InNetScheduler: In-network scheduling for time- and event-triggered critical traffic in TSN

Xiangwen Zhuge*, Xinjun Cai*, Xiaowu He*, Zeyu Wang*, Fan Dang†, Wang Xu† and Zheng Yang*✉
* School of Software and BNRist, Tsinghua University † Global Innovation Exchange, Tsinghua University
{zgxw18, caixj08, horacehxw, ycdfwzy}@gmail.com, dangfan@tsinghua.edu.cn,
wangxu.93@hotmail.com, hmilyyz@gmail.com

***Abstract*—Time-Sensitive Networking (TSN) is an enabling technology for Industry 4.0. Traffic scheduling plays a key role for TSN to ensure low-latency and deterministic transmission of critical traffic. As industrial network scales, TSN networks are expected to support a rising number of both time-triggered and event-triggered critical traffic (TCT and ECT). In this work, we present InNetScheduler, the first in-network TSN scheduling paradigm that boosts the throughput, *i.e.*, number of scheduled data flows, of both traffic types. Different from existing approaches that conduct entire scheduling on the server, InNetScheduler leverages the computation resources on switches to promptly schedule latency-critical ECT, and delegate the computational-intensive TCT scheduling to server. The key innovation of InNetScheduler includes a Load-Aware Optimizer to mitigate ECT conflicts, a Relaxated ECT Scheduler to accelerate in-network computation, and End-to-End Determinism Guarantee to lower scheduling jitter. We fully implement a suite of InNetScheduler-compatible TSN switches with hardware-software co-design. Extensive experiments are conducted on both simulation and physical testbeds, and the results demonstrate InNetScheduler's superior performance. By unleashing the power of in-network computation, InNetScheduler points out a direction to extend the capacity of existing industrial networks.**

***Index Terms*—Time-Sensitive Networking, Traffic Scheduling, Time-triggered Critical Traffic, Event-triggered Critical Traffic**

I. Introduction

Time-Sensitive Networking (TSN) has gained significant attention as a pivotal technology towards Industry 4.0, and is widely deployed in time-critical scenarios like automated production lines [1]–[3]. Traffic scheduling is essential to TSN. It ensures the real-time and deterministic transmission of critical traffic by strategically planning its forwarding path and time. The greater number of critical flows an algorithm can schedule, the more production equipment and manufacturing tasks can be supported in a production line.

There are two types of critical traffic to be scheduled in TSN, *i.e.*, time-triggered critical traffic (TCT) and event-triggered critical traffic (ECT) [4]–[6]. For instance, in a typical industrial automation production line, actuator devices require periodic control commands, whereas sensor devices may trigger shutdown signals to protect the production line when overheating. In this scenario, the periodic control commands belong to TCT, and the shutdown signals belong to ECT. Consequently, in practical industrial systems, it is crucial to concurrently schedule both of the above critical traffic.

✉ Zheng Yang is the corresponding author.

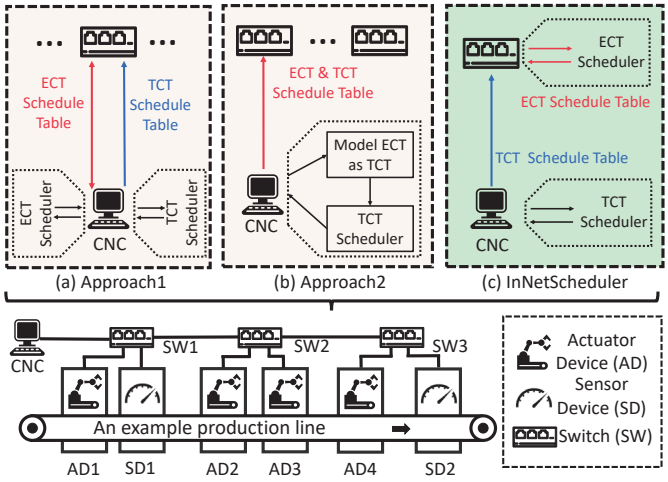


Fig. 1: **A comparison of existing joint scheduling approaches and our InNetScheduler.** (a) Approach1: In CNC, scheduling all TCT offline before the network deployment and scheduling ECT online when it arrives. (b) Approach2: In CNC, modeling ECT as TCT and converting the joint scheduling task into an offline TCT scheduling problem. (c) InNetScheduler: scheduling TCT offline in CNC and scheduling ECT online when it arrives in switches.

