Real-Time Scheduling for 802.1Qbv Time-Sensitive Networking (TSN): A Systematic Review and Experimental Study

Chuanyu Xue*, Tianyu Zhang*, Yuanbin Zhou†, Mark Nixon‡, Andrew Loveless§, Song Han*
*School of Computing, University of Connecticut
†Singapore University of Technology and Design
‡Emerson Automation Solutions, §NASA Johnson Space Center

***Abstract*—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., 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 yet to be thoroughly evaluated, especially through experimental comparisons, to provide a systematical understanding of their performance in diverse application scenarios. In this paper, we fill this gap by presenting a systematic review and experimental study on existing TAS-based scheduling methods for TSN. We first categorize the system models employed in these works along with the specific problems they aim to solve, and outline the fundamental 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 TSN testbed, and compare their performance under both synthetic scenarios and real-life industrial use cases. Through these studies, we identify the limitations of individual scheduling methods and highlight several important findings. We expect this work will provide foundational knowledge and performance benchmarks needed for future studies on real-time TSN scheduling.**

I. 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 [1]–[3]. 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) [4], Asynchronous Traffic Shaper (ATS) [5], and Time-Aware Shaper (TAS) [6], 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 [7], making it a critical technology to support deterministic traffic in industrial applications.

TAS operates in a time-triggered scheduling fashion. It achieves deterministic communications by buffering and releasing traffic at specific time instances following a predetermined schedule. Specifically, as shown in Fig. 1, each egress port in a TSN switch (also called *bridge*) is equipped with a set of time-gated queues to buffer frames from each traffic flow. A scheduled gate mechanism is utilized to open or close the queues and control the transmission of frames according to a predefined Gate Control List (GCL). Each GCL includes a limited number of entries. Each entry provides the status of associated queues over a particular duration. The GCL repeats itself periodically, and the network-wide schedule is generated by the Centralized Network Configuration (CNC) and deployed on individual bridges. In addition to the scheduled gate, the priority filter utilizes a 3-bit Priority Code Point (PCP) field in the packet header to identify the stream priority, and directs incoming traffic to the appropriate egress queue(s) [6].

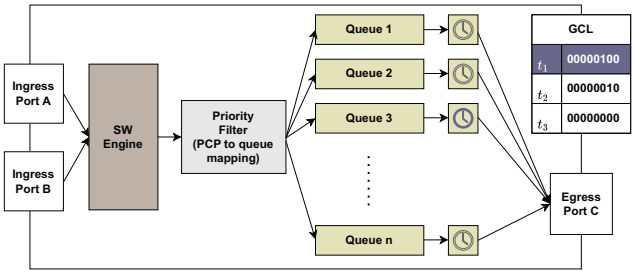

Fig. 1. An illustration of the Time-Aware Shaper (TAS) mechanism in a Time-Sensitive Networking (TSN) bridge.

Although the scheduling mechanism of TAS is clearly defined in the IEEE 802.1Qbv standard, the configuration of TAS, e.g., what to put in the GCL and how to assign queues for individual traffic at each hop, 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 [8]. 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 problems to be studied may vary significantly. This results in a large amount of efforts from both researchers and practitioners to study various system models and develop corresponding algorithms. These studies considerably enrich the literature, paving the way to improve TSN network performance.

2642-7346/24/$31.00 ©2024 IEEE
DOI 10.1109/RTAS61025.2024.00017

There have been several recent survey works on real-time scheduling in TSN networks (e.g., [8]–[14]). 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, [8], [15] provided comparisons among various TSN scheduling approaches, with [8] primarily focusing on TAS-based studies and [15] extending the comparisons to all TSN shapers. However, all the aforementioned works suffer from the following two significant limitations. First, they don't 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 paper 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., [16]–[31]). 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 consistency 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 the comprehensive experimental results, we are able to highlight a set of interesting observations and findings. In general, our study shows that there is no one-size-fits-all solution that can achieve dominating performance in all scenarios; individual scheduling method/model may demonstrate superiority under certain setting(s). Furthermore, we demonstrate that diverse experimental settings complicate the fair evaluation of scheduling methods without introducing bias, which can make conclusions from previous studies only valid under specific settings. We expect that our findings will help the community understand better the benefits and drawbacks of existing TSN scheduling methods and provide valuable insights for the development of future TSN scheduling methods.[1]
In summary, this work makes the following contributions:

1) We provide a comprehensive review of various TSN system models and categorize 17 representative TAS-based scheduling methods accordingly.

[1]For detailed experimental results and open source code, please refer to our technical report [32] and Github repository [33], respectively.

2) We establish a 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 scheduling methods under comprehensive 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 methods.

II. 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 III.

A. Network Models

A TSN network consists of two types of devices: bridges and end stations (ES). A bridge can forward Ethernet frames for one or multiple TSN streams according to a schedule constructed based on the IEEE 802.1Q standard [34]. 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 five attributes:

• **Propagation delay** refers to the time duration of a signal transmitting on the physical link. It is solely dependent on the length of the cable and the type of the physical media used.

• **Processing delay** refers to the time duration from a frame reaches the ingress port until it is fully stored in the egress queue. The delay is determined by the processing capability of the bridge or ES implementation. It is typically modeled as a constant or a bounded value in the literature.

• **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 [34].

• **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 (typically between 8 and 1024 [17]).

• **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 recently released IEEE 802.1AS-rev standard introduces a more precise synchronization error model, enabling the modeling of individual error for each node based on specific network configurations, roles of nodes, and hop distance to the grandmaster [35].

B. Traffic Models

In TSN, a traffic stream refers to a flow of data transmitted from a talker to one or multiple listeners, passing through one or multiple bridges. TSN can accommodate both real-time time-triggered (TT) traffic and lower-criticality asynchronous

traffic (e.g., audio and video (AVB) traffic). The focus of this paper is on TT traffic in real-time industrial applications.

Each TSN stream can be characterized by five parameters: release time, period, transmission delay, deadline, and jitter. Each parameter can be modeled individually to capture the specific characteristics of the targeted traffic pattern.

• **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 former allows the scheduler to configure the release time of each stream, 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, resulting in a fixed time interval between any two consecutive frames.

• **Transmission delay:** The transmission delay is the time required for a frame to be serialized on the wire following egress queuing. The transmission delay is determined by the stream's payload size and the link's line rate.

• **Deadline:** The deadline of a stream defines the time by which the released frame(s) must be received at the listener. The stream deadline can be modeled as *implicit* (equal to the period), *constrained* (less than the period), or *arbitrary*.

• **Jitter:** The jitter of a stream captures the variation of end-to-end (e2e) stream delay (i.e., the difference between the minimum and maximum delays of all the frames of a stream) [36]. Based on the stream jitter, the traffic models can be classified into *zero-jitter* model and *jitter-allowed* model.

C. Scheduling Models

The TAS-based real-time scheduling problem in TSN aims to construct a feasible schedule specifying the assignment of transmission times for each stream on individual bridges to satisfy the timing constraints of all the streams. Based on the network models and traffic models described above, a range of scheduling models have been proposed to define specific constraints on the TSN systems under study. Below, we summarize these scheduling models and categorize them according to their unique features.

