203–109239.

[15] Dietmar Bruckner, Richard Blair, M. Stanica, A. Ademaj, W. Skeffington, D. Kutscher, S. Schriegel, R. Wilmes, K. Wachswender, L. Leurs, et al. 2018. OPC UA TSN a new solution for industrial communication. *Whitepaper. Shaper Group* 168 (2018).

[16] Daniel Bujosa, Mohammad Ashjaei, Alessandro V. Papadopoulos, Thomas Nolte, and Julián Proenza. 2022. HERMES: Heuristic multi-queue scheduler for TSN time-triggered traffic with zero reception jitter capabilities. In *Proceedings of the 30th International Conference on Real-Time Networks and Systems*. 70–80.

[17] Stephen F. Bush. 2022. Toward efficient time-sensitive network scheduling. *IEEE Transactions on Aerospace and Electronic Systems* 58, 3 (2022), 1830–1842.

[18] Hamza Chahed and Andreas Kassler. 2023. TSN network scheduling–challenges and approaches. *Network* 3, 4 (2023), 585–624.

[19] Richard Cochran and Cristian Marinescu. 2010. Design and implementation of a PTP clock infrastructure for the Linux kernel. In *Proceedings of the 2010 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*. IEEE, 116–121.

[20] Silviu S. Craciunas and Ramon Serna Oliver. 2021. Out-of-sync schedule robustness for time-sensitive networks. In *Proceedings of the 2021 17th IEEE International Conference on Factory Communication Systems (WFCS'21)*. IEEE, 75–82.

[21] Silviu S. Craciunas, R. Serna Oliver, and T. Ag. 2017. An overview of scheduling mechanisms for time-sensitive networks. *Proceedings of the Real-time summer school LÉcole dÉté Temps Réel (ETR)* (2017), 1551–3203.

[22] Silviu S. Craciunas, Ramon Serna Oliver, Martin Chmelík, and Wilfried Steiner. 2016. Scheduling real-time communication in IEEE 802.1 Qbv time sensitive networks. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*. 183–192.

[23] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *Proceedings of the 14th International Conference of Tools and Algorithms for the Construction and Analysis of Systems (TACAS), 2008*. Springer, 337–340.

[24] Libing Deng, Guoqi Xie, Hong Liu, Yunbo Han, Renfa Li, and Keqin Li. 2022. A survey of real-time ethernet modeling and design methodologies: From AVB to TSN. *ACM Computing Surveys (CSUR)* 55, 2 (2022), 1–36.

[25] Radu Dobrin, Nitin Desai, and Sasikumar Punnekkat. 2019. On fault-tolerant scheduling of time sensitive networks. In *Proceedings of the 4th International Workshop on Security and Dependability of Critical Embedded Real-Time Systems (CERTS'19)*.

[26] Rudy Belliardi, Josef Dorr, Thomas Enzinger, Florian Essler, János Farkas, Mark Hantel, Maximilian Riegel, Marius-Petru Stanica, Guenter Steindl, Reiner Wamßer, et al. 2018. Use cases iec/IEEE 60802 v1. 3. *IEEE 60802* (2018), V1.

[27] Frank Dürr and Naresh Ganesh Nayak. 2016. No-wait packet scheduling for IEEE time-sensitive networks (TSN). In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*. 203–212.

[28] Doğanalp Ergenç, Cornelia Brülhart, Jens Neumann, Leo Krüger, and Mathias Fischer. 2021. On the security of IEEE 802.1 time-sensitive networking. In *Proceedings of the 2021 IEEE International Conference on Communications Workshops (ICC Workshops'21)*. IEEE, 1–6.

[29] Jonathan Falk, Frank Dürr, and Kurt Rothermel. 2018. Exploring practical limitations of joint routing and scheduling for TSN with ILP. In *Proceedings of the 2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'18)*. IEEE, 136–146.

[30] Jonathan Falk, Frank Dürr, and Kurt Rothermel. 2020. Time-triggered traffic planning for data networks with conflict graphs. In *Proceedings of the 2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'20)*. IEEE, 124–136.

[31] Tommaso Fedullo, Alberto Morato, Federico Tramarin, Luigi Rovati, and Stefano Vitturi. 2022. A comprehensive review on time sensitive networks with a special focus on its applicability to industrial smart and distributed measurement systems. *Sensors* 22, 4 (2022), 1638.