To increase the network throughput, *i.e.*, the number of scheduled critical flows, pioneering researches have explored the joint scheduling of TCT and ECT. All of them conducts scheduling tasks in TSN's centralized network configuration (CNC). As shown in Fig. 1(a), an intuitive approach is to schedule all TCT offline before the network deployment, and then schedule ECT online as it arrives during the production process. Due to the high communication delay between CNC and TSN switches, this approach is unable to finalize the scheduling before ECT's deadline. Another group of studies shown in Fig. 1(b) first model ECT as TCT, and then convert the joint scheduling task into an offline TCT scheduling problem in CNC [7], [8]. Since the scheduling algorithm can not access ECT's arrival time, they have to reserve redundant network resources to ensure ECT's successful transmission, sacrificing the network throughput. In summary, given that two aforementioned approaches are deployed in CNC, they are unable to get ECT's arrival time accurately and promptly, therefore limiting their network throughput.

Granted with multi-core processors and operating systems, commercial switches nowadays can perform not only pure packet forwarding but computations as well [9]–[11]. This trend leads to the birth of a new computational paradigm called *in-network computing* [12]. Moreover, switches are located at the core of the network, making it possible for them to accurately and promptly obtain the ECT's arrival information. Consequently, as Fig. 1(c) illustrates, our key insight is to leverage in-network computing paradigm to *(i)* incorporate both CNC and switches in TSN scheduling; and *(ii)* exploit the ECT's arrival information to improve network throughput.

Albeit inspiring, applying the above intuition in TSN scheduling is non-trivial and faces the following challenges. **(1) In-network paradigm causes ECT conflicts:** In the in-network scheduling paradigm, each TSN switches are supposed to independently schedule the passing by ECT. Due to the unawareness of each other's scheduling decision, they might inadvertently make ECT conflicts on the links with limited time-slot resources, and consequently degrade the network throughput. **(2) Complex scheduling hinders task computation:** To meet the transmission deadline of ECT, the in-network scheduling needs to be conducted immediately (typically within 1 millisecond). However, TSN scheduling is a NP-Hard combinatorial optimization problem, and existing scheduling methods incur significant computational overhead. As a result, it is challenging for the resource-constrained network switches to conduct scheduling tasks on time. **(3) Volatile resources impair scheduling determinism:** ECT scheduling comprises both task computation and critical data exchange. Since the operating system dynamically allocates computation resources and other applications occasionally preempt internal communication channels, the time fluctuation of in-network scheduling is considerable. This makes it hard to ensure the completion of ECT scheduling before deadline.

In this work, we present InNetScheduler, the first in-network TSN scheduling paradigm that improves the throughput of both TCT and ECT. Specifically, we delegate the computational-intensive offline TCT scheduling to CNC, and the latency-sensitive online ECT scheduling to switches. To tackle the above mentioned challenges, we propose the following three technologies. (1) **Load-aware Optimizer:** InNetScheduler takes the load balancing into account when making the offline TCT scheduling. This reduces the number of resource-limited links and thus significantly decreases the probability of ECT conflicts. (2) **Relaxated ECT Scheduler:** In order to meet the deadline, InNetScheduler adopts a dynamic programming algorithm of complexity $\mathcal{O}(n^3)$ to approximate the NP-Hard ECT scheduling problem. (3) **End-to-End Determinism Guarantee:** InNetScheduler incorporates multiple techniques to ensure the end-to-end determinism of ECT scheduling on switches, including hardware resource isolation, software process management, and reliable data exchange.