• **Queuing delay:** Compared with other delay components, the queuing delay (i.e., the amount of time that a frame spends waiting in the egress queue) is decided by the schedule and has the most impact on the e2e delay of a stream. Based on the assumptions on queuing delay, the scheduling models can be classified into *no-wait* model and *wait-allowed* model. The no-wait model requires consecutive frame transmissions along the path, i.e., frames should be forwarded without queuing delay. While the wait-allowed model is more general as it allows frames to be stored in the queue with queuing delays.

• **Scheduling entity:** Depending on the objects used for the allocation of GCL entry [10], the scheduling models can be classified into *frame-based* model and *window-based* model. In the frame-based model, each GCL entry specifies the transmission time of a specific frame. In the window-based model, each GCL entry specifies a transmission time window that can be shared by a set of frames to be transmitted. In general, the window-based model can be further classified as assigned, partially assigned or non-assigned based on different frame-to-window allocations [37].

• **Queue isolation (QI):** Proper queue assignment isolates streams into different queues to avoid schedule inconsistency [16], [38], i.e., difference between the designed schedule and actual transmissions, which is mainly caused by the FIFO property of TSN egress queues. Queue isolation can be realized in the *frame* level and *stream* level. The goal of both frame-based QI and stream-based QI is to prevent changes in the forwarding order of frames that are in the same queue.

• **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. Compared to the FR model, the JRS model provides more flexibility while incurring high computational overhead.

• **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 policy may result in high latency due to the large fragment size. To address this issue, the *fragmentation* (FRAG) model is proposed, which allows the determination of the number and size of fragments.

• **Preemption:** The IEEE 802.1Qbu standard defines frame preemption as the capacity of an express frame to interrupt the transmission of a preemptable frame, and subsequently resume the preempted frame at the earliest available opportunity [39]. 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.

III. Real-Time TSN Scheduling Methods

Based on the TSN 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 [40], 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 concentrate on real-time scheduling of time-triggered traffic in TSN, approaches centered on enhancing AVB or BE traffics in mixed-criticality scenarios or improving reliability are not included. We also exclude learning-based methods that cannot provide deterministic schedules.

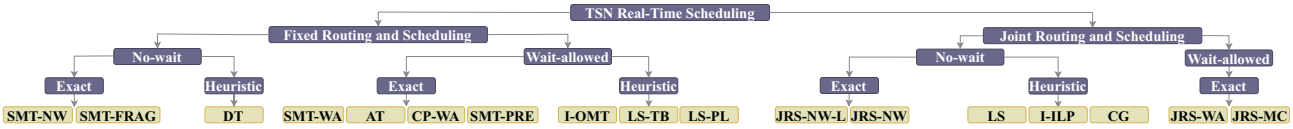

Fig. 2. Classification of the TSN real-time scheduling methods based on the employed system models.

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 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]. Fig. 2 summarizes the categorization.

A. Fixed Routing (FR) Methods

FR-based methods assume that the routing paths of individual streams are pre-determined as input and focus on the generation of a feasible schedule.

1) No-wait: The no-wait model requires that all frames are forwarded along their routing paths without any queuing delay. Among the 17 studied methods, the following methods employ a combination of the FR model and no-wait model, which tend to minimize the e2e latency.

• **SMT-NW:** Durr et al. [22] addressed the problem of reducing the e2e latency of TT traffic. The key idea is to adapt this problem to the no-wait job-shop scheduling problem [41].

• **SMT-FRAG:** Jin et al. [25] proposed a no-wait-based approach allowing fragmentation to improve the schedulability. The key idea is to jointly determine the traffic schedule along with the number/size of fragments for individual streams.

• **DT:** Zhang et al. [42] studied the high computational overhead issue induced by the constraint of non-overlap transmissions among any traffic in the no-wait model. It proposed a stream-aware model conversion algorithm to accelerate the feasible schedule search based on divisibility theory [43].

2) Wait-allowed: In the wait-allowed model, frames can be stored in the egress queue and forwarded at a later time. Thus, it introduces a larger solution space compared to the no-wait model. The following seven scheduling methods employ a combination of the FR model and wait-allowed model.

• **SMT-WA:** Craciunas et al. [16] focused on the system modeling of wait-allowed-based scheduling. It provided the SMT formulation for the scheduling constraints associated with the wait-allowed model and first introduced the queuing isolation model in the scheduling solution.

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

• **AT:** Oliver et al. [17] considered the GCL length limitation and introduced a window-based scheduling method that applied array theory to an SMT solver.

• **I-OMT:** Jin et al. [28] also studied the GCL length limitation. Instead of setting the GCL length as a constraint, it aimed to minimize the number of used GCL entries by proposing an iterative-Optimization Modulo Theories (OMT)-based approach to scheduling streams in groups.

• **CP-WA:** Vlk et al. [27] modeled the deterministic TT traffic using constraint programming (CP). It claimed that CP is more efficient compared to other formalization methods, e.g., SMT and ILP, and a decomposition optimization was also proposed to enhance the scalability of the solution.

• **SMT-PRE:** Zhou et al. [29] aimed to increase the system schedulability by enabling preemption among frames. An SMT-based approach was proposed to assign streams into two classes: express class and preemptable class.

• **LS-TB:** Vlk et al. [23] focused on addressing the scalability issue (i.e., low efficiency) in scheduling large-scale TSN traffic sets by removing reliance on third-party solvers. The proposed algorithm reverts to a previous search stage and modifies the timing and queue assignments if the current frame conflicts with any other scheduled frames.

• **LS-PL:** Bujosa et al. [31] also focused on improving the scalability of TSN networks. They proposed a heuristic algorithm that groups links into phases based on their scheduling dependency and schedules these links parallelly phase by phase.

B. Joint Routing and Scheduling (JRS) Methods

Under the FR model, a feasible schedule may not be found when the routing paths of the streams are pre-determined. By contrast, the JRS-based methods allow the scheduler to jointly determine the routes and schedules for the streams, thus offering a better chance to find a feasible schedule.

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

• **JRS-NW-L:** Falk et al. [20] proposed an ILP-based approach to determining the routing path of each stream and the schedule. Different from other ILP-based methods (e.g., [18], [21], [22]) that employ the Big-M formulation, this work used the indicator constraints to address the logical constraints.

• **JRS-NW:** Hellmanns et al. [19] addressed the high computational overhead in solving the JRS-based scheduling problem. It evaluated the impact of stream set scale and network scale on the schedulability and provided an optimization framework including three components: input optimization, model generation optimization, and solver parameter tuning.

TABLE I
A SUMMARY OF THE SYSTEM MODELS AND SCHEDULING APPROACHES EMPLOYED IN THE STUDIED TSN SCHEDULING METHODS.