[32] Zhiwei Feng, Qingxu Deng, Mingyang Cai, and Jinghua Li. 2022. E?icient reservation-based fault-tolerant scheduling for IEEE 802.1Qbv time-sensitive networking. *Journal of Systems Architecture* 123 (2022), 102381. https://doi.org/10.1016/j.sysarc.2021.102381

[33] Zhiwei Feng, Zonghua Gu, Haichuan Yu, Qingxu Deng, and Linwei Niu. 2022. Online rerouting and rescheduling of time-triggered flows for fault tolerance in time-sensitive networking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 11 (2022), 4253–4264.

[34] Anaïs Finzi and Ramon Serna Oliver. 2022. General framework for routing, scheduling and formal timing analysis in deterministic time-aware networks. In *Proceedings of the 34th Euromicro Conference on Real-Time Systems (ECRTS'22)*.

[35] Christoph Gärtner, Amr Rizk, Boris Koldehofe, René Guillaume, Ralf Kundel, and Ralf Steinmetz. 2022. On the incremental reconfiguration of time-sensitive networks at runtime. In *Proceedings of the 2022 IFIP Networking Conference (IFIP Networking'22)*. IEEE, 1–9.

[36] Gurobi Optimization, LLC. 2021. Gurobi optimizer reference manual. https://www.gurobi.com/documentation/9.1/refman/index.html. Accessed: 2025-05-30.

[37] Wenxuan Han, Yanjue Li, and Changchuan Yin. 2022. A traffic scheduling algorithm combined with ingress shaping in TSN. In *Proceedings of the 2022 14th International Conference on Wireless Communications and Signal Processing (WCSP'22)*. IEEE, 586–591.

[38] Xiaowu He, Xiangwen Zhuge, Fan Dang, Wang Xu, and Zheng Yang. 2023. Deep-Scheduler: Enabling flow-aware scheduling in time-sensitive networking. In *Proceedings of the IEEE INFOCOM*.

[39] David Hellmanns, Jonathan Falk, Alexander Glavackij, René Hummen, Stephan Kehrer, and Frank Dürr. 2020. On the performance of stream-based, class-based time-aware shaping and frame preemption in TSN. In *Proceedings of the IEEE International Conference on Industrial Technology*. IEEE, 298–303.

[40] David Hellmanns, Alexander Glavackij, Jonthan Falk, René Hummen, Stephan Kehrer, and Frank Dürr. 2020. Scaling TSN scheduling for factory automation networks. In *Proceedings of the 2020 16th IEEE International Conference on Factory Communication Systems (WFCS'20)*. IEEE, 1–8.

[41] David Hellmanns, Lucas Haug, Moritz Hildebrand, Frank Dürr, Stephan Kehrer, and René Hummen. 2021. How to optimize joint routing and scheduling models for TSN using integer linear programming. In *Proceedings of the 29th International Conference on Real-Time Networks and Systems*. 100–111.

[42] Xinyi Hong, Yuhao Xi, and Peng Liu. 2024. Resource-aware online traffic scheduling for time-sensitive networking. *IEEE Transactions on Industrial Informatics* 20, 12 (2024), 14267–14276. https://doi.org/10.1109/TII.2024.3449988

[43] Bahar Houtan, Mohammad Ashjaei, Masoud Daneshtalab, Mikael Sjödin, and Saad Mubeen. 2021. Synthesising schedules to improve QoS of best-effort traffic in TSN networks. In *Proceedings of the 29th International Conference on Real-Time Networks and Systems*. 68–77.

[44] IEEE. 2016. IEEE Standard for Local and metropolitan area networks - Bridges and Bridged Networks - Amendment 26: Frame Preemption. *IEEE Std 802.1Qbu-2016* (Amendment to IEEE Std 802.1Q-2014) (2016), 1–52. https://doi.org/10.1109/IEEESTD.2016.7553415

[45] IEEE. 2018. IEEE Standard for Local and Metropolitan Area Network -Bridges and Bridged Networks. *IEEE Std 802.1 Q-2018 (Revision of IEEE Std 802.1 Q-2014)* (2018), 1–1993.

[46] Xi Jin, Changqing Xia, Nan Guan, Chi Xu, Dong Li, Yue Yin, and Peng Zeng. 2020. Real-time scheduling of massive data in time sensitive networks with a limited number of schedule entries. *IEEE Access* 8 (2020), 6751–6767. https://doi.org/10.1109/ACCESS.2020.2964690

[47] Xi Jin, Changqing Xia, Nan Guan, and Peng Zeng. 2021. Joint algorithm of message fragmentation and no-wait scheduling for time-sensitive networks. *IEEE/CAA Journal of Automatica Sinica* 8, 2 (2021), 478–490.