The main contributions are as follows:

- We present InNetScheduler, as far as we are aware of, the first in-network TSN scheduling paradigm that delegates ECT scheduling to switches. It significantly improves the network throughput of all critical traffic, enabling production lines to support more devices and manufacturing tasks.
- We propose three novel technologies for InNetScheduler: Load-aware Optimizer mitigates possible conflicts of ECT. Relaxated ECT Scheduler boosts the in-network computation of ECT scheduling. End-to-End Determinism Guarantee assures extremely low jitter of scheduling computation and data exchange.
- We conduct extensive experiments on different topologies and problem sizes to validate InNetScheduler's capacity. Compared with SOTA methods, InNetScheduler improves the network throughput of TCT and ECT by 7% and 17%, respectively. In addition, we have implemented a set of hardware-software co-design TSN testbeds supporting InNetScheduler paradigm. The successful operation on a real-world topology demonstrates InNetScheduler 's practicality.

II. Preliminary

A. Switch Model of TSN

Fig. 2 highlights the fundamental differences between traditional Ethernet switch and TSN switch, with the latter demonstrating superior precision in data packet control by reserving specific time slots for potentially critical traffic. Leveraging the precise time synchronization specified in IEEE 802.1AS [13], all switches within a TSN system can attain nanosecond-level time synchronization. With the presence of an accurate global clock, TSN switches can allocate transmission resources for critical traffic within the system using a Gate Control List (GCL), which can accurately manage the timing of data packet transmission for each queue.

Different operating logics of GCL correspond to distinct traffic shaping mechanisms. The two most prevalent shaping mechanisms in TSN are Time-Aware Shaper (TAS) [14] and Cyclic Queuing and Forwarding (CQF) [15]. In TAS, GCL operates at the granularity of per-packet. Implementing the TAS mechanism necessitates meticulous consideration of the sequence and timing of each data packet's entry and exit from the queue. As depicted in Fig. 2, the time slot t from 4 to 9 on the TAS switch is specifically reserved for a data packet in queue q3.

CQF is an updated protocol proposed by TSN working group. In CQF, GCL manages key traffic at the granularity of per-queue. As seen in Fig. 2, the parity and odd queues to alternately perform enqueue and dequeue operations, so CQF can ensure that data packets are sent from the upstream node in one time slot, and the data packets are received in the downstream node in the same time slot, and the data packets are sent out in the next time slot.

Beyond that, researchers have recently tried to enhance TSN's switching capability from various perspectives. Tan *et al.* [16] have amalgamated TSN technology with determinism at the IP layer. Wang *et al.* [17] have incorporated edge com-

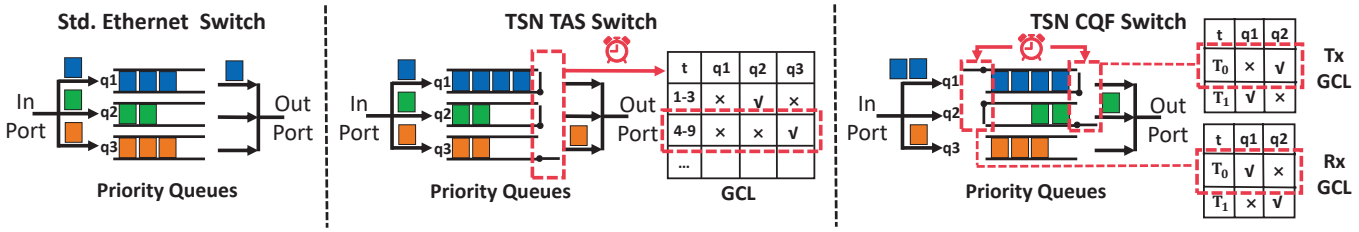
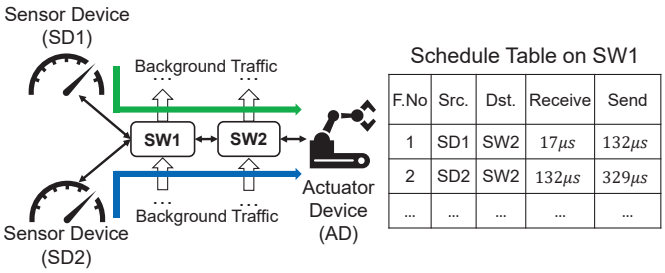


Fig. 2: Comparison of three typical switch models.



F.No	Src.	Dst.	Receive	Send
1	SD1	SW2	$17\mu s$	$132\mu s$
2	SD2	SW2	$132\mu s$	$329\mu s$
...	...	...	...	...