Article	Year	Fully schedulable	Strictly periodic	No-wait	Window -based	Queueing	Routing	Multicast	Heuristic	Exact	Algorithms	Enhancements
Dürr et al. (SMT-NW) [22]	2016	✓	✓	✓					(✓)	✓	ILP (Tabu)	
Schweissguth et al. (JRS-WA) [18]	2017	✓	✓				✓			✓	ILP	Paths reduce
Oliver et al. (AT) [17]	2018	✓			✓	✓				✓	SMT	Array-theory
Falk et al. (JRS-NW-L) [20]	2018	✓	✓	✓			✓			✓	ILP	Logic indicator
Pahlevan et al. (LS) [24]	2019	✓	✓	✓			✓		✓		List scheduler	
Schweissguth et al. (JRS-MC) [21]	2020	✓	✓				✓	✓		✓	ILP	Paths reduce
Atallah et al. (I-ILP) [26]	2020	✓	✓	✓			✓	✓	✓		Iterative-ILP	
Jin et al. (I-OMT) [28]	2020				✓	✓			✓	(✓)	Iterative-OMT	
Falk et al. (CG) [30]	2020	✓	✓	✓			✓		✓	(✓)	Conflict-graph	
Hellmanns et al. (JRS-NW) [19]	2021	✓	✓	✓			✓			✓	ILP	Path cut-off
Jin et al. (SMT-FRAG) [25]	2021	✓	✓	✓					(✓)	✓	SMT (WCRT)	Fragmentation
Vlk et al. (CP-WA) [27]	2021	✓	✓			✓			(✓)	✓	CP (Decompose)	
Vlk et al. (LS-TB) [23]	2022		✓			✓			✓	(✓)	List scheduler	Traceback
Bujosa et al. (LS-PL) [31]	2022		✓			✓			✓		List scheduler	Per-link search
Zhou et al. (SMT-PRE) [29]	2022	✓								✓	SMT	Preemption
Zhang et al. (DT) [42]	2022	✓	✓	✓					✓	(✓)	Divisibility	

• **LS:** Pahlevan et al. [24] proposed a heuristic-based list scheduling algorithm to improve the efficiency of solving the JRS-based scheduling problem. The algorithm schedules streams one by one without backtracking, which stops and returns infeasible if any stream cannot be scheduled.

• **I-ILP:** Atallah et al. [26] considered multicast routing and proposed a heuristic solution based on 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:** Falk et al. [30] aimed to reduce the high computational overhead in existing JRS-based methods. The key idea is to gradually construct a conflict graph by capturing the collision between individual stream's transmission time, and identify an independent set to obtain a feasible schedule.

2) Wait-allowed: The following two works employ a combination of the JRS model and wait-allowed model.

• **JRS-WA:** Schweißguth et al. [18] first proposed the JRS framework and addressed the issue that FR-based methods may exclude feasible solutions without considering routing in the design space. It proposed an ILP-based approach and improved the searching efficiency by excluding infeasible routing paths during pre-processing.

• **JRS-MC:** Schweißguth et al. [21] further extended JRS-WA to support multicast traffic streams by adding additional scheduling constraints to prevent loops and negative latency.

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

It is worth noting that besides the fundamental feasibility requirement, we also reviewed TSN scheduling methods in the literature with other optimization objectives, e.g., delay and jitter minimization [44], [45], co-existence with non-TT traffic [46]–[48], and reliability [49]–[52]. Due to the page limit, we cannot include them in this paper, but summarize them in our technical report [32] for the completeness of the review.

IV. TESTBED VALIDATION

To validate the correctness and effectiveness of the studied TSN scheduling methods on COTS hardware, we set up a TSN testbed and implemented all the scheduling algorithms on it. The testbed consists of 8 bridges and 8 ESs as shown in Fig. 3(a). Each bridge is an FPGA hardware-based TTTech TSN evaluation board [53], and each ES is implemented using the Linux Ethernet stack with an external Network Interface Controller (NIC) Intel i210 as shown in Fig. 3(b). The network is set up following the ring topology as shown in Fig. 3(c) which is commonly applied in industrial scenarios [54]. We use the Linux PTP stack [55] with the gPTP profile for synchronization on end-stations, 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.

This testbed serves two main objectives: i) to calibrate key parameters assumed in the TSN system model, and ii) to validate the correctness of the scheduling methods through the comparison between the testbed results and simulation results. Given the limited scale of our testbed (8 bridges and 8 end stations only), and the difficulty to configure extensive scenarios on the testbed, we focus on functional validation rather than performance comparison using the testbed.

A. Measurements of Delays and Synchronization Error

As mentioned in Section II, most TAS-based scheduling methods assume that the processing delay, propagation delay, synchronization error, and clock offset on ES are constant or bounded numbers. **To the best of our knowledge, however, there is no existing study validating these assumptions through experimental measurements in real-world TSN testbeds.** We argue that such validation is critical as it provides the foundation for both existing and future TAS-based scheduling method design and analysis.

1) Propagation Delay: To measure the propagation delay, we directly connect a talker and a 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 Fig. 3(d), 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.

2) Processing Delay: Since we cannot measure the processing delay on the TTTech evaluation board directly, we

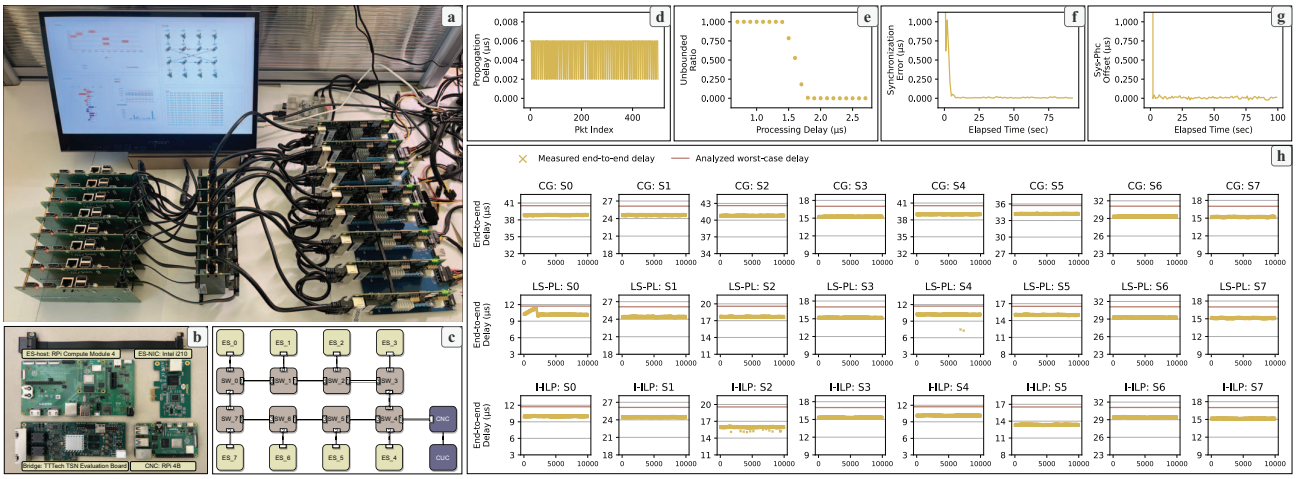

Fig. 3. (a) Overview of the 8-bridge TSN testbed; (b) Hardware components for TSN bridge and end-station; (c) The logical topology of the testbed; (d) – (g) Measurement results of the propagation delay, processing delay, synchronization error, and physical clock to system clock offset; (h) Measurement results of the e2e delay for three representative methods out of the 17 scheduling methods.

infer its upper bound by observing the e2e delay of a stream. Specifically, we gradually increase 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. Fig. 3(e) shows that the one-hop processing delay can be bounded within 1.9 μs in our testbed.