[48] Yoohwa Kang, Sunwoo Lee, Songi Gwak, Taekyeong Kim, and Donghyeok An. 2021. Time-sensitive networking technologies for industrial automation in wireless communication systems. *Energies* 14, 15 (2021), 4497.

[49] Kate Keahey, Jason Anderson, Zhuo Zhen, Pierre Riteau, Paul Ruth, Dan Stanzione, Mert Cevik, Jacob Colleran, Haryadi S. Gunawi, Cody Hammock, et al. 2020. Lessons learned from the chameleon testbed. In *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC'20)*. 219–233.

[50] Stephan Kehrer, Oliver Kleineberg, and Donal Heffernan. 2014. A comparison of fault-tolerance concepts for IEEE 802.1 Time Sensitive Networks (TSN). In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA'14)*. IEEE, 1–8.

[51] Wazir Zada Khan, M. H. Rehman, Hussein Mohammed Zangoti, Muhammad Khalil Afzal, Nasrullah Armi, and Khaled Salah. 2020. Industrial internet of things: Recent advances, enabling technologies and open challenges. *Computers & Electrical Engineering* 81 (2020), 106522.

[52] Leon Kist and Philippe Buschmann. 2022. Survey on scheduling approaches in TSN. In *Proceedings of the Seminar Innovative Internet Technologies and Mobile Communications (IITM'22)*.

[53] Barbara Kitchenham. 2004. Procedures for performing systematic reviews. *Keele, UK, Keele University* 33, 2004 (2004), 1–26.

[54] Eleftherios Kyriakakis, Jens Sparsø, and Martin Schoeberl. 2018. Hardware assisted clock synchronization with the IEEE 1588-2008 precision time protocol. In *Proceedings of the 26th International Conference on Real-Time Networks and Systems*. 51–60.

[55] Zonghui Li, Hai Wan, Zaiyu Pang, Qiubo Chen, Yangdong Deng, Xibin Zhao, Yue Gao, Xiaoyu Song, and Ming Gu. 2019. An enhanced reconfiguration for deterministic transmission in time-triggered networks. *IEEE/ACM Transactions on Networking* 27, 3 (2019), 1124–1137.

[56] Rouhollah Mahfouzi, Amir Aminifar, Soheil Samii, Ahmed Rezine, Petru Eles, and Zebo Peng. 2018. Stability-aware integrated routing and scheduling for control applications in Ethernet networks. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 682–687.

[57] Tieu Long Mai, Nicolas Navet, and Jörn Migge. 2019. On the use of supervised machine learning for assessing schedulability: Application to Ethernet TSN. In *Proceedings of the 27th International Conference on Real-Time Networks and Systems*. 143–153.

[58] Lisa Maile, Kai-Steffen Hielscher, and Reinhard German. 2020. Network calculus results for TSN: An introduction. In *Proceedings of the 2020 Information Communication Technologies Conference (ICTC'20)*. IEEE, 131–140.

[59] Želimir Maletić, Milan Mlađen, and Miloš Ljubojević. 2023. A survey on the current state of time-sensitive networks standardization. In *Proceedings of the 2023 10th International Conference on Electrical, Electronic and Computing Engineering (IcETRAN'23)*. IEEE, 1–6.

[60] Alessandro Mascis and Dario Pacciarelli. 2002. Job-shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research* 143, 3 (2002), 498–517.

[61] Junhong Min, Woongsoo Kim, Jeongyeup Paek, and Ramesh Govindan. 2023. Effective routing and scheduling strategies for fault-tolerant timesensitive networking. *IEEE Internet of Things Journal* 11, 6 (2023), 11008–11020.

[62] Junhong Min, Yongjun Kim, Moonbeom Kim, Jeongyeup Paek, and Ramesh Govindan. 2023. Reinforcement learning based routing for time-aware shaper scheduling in time-sensitive networks. *Computer Networks* 235 (2023), 109983. https://doi.org/10.1016/j.comnet.2023.109983

[63] Anna Minaeva and Zdeněk Hanzálek. 2021. Survey on periodic scheduling for time-triggered hard real-time systems. *ACM Computing Surveys (CSUR)* 54, 1 (2021), 1–32.

[64] Gilson Miranda, Esteban Municio, Jetmir Haxhibeqiri, Jeroen Hoebeke, Ingrid Moerman, and Johann M. Marquez-Barja. 2023. Enabling time-sensitive network management over multi-domain wired/Wi-Fi networks. *IEEE Transactions on Network and Service Management* 20, 3 (2023), 2386–2399.