Fig. 3: TSN scheduling example.

puting into the TSN framework. These efforts are orthogonal to and can be integrated in InNetScheduler.

B. TSN Scheduling Problem and Formulation

Fig. 3 illustrates an example of TSN scheduling. TSN system under consideration comprises two sensor devices, two TSN switches, an actuator device and several background traffic flows. The sensors intermittently or abruptly transmit critical data flows to the actuator device. Given that each physical link can accommodate only a single data flow at any given moment, ensuring real-time and deterministic data flow involves pre-planning designated forwarding paths and reserving time slots. This ensures exclusive forwarding resources at specific temporal and spatial points. Analogous to classic scheduling dilemmas such as train timetable coordination, the TSN scheduling issue is an NP-Hard problem.

We represent the network topology as a directed graph $\mathcal{G}(V, L)$, with switches and end nodes serving as the nodes V, and physical links connection between $v_a, v_b \in V$ can be modeled as bidirectional edges $l_i = (v_a, v_b), l_j = (v_b, v_a) \in L$. Moreover, we employ a five-dimensional vector$(src, dst, period, len, MD)$ to denote the critical data flows requiring scheduling, these five elements represent the source, destination, period, packet length and maximum allow delay of critical traffic, respectively. Drawing from prior research and theoretical computations, we segment continuous time into discrete time slots for allocation, each slot spanning 250 µs and the length of CQF queue for critical traffic is 10 [15], [18]. In a network environment with a bandwidth of 1000Mbps, this implies that 120 µs are allocated for the transmission of critical data frames, and the remaining 130 µs are dedicated to background traffic.

TSN scheduling is a path and time slot assignment process with the following constraints: (1) Temporal Constraint: Each flow should transmit packets in accordance with a specified period, and the end-to-end delay must not exceed MD; (2) Queue Resource Constraint: The utilization of resources in each queue must not surpass the queue's capacity; (3) Adjacent Link Constraint: The sequence of time slot indices where a single flow converges along the forwarding path must be both incremental and contiguous; (4) Flow Isolation Constraint: Packets from distinct data flows must not interleave on the same link. These constraints collectively constitute an NP-Hard combinatorial optimization problem, which can be represented as either an Integer Linear Programming (ILP) or Satisfiability Modulo Theories (SMT) problem.

III. Method

A. Overview of InNetScheduler

InNetScheduler tackles the above challenges through software and hardware co-design. Fig. 4 sketches the architecture of InNetScheduler. We have observed that TCT exhibits predictable and static characteristics, allowing TCT scheduling operating in CNC offline, to prepare for ECT scheduling and maximize throughput. On the other hand, ECT is characterized by random and dynamic occurrences, needed to be calculated in real time when the traffic arrives.

1) TCT scheduling: As shown in the Fig. 4, TCT scheduling phase takes place in CNC (§III-B). The algorithm has two key parts: first, all TCT is scored and sorted by a Enhanced XGBooster Sorter, which uses a gradient boosting strategy and generates an optimized scheduling order to improve the throughput of critical traffic (§III-B1); the second part is Load Balance Module that ensures TCT is evenly distributed throughout the network, reserving as much balanced resources as possible for ECT scheduling(§III-B2). The scheduler needs to meet the various constraints mentioned above. After TCT scheduling phase, TCT scheduling table is generated, serving as the basis for subsequent ECT scheduling.

2) ECT scheduling: ECT scheduling is performed in switches online while traffic arrives. ECT scheduling needs to consider network throughput as well as real-time performance. We leverage the latest commercial Xilinx-Zynq7000 [19], a computing platform launched by Xilinx, to implement InNetScheduler through software and hardware co-design. As seen in Fig. 4, Zynq consists of a processing system (PS, for software development) and user-programmable logic (PL, for hardware design) two modules.