3) Synchronization Error: Fig. 3(f) 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 [35]. After that, the synchronization error can be bounded within 10 ns.

4) Clock Offset on System: Fig. 3(g) 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, in our subsequent simulation-based evaluation experiments, we set the propagation delay, processing delay, synchronization error, and clock offset as 6 ns, 1.9 μs, 10 ns, and 50 ns, respectively.

B. Performance Validation

Before conducting extensive simulation-based experiments, we need to validate if the performance of the TSN scheduling methods is consistent on both the real-world testbed and through simulations. Such validation not only confirms the theoretical performance of each method but also ensures 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, and we record the e2e delay of 10000 frames for each stream.

Fig. 3(h) compares the measured e2e delays of three out of the 17 methods (other results can be found in [32]) on the testbed (yellow line) and the analyzed worst-case delay from the simulation (red line). **Overall, our testbed results validate the correctness of all the methods** since the analyzed worst-case e2e delays of each method are always bounded by the corresponding measurement results. Beyond that, we have two important observations. First, most of the streams experience a relatively stable delay ($<$100 ns variation), but some streams are observed to have delay fluctuations under certain methods. For example, in Fig. 3(h), 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 (see Fig. 3(h)). This gap primarily stems from two factors: 1) an enforced error margin of up to 3.2 μs by the TTTech evaluation board to accommodate timing errors on the bridge; and 2) an up to 1.9 μs processing delay on the

TABLE II
PARAMETER SETTINGS FOR SIMULATION-BASED EVALUATION.

	Parameter	Type	Value
Stream set	Number of streams	-	{10, 40, 70, …, 190, 220}
	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}
	Number of frames	-	{8, 16, 32, …, 2048, 4096}
	Stream payload (bytes)	Tiny size	50
		Small size	50 – 500
		Medium size	200 – 1500
		Large size	500 – 4500
		Extra size	1500 – 4500
	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}

bridge identified during our measurements.

V. SIMULATION-BASED EXPERIMENTAL SETUP

We now present the details of our simulation-based experimental setup to evaluate the 17 scheduling methods.

A. Parameter Settings

To ensure a fair evaluation among the selected TSN scheduling methods, we follow the parameter settings below in the experiments, which are summarized in Table II.

1) Stream Set Settings: We control the randomly generated TSN stream set by tuning the following parameters:

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 typical 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 [56], resulting in a very low schedulability ratio and impractical runtime for most of the evaluated methods.

Stream period. Following the TSN profile for industrial automation use cases in IEC/IEEE 60802 [36], 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 [57]. Thus, we define 6 stream period types, as shown in Table II, 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, …, $\geq$4096 to facilitate point plotting as shown in Table II.

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 [34], 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 II) based on the typical configurations in industrial applications.

Stream deadline. Theoretically, the minimum e2e delay experienced by a stream equals to 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 IV. We define 5 stream deadline types (see Table II) to aid the generation of random stream sets in our experiments.

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

Network topology. In the experiments, we employ four commonly used topologies: 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 [35] under our topology settings. The number of links is determined accordingly under different network topologies, as detailed in Table II.

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 [58]. The number of queues on each egress port is fixed to 8 which is a common setting in TSN bridges. We also assume that all eight queues are exclusively dedicated to handling critical TT traffic.

B. Algorithm Implementation

We implement all the 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 [59]. For ILP-based methods, we use the Gurobi optimizer, one of the most advanced ILP solvers [60][3]. For methods without relying on third-party software, we implement them from scratch using native Python. 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.

[3]Following the original papers, we use the CPLEX ILP solver for JRS-NW-L for the logical indicator [61], and the IBM CP Optimizer for CP-WA, and the Sklearn library [62] to implement the spectral clustering based stream set partition algorithm for I-ILP.

JRS-NW-L/JRS-MC. Since the model generation optimization techniques proposed in JRS-NW-L and JRS-MC were found to be counter-effective in a recent study [19], we omit such optimizations 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 paper as it is rarely called (only 0.96%) in the problem-solving process.

LS. The FINDIT function used in LS 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 make sure each frame is mapped 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 (e.g., max number of windows for AT, max fragment count for SMT-FRAG/SMT-PRE), we follow their default settings in the original papers. In addition, as suggested in the IEEE 802.1Qcc standard [63], we apply the shortest path routing algorithm to construct the routing path in FR-based methods.

C. Evaluation Environment

Our experiments are conducted on Chameleon Cloud, an NSF-sponsored public cloud computing platform [64]. We utilize 8 nodes equipped with 2 AMD EPYC® CPUs, 64 cores per CPU with a clock speed of 2.45 GHz, and 256 GB DDR4 memory. To make the benchmark robust and representative, we ran a total of 38400 problem instances covering all combinations of our parameter settings in Table II, 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 is 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 exceeds the time threshold, the algorithm is terminated and returned 'unknown'.

VI. Experimental Evaluation

We perform comprehensive simulation-based evaluation for the 17 TAS-based scheduling methods by comparing their *schedulability* and *scalability*. We have also conducted experiments using other performance metrics to evaluate the quality of schedule, e.g., GCL length, link utilizations, and queue utilization. These results are available in [32].

A. Schedulability

1) Setup: As discussed in Section V-C, 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 the unknown results, we are unable to 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. 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. Calculated from the known results for both methods, SA can effectively reduce the impact posed by unknown outcomes while ensuring a sufficient number of compared instances.

2) Results: Based on the two evaluation scenarios, we conduct extensive experiments under various stream and network settings as described in Section V-A.

The first set of experiments evaluates the SRs of all the methods by varying the parameter settings summarized in Table II. Specifically, Fig. 4 shows the SR as functions of the number of streams, frames, bridges and links, respectively. In each subfigure, only one parameter is varied with all other parameters fixed. We use dashed lines to denote data points comprising over 90% unknown results. Fig. 5 shows the SR of each method under different topologies, periodicity patterns, payload sizes, and deadlines.

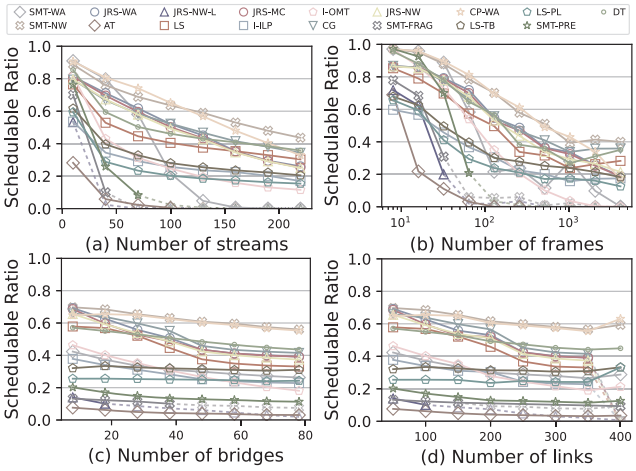

Fig. 4. SR comparison under different stream set and network settings by varying the parameter values.