[65] Gilson Miranda, Esteban Municio, Jetmir Haxhibeqiri, Daniel F. Macedo, Jeroen Hoebeke, Ingrid Moerman, and Johann M. Marquez-Barja. 2022. Time-sensitive networking experimentation on open testbeds. In *Proceedings of the IEEE INFOCOM Workshops*. IEEE, 1–6.

[66] Morteza Mohaqeqi, Mitra Nasri, Yang Xu, Anton Cervin, and Karl-Erik Årzén. 2018. Optimal harmonic period assignment: Complexity results and approximation algorithms. *Real-Time Systems* 54 (2018), 830–860.

[67] Ahmed Nasrallah, Venkatraman Balasubramanian, Akhilesh Thyagaturu, Martin Reisslein, and Hesham ElBakoury. 2019. TSN algorithms for large scale networks: A survey and conceptual comparison. arXiv:1905.08478. Retrieved from https://arxiv.org/abs/1905.08478

[68] Ahmed Nasrallah, Akhilesh S. Thyagaturu, Ziyad Alharbi, Cuixiang Wang, Xing Shao, Martin Reisslein, and Hesham ElBakoury. 2018. Ultra-low latency (ULL) networks: The IEEE TSN and IETF DetNet standards and related 5G ULL research. *IEEE Communications Surveys & Tutorials* 21, 1 (2018), 88–145.

[69] Ramon Serna Oliver, Silviu S. Craciunas, and Wilfried Steiner. 2018. IEEE 802.1 Qbv gate control list synthesis using array theory encoding. In *Proceedings of the 2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'18)*. IEEE, 13–24.

[70] Maryam Pahlevan, Nadra Tabassam, and Roman Obermaisser. 2019. Heuristic list scheduler for time triggered traffic in time sensitive networks. *ACM Sigbed Review* 16, 1 (2019), 15–20.

[71] Zaiyu Pang, Xiao Huang, Zonghui Li, Sukun Zhang, Yanfen Xu, Hai Wan, and Xibin Zhao. 2020. Flow scheduling for conflict-free network updates in time-sensitive software-defined networks. *IEEE Transactions on Industrial Informatics* 17, 3 (2020), 1668–1678.

[72] Don Pannell. 2019. Choosing the right TSN tools to meet a bounded latency. *IEEE SA Ethernet & IP@ Automotive Technology Day* (2019).

[73] Pusik Park, Myeonghwan Son, Jeongdo Lee, and Jongho Yoon. 2021. Performance evaluation of the efficient precise time synchronization protocol for the redundant ring topology network. In *Proceedings of the 2021 IEEE/AIAA 40th Digital Avionics Systems Conference (DASC'21)*. 1–10.

[74] Gaetano Patti, Lucia Lo Bello, and Luca Leonardi. 2022. Deadline-aware online scheduling of TSN flows for automotive applications. *IEEE Transactions on Industrial Informatics* 19, 4 (2022), 5774–5784.

[75] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *The Journal of Machine Learning Research* 12 (2011), 2825–2830.

[76] Yifei Peng, Boxin Shi, Tigang Jiang, Xiaodong Tu, Du Xu, and Kun Hua. 2023. A survey on in-vehicle time-sensitive networking. *IEEE Internet of Things Journal* 10, 16 (2023), 14375–14396.

[77] Michael Lander Raagaard, Paul Pop, Marina Gutiérrez, and Wilfried Steiner. 2017. Runtime reconfiguration of time-sensitive networking (TSN) schedules for fog computing. In *Proceedings of the 2017 IEEE Fog World Congress (FWC'17)*. IEEE, 1–6.

[78] Niklas Reusch, Mohammadreza Barzegaran, Luxi Zhao, Silviu S. Craciunas, and Paul Pop. 2023. Configuration optimization for heterogeneous time-sensitive networks. *Real-Time Systems* 59, 4 (2023), 705–747.

[79] Niklas Reusch, Paul Pop, and Silviu S. Craciunas. 2020. Work-in-progress: Safe and secure configuration synthesis for TSN using constraint programming. In *2020 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 387–390.

[80] Adrien Roberty, Siwar Ben Hadj Said, Frederic Ridouard, Henri Bauer, and Annie Geniet. 2023. Reinforcement learning for time-aware shaping (IEEE 802.1 Qbv) in time-sensitive networks. In *Proceedings of the 2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation*. 1–4.