Fig. 4 demonstrates the architecture of ECT scheduling.The inputs to the ECT scheduling are the network topology, the TCT scheduling table, and ECT flows. The ECT scheduling consists of two parts: Fast Load Balance Module selects

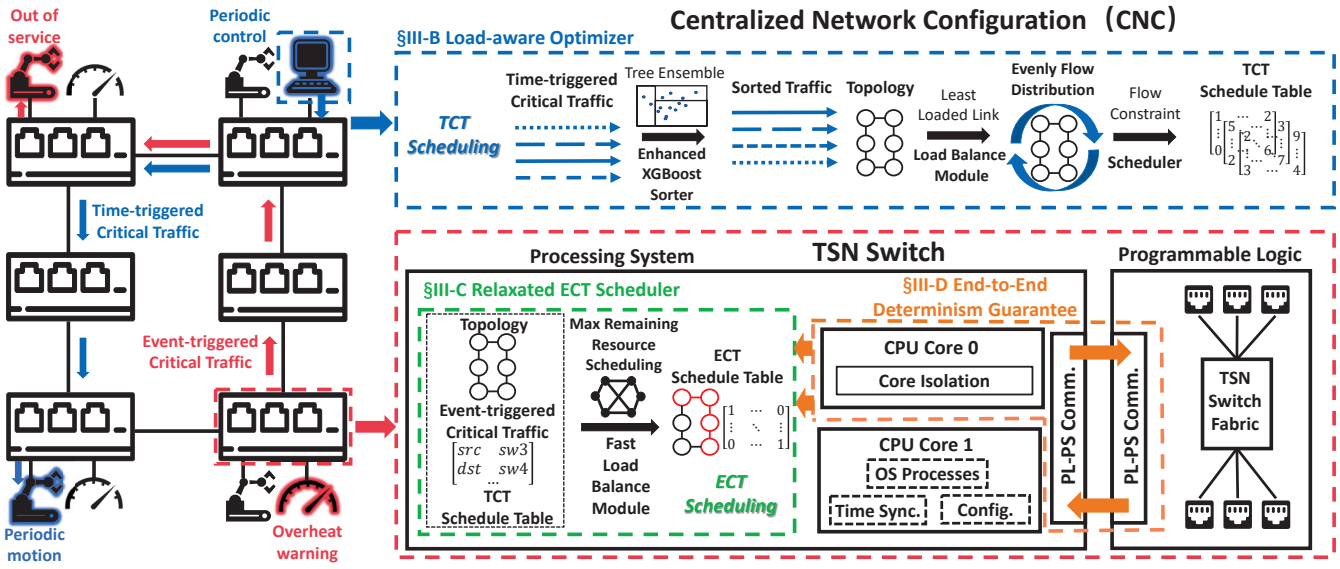


Fig. 4: Overview of InNetScheduler.

appropriate links in space, and Max Remaining Resource Scheduling plans reasonable slots in time. These designs greatly accelerate the execution of the ECT scheduling and ensure that the throughput of ECT is maintained at a high level. (§III-C). To ensure scheduling determinism, we have made specific hardware designs, to guarantee deterministic resources for ECT scheduling computation and data exchange in switches(§III-D).

B. Load-aware Optimizer

The network throughput for critical traffic is a key performance indicator of TSN systems. In practical industrial settings, scheduling tasks can be exceedingly complex. Based on our study of glass factories, the critical traffic in an industrial production line can amount to thousands. Due to the in-network scheduling paradigm, the ECT scheduling in switches is independent of each other. Therefore, it is easy to cause preemption on some key links with few resources, resulting in scheduling failure. Consequently, throughput of ECT can not be assured.

To address the aforementioned challenges, we have designed modules for both TCT and ECT scheduling to ensure a high network throughput, its core is network load sharing. TCT scheduling employs a traffic sorter based on gradient boosting theory, in conjunction with load balancing, to prepare for ECT scheduling. This approach facilitates the even distribution of traffic across the network, thereby laying the groundwork for ECT scheduling. For details on ECT scheduling, please refer to the §III-C.