The second set of experiments performs the pairwise comparison using the SA metric and the results are shown in a heatmap in Fig. 6. 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.

3) Discussion: We discuss the experimental results across two dimensions of granularity. We first perform comparisons among different scheduling models discussed in Section II to show their pros and cons. We then compare the performance of each method to discuss their advantages and limitations.

Model comparisons. We discuss the results of model comparison by categorizing two types of models: models with varied or stable performance between ES1 and ES2. This classification is based on their observed trends of results in the experiments. The first comparison includes: 1) JRS and FR models, 2) FRAG/PRE and non-FRAG/non-PRE models, and 3) no-wait and wait-allowed models. The second comparisons includes: 1) fully and partially schedulable models, and 2) frame-based model and window-based scheduling model.

a) Models with inconsistent performance on SR and SA. The experimental 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.

Specifically, JRS model dominates FR model on SA, but it may cause lower performance on SR. For example, comparing in Fig. 6, 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 Fig. 4(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 Fig. 5(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.

We also find similar patterns when comparing FRAG/PRE vs. non-FRAG/non-PRE models and no-wait vs. wait-allowed models. For example, in Fig. 6, SMT-FRAG and SMT-PRE show the average SA values of 14.7% (Top-1) and 14.0% (Top-6), respectively, outperforming the average of others. However, both methods experience significantly reduced schedulability due to their larger computational overhead as shown in Fig. 4. Similarly, wait-allowed-based methods (SMT-WA and JRS-WA) dominate their counterpart no-wait-based methods (SMT-NW and JRS-NW) on SA as expected, but their difference in SR is negligible (see Fig. 4 and Fig. 6).

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.

As shown in Fig. 6, LS shows higher average SA (7.24%) than LS-PL (2.8%) and LS-TB (4.29%), where the former method follows the fully schedulable model and the latter two methods follow the partially schedulable model. The comparison results are retained when evaluating the SR performance. As shown in Fig. 4, LS consistently outperforms LS-PL and LS-TB under varied workloads and network scales.

Similarly, as shown in Fig. 6, the frame-based methods SMT-NW and SMT-WA dominate the window-based method AT on SA. Consistently, as shown in Fig. 4, both SMT-NW and SMT-WA also consistently outperform AT in terms of SR by increasing either the stream set 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 comparison of schedulability performance among individual scheduling methods. According to the classification in Table I, each

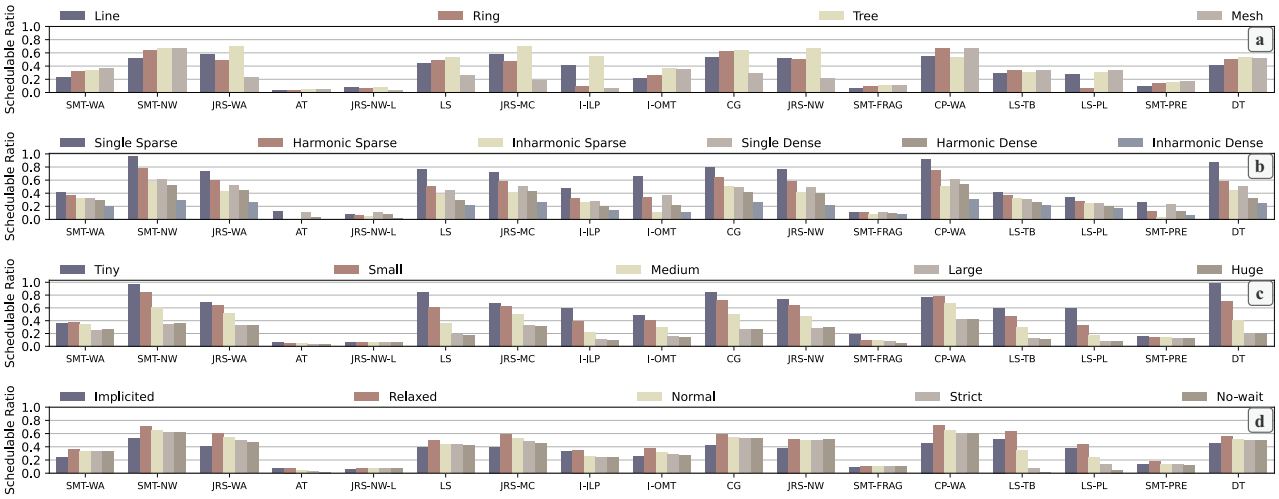

Fig. 5. SR comparison under different stream set and network settings by varying the parameter types: (a) Network topology. (b) Periodicity pattern. (c) Payload size. (d) Deadline.

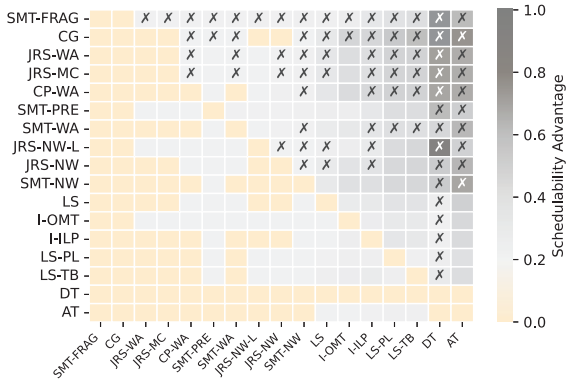

Fig. 6. Pairwise SA comparison among the studied scheduling methods.

method is either a heuristic or exact solution. Thus, we first perform comparisons between heuristic approaches and exact solutions. We then delve into details of 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 Fig. 4, 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 the SR of SMT-WA rapidly declines to zero. Both methods are under the FR model and wait-allowed model as shown in Fig. 2. Similar trends can also be observed by comparing other pairs of heuristic and exact solutions, such as JRS-NW vs. DT.

Due to their inherent efficiency, heuristic approaches also benefit more from complex models compared to exact solutions. For example, in Fig. 4(a), heuristic CG and exact solution JRS-NW exhibit similar SR when the number of streams is less than 80. However, when the number of streams increases, CG outperforms JRS-NW with a widening gap. Both methods are under JRS model and no-wait model. Similar trends can also be observed in Fig. 4 that the heuristic method I-OMT outperforms the exact method I-ILP consistently.

b) Comparison among heuristic algorithms. Our results show that the performance of four heuristic algorithms significantly degrades under certain scenarios. 1) I-ILP demonstrates lower schedulability on routable topologies because of its inefficient DAMR routing algorithm. For example, as shown in Fig. 5(a), SRs of I-ILP drop from 41.6% (line) and 54.6% (tree) to 9.6% (ring) and 7.4% (mesh). 2) Fig. 5(a) shows that LS-PL suffers from a notably low SR (7.0%) in networks with ring topology. This is due to the cyclic dependencies, causing frequent failures in its phase division algorithm. 3) Under strict deadline settings, both LS-TB and LS-PL show low schedulability in Fig. 5(d) 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%). 4) As shown in Fig. 5(b), 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%). This decrease is mainly 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 finding 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 (e.g., with more than 100 streams), 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).