[81] Jorge Sasiain, David Franco, Asier Atutxa, Jasone Astorga, and Eduardo Jacob. 2025. Toward the integration and convergence between 5G and TSN technologies and architectures for industrial communications: A survey. *IEEE Communications Surveys & Tutorials* 27, 1 (2025), 259–321. https://doi.org/10.1109/COMST.2024.3422613

[82] Zenepe Satka, Mohammad Ashjaei, Hossein Fotouhi, Masoud Daneshtalab, Mikael Sjödin, and Saad Mubeen. 2023. A comprehensive systematic review of integration of time sensitive networking and 5G communication. *Journal of Systems Architecture* 138 (2023), 102852.

[83] Eike Schweissguth, Peter Danielis, Dirk Timmermann, Helge Parzyjegla, and Gero Mühl. 2017. ILP-based joint routing and scheduling for time-triggered networks. In *Proceedings of the 25th International Conference on Real-Time Networks and Systems*. 8–17.

[84] Eike Schweissguth, Helge Parzyjegla, Peter Danielis, Gero Mühl, Dirk Timmermann, Stefan Mehner, Oliver Hohlfeld, David Hellmanns, and Jonathan Falk. 2023. TSN scheduler benchmarking. In *Proceedings of the 2023 IEEE 19th International Conference on Factory Communication Systems*. 1–8.

[85] Eike Schweissguth, Dirk Timmermann, Helge Parzyjegla, Peter Danielis, and Gero Mühl. 2020. ILP-based routing and scheduling of multicast realtime traffic in time-sensitive networks. In *Proceedings of the 2020 IEEE 26th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'20)*. IEEE, 1–11.

[86] Youhwan Seol, Doyeon Hyeon, Junhong Min, Moonbeom Kim, and Jeongyeup Paek. 2021. Timely survey of timesensitive networking: Past and future directions. *IEEE Access* 9 (2021), 142506–142527. https://doi.org/10.1109/ACCESS.2021.3120769

[87] Luis Silva, Paulo Pedreiras, Pedro Fonseca, and Luis Almeida. 2019. On the adequacy of SDN and TSN for Industry 4.0. In *Proceedings of the 2019 IEEE 22nd International Symposium on Real-time Distributed Computing (ISORC'19)*. IEEE, 43–51.

[88] Emiliano Sisinni, Abusayeed Saifullah, Song Han, Ulf Jennehag, and Mikael Gidlund. 2018. Industrial internet of things: Challenges, opportunities, and directions. *IEEE Transactions on Industrial Informatics* 14, 11 (2018), 4724–4734.

[89] Tobias Striffler, Nicola Michailow, and Michael Bahr. 2019. Time-sensitive networking in 5th generation cellular networks-current state and open topics. In *Proceedings of the 2019 IEEE 2nd 5G World Forum (5GWF'19)*. IEEE, 547–552.

[90] Thomas Stüber, Manuel Eppler, Lukas Osswald, and Michael Menth. 2024. Performance comparison of offline scheduling algorithms for the time-aware shaper (TAS). *IEEE Transactions on Industrial Informatics* 20, 7 (2024), 9736–9748. https://doi.org/10.1109/TII.2024.3385503

[91] Thomas Stüber, Lukas Osswald, Steffen Lindner, and Michael Menth. 2023. A survey of scheduling algorithms for the time-aware shaper in time-sensitive networking (TSN). *IEEE Access* 11 (2023), 61192–61233.

[92] Wenjing Sun, Yuan Zou, Nan Guan, Xudong Zhang, Guodong Du, and Ya Wen. 2024. Graph attention network -based deep reinforcement learning scheduling framework for in-vehicle time-sensitive networking. *IEEE Transactions on Industrial Informatics* 20, 7 (2024), 9825–9836. https://doi.org/10.1109/TII.2024.3388669

[93] Ammad Ali Syed, Serkan Ayaz, Tim Leinmüller, and Madhu Chandra. 2021. Fault-tolerant dynamic scheduling and routing for TSN based in-vehicle networks. In *Proceedings of the 2021 IEEE Vehicular Networking Conference (VNC'21)*. IEEE, 72–75.

[94] Marian Ulbricht, Stefan Senk, Hosein K. Nazari, How-Hang Liu, Martin Reisslein, Giang T. Nguyen, and Frank H. P. Fitzek. 2023. Tsn-flextest: Flexible tsn measurement testbed. *IEEE Transactions on Network and Service Management* 21, 2 (2023), 1387–1402.