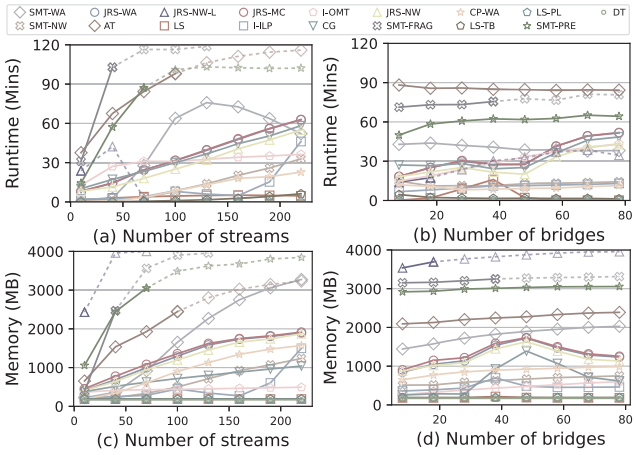

Fig. 7. Runtime and memory consumption comparisons under varied stream set and network settings.

B. Scalability

In this section, we compare the scalability of the scheduling methods in terms of runtime and memory consumption, which are critical performance metrics to evaluate how well the scheduling algorithm will scale in practice [65].

1) Setup: In our experiments, the runtime of a scheduling method consists of the pre-processing time (filtering invalid solution space), the constraint adding time, and the problem solving time. If a 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 set a 4GB threshold to allocate enough RAM while avoiding swap space use, and track the peak memory usage in each experiment.

2) Results: Fig. 7 shows the runtime and memory consumption performance with varied number of streams and bridges.

Overall trend. In Fig. 7(a)(b), when 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 with 10 streams to 48.6 minutes with 220 streams. Likewise, the average memory consumption increases from 476 MB with 10 streams to 1800 MB with 220 streams.

Interestingly, adding more bridges to the network has limited effect on the runtime. Overall, as shown in Fig. 7(c), the runtime of most methods slightly increases from 23.6 minutes with 8 bridges to 34.1 minutes with 78 bridges. Among these methods, FR-based methods only show a modest increase from 25.4 to 29.2 minutes, while the JRS-based methods show a more substantial rise from 19.2 minutes to 41.8 minutes. As shown in Fig. 7(d), the memory consumption remains relatively steady for FR-based methods with an average increase of 183 MB when the number of bridges is increased from 8 to 78. As an exception, JRS-based methods peak at an average of 2070 MB with 48 bridges before dropping. This observation may be due to several factors. For example, a larger network may lead to longer routing paths, thereby requiring more scheduling effort. While at the same time, this may reduce traffic density and 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.

VII. Takeaway Lessons

We now summarize takeaways from this study, on both fair performance evaluation and TSN scheduling algorithm design.

A. Fair Performance Evaluation

Parameter settings. Research studies may make unfair comparisons under specific settings and result in biased conclusions. To mitigate this issue, we propose two ways to avoid bias. 1) We include a broader range of parameter settings to better understand the overall performance of the individual methods and improve the fidelity and applicability of the evaluation. 2) We select experimental settings based on real-world scenarios or from standards and profiles if the computing resource is limited to perform extensive experiments. For instance, [36], [56] offer realistic use cases that can serve as common evaluation scenarios. However, it is worth noting that given the early stage of TSN research, the availability of real-world 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 Schedulable Ratio (SR) and Schedulability Advantage (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).

B. Algorithm Design

We provide the following insights and recommendations for future TAS-based real-time scheduling algorithm 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 TT traffic and synchronization traffic. Collisions between the two traffic types 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 predefined 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 interframe 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 VI, it is important to select the right model and scheduling methods for performance optimization under specific scenarios. Below, we provide suggestions based on the pros and cons of the models and algorithms observed from our experiments. First of all, using simple models (e.g., no-wait model) or heuristic approaches (e.g., the list scheduler) can achieve better performance with large stream set under resource limitation. Secondly, using JRS model can be counter-effective in large-scale network compared with the FR model due to its low efficiency. Thirdly, enlarging the search space on ES side (fully/partially schedulable) might be more effective on improving schedulability than search space on the bridge side (no-wait/wait-allowed).

VIII. Threats to Validity

It is worth noting that although our findings and conclusions are based on thorough evaluation of representative algorithms in the literature, they may not be applicable to all comparison scenarios. We thus summarize the following limitations of our experimental studies to make the readers be aware of the potential threats to their general applicability and validity.

A. Model and Algorithm Comparison

The primary goal of this study is to evaluate the performance of 17 representative TAS-based scheduling methods under various scenarios. The discussions on the model and algorithm comparisons are based on the observations from the evaluation results of individual methods under practical experimental settings. Providing a thorough and independent model/algorithm comparison requires a completely different experiment design to isolate model, algorithm, and implementation, which is out of the scope of this study.

In addition, since TSN research in recent years has been explosive, we cannot include all methods in the experimental comparison. For example, only 2 window-based methods [17], [28] are considered in the performance comparison, and this is not sufficient to conclude the performance of window-based model in the general case. Furthermore, the evaluation on some individual methods may not be sufficient to represent the performance of the model and algorithm they employ. For example, AT and SMT-PREP are only designed as proofs of concept without performance optimization. Similarly, the efficiency of some exact solutions may be further improved using incremental scheduling or decomposition approaches [66].

B. Individual Method Comparison

Although many well-designed experimental setups are employed in our study to ensure fairness, potential issues *may exist* to result in inconsistent conclusions.

Additional parameter settings. For the methods that require additional parameters (e.g., max number of windows for AT), we set those parameters in our experiments the same as their settings in the original papers. However, the performance of individual methods may be further improved by fine-tuning the parameters, especially for those methods that are sensitive to certain parameter settings, e.g., windows/fragmentations/preemptions setup in [17], [25], [29].

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, there may exist some issues that could potentially limit the performance of certain methods: 1) the selected solver and corresponding problem formulation may significantly affect results. For example, the observed low performance of JRS-NW-L may be due to the low efficiency of Cplex ILP solver on addressing logical constraints. Furthermore, 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 on adding constraints on variables rather than searching for solutions, especially with large problem instances.

IX. Conclusion and Future Work

This paper examines 17 representative TAS-based real-time scheduling methods in Time-Sensitive Networking (TSN) and establishes a benchmark for performance comparison among individual methods and across different system models. Comprehensive experiments are designed and conducted using both high-fidelity simulation and real-world testbed to help evaluate the performance of the state-of-the-art methods and identify open problems in TSN scheduling design and implementation.

For future work, we will include realistic problem instances from avionic and automobile industries, as well as incorporating fault tolerance scenarios and various traffic shapers into the evaluation. To further evaluate the correctness and practicability of the existing methods, more comprehensive empirical experiments will be conducted on our TSN testbed. Finally, we will encourage the community to utilize our open-source toolkit to evaluate their scheduling methods to boost the development of TSN-related R&D projects.

X. Acknowledgement

The work is supported in part by the National Science Foundation Grant CNS-1932480, CNS-2008463, CCF-2028875, CNS-1925706, and the NASA STRI Resilient Extraterrestrial Habitats Institute (RETHi) under grant number 80NSSC19K1076.

References

[1] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial internet of things: Challenges, opportunities, and directions," *IEEE transactions on industrial informatics*, vol. 14, no. 11, pp. 4724–4734, 2018.

[2] W. Z. Khan, M. Rehman, H. M. Zangoti, M. K. Afzal, N. Armi, and K. Salah, "Industrial internet of things: Recent advances, enabling technologies and open challenges," *Computers & Electrical Engineering*, vol. 81, p. 106522, 2020.

[3] J. Wang, T. Zhang, D. Shen, X. S. Hu, and S. Han, "Harp: Hierarchical resource partitioning in dynamic industrial wireless networks," in *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2022, pp. 1029–1039.

[4] "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*, pp. 1–72, 2010.

[5] "IEEE standard for local and metropolitan area networks–bridges and bridged networks - amendment 34:asynchronous traffic shaping," *IEEE Std 802.1Qcr-2020*, pp. 1–151, 2020.

[6] "IEEE standard for local and metropolitan area networks – bridges and bridged networks - amendment 25: Enhancements for scheduled traffic," *IEEE Std 802.1Qbv-2015*, pp. 1–57, 2016.

[7] L. Zhao, P. Pop, and S. Steinhorst, "Quantitative performance comparison of various traffic shapers in time-sensitive networking," *IEEE Transactions on Network and Service Management*, vol. 19, no. 3, pp. 2899–2928, 2022.

[8] T. Stüber, L. Osswald, S. Lindner, and M. Menth, "A survey of scheduling algorithms for the time-aware shaper in time-sensitive networking (tsn)," *IEEE Access*, 2023.

[9] G. Wang, T. Zhang, C. Xue, J. Wang, M. Nixon, and S. Han, "Time-sensitive networking for industrial automation: Challenges, opportunities, and directions," *arXiv preprint arXiv:2306.03691*, 2023.

[10] D. Hellmanns, J. Falk, A. Glavackij, R. Hummen, S. Kehrer, and F. Dürr, "On the performance of stream-based, class-based time-aware shaping and frame preemption in TSN," in *2020 IEEE International Conference on Industrial Technology (ICIT)*. IEEE, 2020, pp. 298–303.

[11] A. Minaeva and Z. Hanzálek, "Survey on periodic scheduling for time-triggered hard real-time systems," *ACM Computing Surveys (CSUR)*, vol. 54, no. 1, pp. 1–32, 2021.

[12] L. Deng, G. Xie, H. Liu, Y. Han, R. Li, and K. Li, "A survey of real-time ethernet modeling and design methodologies: From AVB to TSN," *ACM Computing Surveys (CSUR)*, vol. 55, no. 2, pp. 1–36, 2022.

[13] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. ElBakoury, "Ultra-low latency (ULL) networks: The IEEE TSN and IETF DetNet standards and related 5G ULL research," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 88–145, 2018.

[14] Y. Seol, D. Hyeon, J. Min, M. Kim, and J. Paek, "Timely survey of time-sensitive networking: Past and future directions," *IEEE Access*, vol. 9, pp. 142 506–142 527, 2021.

[15] A. Nasrallah, V. Balasubramanian, A. Thyagaturu, M. Reisslein, and H. ElBakoury, "TSN algorithms for large scale networks: A survey and conceptual comparison," *arXiv preprint arXiv:1905.08478*, 2019.

[16] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, "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*, 2016, pp. 183–192.

[17] R. S. Oliver, S. S. Craciunas, and W. Steiner, "IEEE 802.1 Qbv gate control list synthesis using array theory encoding," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2018, pp. 13–24.

[18] E. Schweissguth, P. Danielis, D. Timmermann, H. Parzyjegla, and G. Mühl, "ILP-based joint routing and scheduling for time-triggered networks," in *Proceedings of the 25th International Conference on Real-Time Networks and Systems*, 2017, pp. 8–17.

[19] D. Hellmanns, L. Haug, M. Hildebrand, F. Dürr, S. Kehrer, and R. Hummen, "How to optimize joint routing and scheduling models for TSN using integer linear programming," in *29th International Conference on Real-Time Networks and Systems*, 2021, pp. 100–111.

[20] J. Falk, F. Dürr, and K. Rothermel, "Exploring practical limitations of joint routing and scheduling for TSN with ILP," in *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2018, pp. 136–146.

[21] E. Schweissguth, D. Timmermann, H. Parzyjegla, P. Danielis, and G. Mühl, "ILP-based routing and scheduling of multicast realtime traffic in time-sensitive networks," in *2020 IEEE 26th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2020, pp. 1–11.

[22] F. Dürr and N. G. Nayak, "No-wait packet scheduling for IEEE time-sensitive networks (TSN)," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, 2016, pp. 203–212.

[23] M. Vlk, K. Brejchová, Z. Hanzálek, and S. Tang, "Large-scale periodic scheduling in time-sensitive networks," *Computers & Operations Research*, vol. 137, p. 105512, 2022.

[24] M. Pahlevan, N. Tabassam, and R. Obermaisser, "Heuristic list scheduler for time triggered traffic in time sensitive networks," *ACM Sigbed Review*, vol. 16, no. 1, pp. 15–20, 2019.

[25] X. Jin, C. Xia, N. Guan, and P. Zeng, "Joint algorithm of message fragmentation and no-wait scheduling for time-sensitive networks," *IEEE/CAA Journal of Automatica Sinica*, vol. 8, no. 2, pp. 478–490, 2021.

[26] A. A. Atallah, G. B. Hamad, and O. A. Mohamed, "Routing and scheduling of time-triggered traffic in time-sensitive networks," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 7, pp. 4525–4534, 2019.

[27] M. Vlk, Z. Hanzálek, and S. Tang, "Constraint programming approaches to joint routing and scheduling in time-sensitive networks," *Computers & Industrial Engineering*, vol. 157, p. 107317, 2021.

[28] X. Jin, C. Xia, N. Guan, C. Xu, D. Li, Y. Yin, and P. Zeng, "Real-time scheduling of massive data in time sensitive networks with a limited number of schedule entries," *IEEE Access*, vol. 8, pp. 6751–6767, 2020.

[29] Y. Zhou, S. Samii, P. Eles, and Z. Peng, "Time-triggered scheduling for time-sensitive networking with preemption," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2022, pp. 262–267.

[30] J. Falk, F. Dürr, and K. Rothermel, "Time-triggered traffic planning for data networks with conflict graphs," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2020, pp. 124–136.