[95] Marek Vlk, Kateřina Brejchová, Zdeněk Hanzálek, and Siyu Tang. 2022. Large-scale periodic scheduling in timesensitive networks. *Computers & Operations Research* 137 (2022), 105512. https://doi.org/10.1016/j.cor.2021.105512

[96] Marek Vlk, Zdeněk Hanzálek, Kateřina Brejchová, Siyu Tang, Sushmit Bhattacharjee, and Songwei Fu. 2020. Enhancing schedulability and throughput of time-triggered traffic in IEEE 802.1 Qbv time-sensitive networks. *IEEE Transactions on Communications* 68, 11 (2020), 7023–7038.

[97] Marek Vlk, Zdeněk Hanzálek, and Siyu Tang. 2021. Constraint programming approaches to joint routing and scheduling in time-sensitive networks. *Computers & Industrial Engineering* 157 (2021), 107317. https://doi.org/10.1016/j.cie.2021.107317

[98] Jiachen Wang, Tianyu Zhang, Dawei Shen, Xiaobo Sharon Hu, and Song Han. 2022. HARP: Hierarchical resource partitioning in dynamic industrial wireless networks. In *Proceedings of the 2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS'22)*. IEEE, 1029–1039.

[99] Zitong Wang, Feng Luo, Yunpeng Li, Haotian Gan, and Lei Zhu. 2025. Schedulability analysis in time-sensitive networking: A systematic literature review. *Ad Hoc Networks* (2025), 103897.

[100] Liu Yang, Yifei Wei, F. Richard Yu, and Zhu Han. 2022. Joint routing and scheduling optimization in time-sensitive networks using graph-convolutional-network-based deep reinforcement learning. *IEEE Internet of Things Journal* 9, 23 (2022), 23981–23994.

[101] Mingwu Yao, Jiamu Liu, Jing Du, Dongqi Yan, Yanxi Zhang, Wei Liu, and Anthony Man-Cho So. 2023. A unified flow scheduling method for time sensitive networks. *Computer Networks* 233 (2023), 109847.

[102] Qinghan Yu, Hai Wan, Xibin Zhao, Yue Gao, and Ming Gu. 2019. Online scheduling for dynamic VM migration in multicast time-sensitive networks. *IEEE Transactions on Industrial Informatics* 16, 6 (2019), 3778–3788.

[103] Tianyu Zhang, Gang Wang, Chuanyu Xue, Jiachen Wang, Mark Nixon, and Song Han. 2024. Time-sensitive networking (TSN) for industrial automation: Current advances and future directions. *ACM Computing Surveys* 57, 2 (2024), 1–38.

[104] Yinghui Zhang, Jiamin Wu, Mingli Liu, and Aiping Tan. 2022. TSN-based routing and scheduling scheme for Industrial Internet of Things in underground mining. *Engineering Applications of Artificial Intelligence* 115 (2022), 105314. https://doi.org/10.1016/j.engappai.2022.105314

[105] Yanzhou Zhang, Qimin Xu, Shouliang Wang, Yingxiu Chen, Lei Xu, and Cailian Chen. 2022. Scalable no-wait scheduling with flow-aware model conversion in time-sensitive networking. In *Proceedings of the IEEE Global Communications Conference (GLOBECOM'22)*. IEEE, 413–418.

[106] Jiayi Zhao and Jing Cheng. 2024. The time-sensitive networking scheduling algorithm based on Q-learning. *International Journal of Advanced Network, Monitoring and Controls* 9, 1 (2024), 78–86. https://doi.org/10.2478/ijanmc2024-0008

[107] Luxi Zhao, Paul Pop, and Sebastian Steinhorst. 2022. Quantitative performance comparison of various traffic shapers in time-sensitive networking. *IEEE Transactions on Network and Service Management* 19, 3 (2022), 2899–2928.

[108] Boyang Zhou and Liang Cheng. 2023. TSN-VM: A real-time and distributed algorithm for scheduling-violation mitigation in time-sensitive networking. *Authorea Preprints* (2023).

[109] Yuanbin Zhou, Soheil Samii, Petru Eles, and Zebo Peng. 2021. ASIL-decomposition based routing and scheduling in safety-critical time-sensitive networking. In *Proceedings of the 2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS'21)*. IEEE, 184–195.

[110] Yuanbin Zhou, Soheil Samii, Petru Eles, and Zebo Peng. 2022. Time-triggered scheduling for time-sensitive networking with preemption. In *Proceedings of the 2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC'22)*. IEEE, 262–267.

Received 4 August 2024; revised 25 April 2025; accepted 6 May 2025