[31] D. Bujosa, M. Ashjaei, A. V. Papadopoulos, T. Nolte, and J. Proenza, "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*, 2022, pp. 70–80.

[32] C. Xue, T. Zhang, Y. Zhou, M. Nixon, A. Loveless, and S. Han, "Real-time scheduling for 802.1Qbv time-sensitive networking (TSN): A systematic review and experimental study," *arXiv preprint arXiv:2305.16772*, 2023.

[33] C. Xue, "Open-source toolkit for TSN scheduling algorithm design and evaluation," Feb 2024. [Online]. Available: https://github.com/ChuanyuXue/tsnkit

[34] IEEE Standards Association and others, "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)*, pp. 1–1993, 2018.

[35] "IEEE standard for local and metropolitan area networks–timing and synchronization for time-sensitive applications," *IEEE Std 802.1AS-2020*, pp. 1–421, 2020.

[36] J. Dorr, K. Weber, and S. Zuponcic, *Use Cases IEC/IEEE 60802*. [Online]. Available: https://www.ieee802.org/1/files/public/docs2018/60802-industrial-use-cases-0918-v13.pdf

[37] M. Barzegaran, N. Reusch, L. Zhao, S. S. Craciunas, and P. Pop, "Real-time traffic guarantees in heterogeneous time-sensitive networks," in *Proceedings of the 30th International Conference on Real-Time Networks and Systems*, 2022, pp. 46–57.

[38] Y. Lin, X. Jin, T. Zhang, M. Han, N. Guan, and Q. Deng, "Queue assignment for fixed-priority real-time flows in time-sensitive networks: Hardness and algorithm," *Journal of Systems Architecture*, vol. 116, p. 102141, 2021.

[39] IEEE, "IEEE standard for local and metropolitan area networks—bridges and bridged networks—amendment 26: frame preemption: 802.1 qbu-2016," 2016.

[40] B. Kitchenham, "Procedures for performing systematic reviews," *Keele, UK, Keele University*, vol. 33, no. 2004, pp. 1–26, 2004.

[41] A. Mascis and D. Pacciarelli, "Job-shop scheduling with blocking and no-wait constraints," *European Journal of Operational Research*, vol. 143, no. 3, pp. 498–517, 2002.

[42] Y. Zhang, Q. Xu, S. Wang, Y. Chen, L. Xu, and C. Chen, "Scalable no-wait scheduling with flow-aware model conversion in time-sensitive networking," in *IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2022, pp. 413–418.

[43] J. Pommaret and A. Quadrat, "Generalized bezout identity," *Applicable Algebra in Engineering, Communication and Computing*, vol. 9, no. 2, pp. 91–116, 1998.

[44] R. Mahfouzi, A. Aminifar, S. Samii, P. Eles, and Z. Peng, "Security-aware routing and scheduling for control applications on ethernet TSN networks," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 25, no. 1, pp. 1–26, 2019.

[45] X. Dai, S. Zhao, Y. Jiang, X. Jiao, X. S. Hu, and W. Chang, "Fixed-priority scheduling and controller co-design for time-sensitive networks," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–9.

[46] M. Barzegaran, B. Zarrin, and P. Pop, "Quality-of-control-aware scheduling of communication in TSN-based fog computing platforms using constraint programming," in *2nd Workshop on Fog Computing and the IoT (Fog-IoT 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[47] B. Houtan, M. Ashjaei, M. Daneshtalab, M. Sjödin, and S. Mubeen, "Synthesising schedules to improve QoS of best-effort traffic in TSN networks," in *29th International Conference on Real-Time Networks and Systems*, 2021, pp. 68–77.

[48] P.-J. Chaine and M. Boyer, "Shortening gate closing time to limit bandwidth waste when implementing time-triggered scheduling in TAS/TSN," in *International Conference on Real-Time Networks and Systems (RTNS) 2022*, 2022.

[49] W. Ma, X. Xiao, G. Xie, N. Guan, Y. Jiang, and W. Chang, "Fault tolerance in time-sensitive networking with mixed-critical traffic," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[50] N. Reusch, P. Pop, and S. Craciunas, "Technical report: Safe and secure configuration synthesis for TSN-based distributed cyber-physical systems using constraint programming," 2020.

[51] Y. Zhou, S. Samii, P. Eles, and Z. Peng, "ASIL-decomposition based routing and scheduling in safety-critical time-sensitive networking," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2021, pp. 184–195.

[52] S. S. Craciunas and R. S. Oliver, "Out-of-sync schedule robustness for time-sensitive networks," in *2021 17th IEEE International Conference on Factory Communication Systems (WFCS)*. IEEE, 2021, pp. 75–82.

[53] T. Industrial, "Edge IP solution." [Online]. Available: https://www.tttech-industrial.com/products/slate/edge-ip-solution

[54] P. Park, M. Son, J. Lee, and J. Yoon, "Performance evaluation of the efficient precise time synchronization protocol for the redundant ring topology network," in *2021 IEEE/AIAA 40th Digital Avionics Systems Conference (DASC)*, 2021, pp. 1–10.

[55] R. Cochran and C. Marinescu, "Design and implementation of a PTP clock infrastructure for the Linux kernel," in *2010 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*. IEEE, 2010, pp. 116–121.

[56] W. Fischer, J. Gelish, and M. Hegarty, *Aerospace TSN use cases, traffic types, and requirements*. [Online]. Available: https://www.ieee802.org/1/files/public/docs2021/dp-Jabbar-et-al-Aerospace-Use-Cases-0321-v06.pdf

[57] M. Mohaqeqi, M. Nasri, Y. Xu, A. Cervin, and K.-E. Årzén, "Optimal harmonic period assignment: complexity results and approximation algorithms," *Real-Time Systems*, vol. 54, pp. 830–860, 2018.

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

[59] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *in Proceedings of the 14th International Conference of Tools and Algorithms for the Construction and Analysis of Systems (TACAS), 2008*. Springer, 2008, pp. 337–340.

[60] Gurobi Optimization, LLC, "Gurobi optimizer reference manual," 2021. [Online]. Available: https://www.gurobi.com/documentation/9.1/refman/index.html

[61] Center, IBM Knowledge, "IBM ILOG CPLEX Optimization Studio," 2019. [Online]. Available: https://www.ibm.com/products/ilog-cplex-optimization-studio

[62] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *the Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[63] "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.

[64] K. Keahey, J. Anderson, Z. Zhen, P. Riteau, P. Ruth, D. Stanzione, M. Cevik, J. Colleran, H. S. Gunawi, C. Hammock *et al.*, "Lessons learned from the chameleon testbed," in *2020 USENIX annual technical conference (USENIX ATC 20)*, 2020, pp. 219–233.

[65] D. Pannell, "Choosing the right TSN tools to meet a bounded latency," *IEEE SA Ethernet & IP@ Automotive Technology Day*, 2019.

[66] A. Finzi and R. Serna Oliver, "General framework for routing, scheduling and formal timing analysis in deterministic time-aware networks," in *34th Euromicro Conference on Real-Time Systems (ECRTS 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.