1) Enhanced XGBoost Sorter: For any scheduling algorithm that does not completely traverse the solution space, the input order of the flow is crucial to the number of flow entries to be solved, and many studies have proved this point. The XGBoost regressor enhances the predictive accuracy of the model by iteratively constructing and integrating multiple decision tree models, where each new model is designed to rectify the prediction error of its predecessor. The procedure of gradient boosting in XGBoost is illustrated in Alg. 1:

Algorithm 1: Enhanced XGBoost Algorithm

Input: Massive Regularized Training Data$\{(x_i, y_i)\}_{i=1}^{N}$, Loss Function $L(y, F(x))$, Weak Learners(Trees) W and Learning Rate α

Initialize Model $\hat{f}_{(0)}(x) = \underset{\theta}{argmin} \sum_{i=1}^{N} L(y_i, \theta)$;

for $m = 1$ *to* M **do**

`// 1.compute gradient & hessians`

gradient $\hat{g}_m(x_i) = [\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)^2}]_{f(x)=\hat{f}_{(m-1)}(x)}$;

hessians $\hat{h}_m(x_i) = [\frac{\partial^2 L(y_i, f(x_i))}{\partial f(x_i)^2}]_{f(x)=\hat{f}_{(m-1)}(x)}$;

`// 2.fit next base learner`

$\hat{\phi}_m = \underset{\phi \in \Phi}{argmin} \sum_{i=1}^{N} \frac{1}{2} \hat{h}_m(x_i)[-\frac{\hat{g}_m(x_i)}{\hat{h}_m(x_i)} - \phi(x_i)]^2$;

$\hat{f}_m(x) = \alpha \hat{\phi}_m(x)$;

`// 3.update model`

$\hat{f}_{(m)}(x) = \hat{f}_m(x) + \hat{f}_{(m-1)}(x)$;

`// 4.early stopping`

check if to execute early stopping

end

return $\hat{f}_{(M)}(x) = \sum_{m=0}^{M} \hat{f}_m(x)$;

We enhance XGBoost regressors with massive data and take "early stopping" to prevent overfitting. It's important to note that the training data for the XGBoost regressor is derived from simulations. We inject a substantial volume of traffic into the virtual network topology and employ a simple greedy algorithm for scheduling these tasks. Upon the occurrence of the first unschedulable traffic failure, we record the success or failure of scheduling of all subsequent traffic. The score for all traffic is calculated as 100 times the ratio of the schedulable

link length of the traffic to the total path length. This design aims to adjust the traffic order to increase network throughput.

2) Load Balance Module: The underlying principle of the load balancing module design is to equalize link loads to facilitate potential future scheduling. In this study, we utilized Python Networkx [20] to construct a model that mirrors actual network characteristics, and based on this, we implemented a load balancing design for forwarding. The algorithm for finding the path with the least load can be reduced to the single-source shortest path algorithm, which can be solved by Dijkstra Algorithm. The implementation in this paper uses a Fibonacci heap. The pseudocode for the complex balancing module is presented in Alg. 2:

Algorithm 2: Load Balance Algorithm

Input: Sorted Flow List L, Network N
Initialization Forward_Table T a hash table ;
while *there are unprocessed flows in* L **do**
  extract a flow f from L;
  select the path p with the minimum load for f based on N;
  T insert $\{f : p\}$;
  update N based on path p;
end
return *Forward_Table* T;

C. Relaxated ECT Scheduler

In contrast to commercial applications, industrial applications necessitate a higher degree of real-time responsiveness and determinism (e.g., sub-millisecond versus tens of milliseconds latency and jitter). While Zynq has furnished specific computational units for executing tasks (e.g., 2 A-Cores), the intricacy of task scheduling and the relative paucity of computational resources pose a substantial challenge in guaranteeing real-time in-network computation.

To address the challenge of real-time computation, we propose Fast Load Balance Module for path selection and Max Remaining Resource Scheduling for time slot management. The design of these two modules is can be fast calculated, paying great attention to computing efficiency.

1) Fast Load Balance Module: As mentioned in §III-B2, the computational complexity of normal Load Balance Module is $\mathcal{O}(k(m + nlogn))$, where m is the number of paths in the graph, n is the number of nodes and k is the total flow number. But such complexity is also a burden for switches. So we adopt the idea of exchanging space for time, and record all possible situations, and do not update the ECT traffic in real time, so as to achieve the algorithm complexity of $\mathcal{O}(1)$.

2) Max Remaining Resource Scheduling: Algorithm schedule traffic line by line. Fig. 5 presents a schematic representation of Max Remaining Resource Scheduling for single flow. The figure illustrates a basic topology along with a straightforward demand for ECT. The traffic will pass through Fast Load Balance Module, where we find the optimal path. We need to transmit the traffic packet from the src to dst

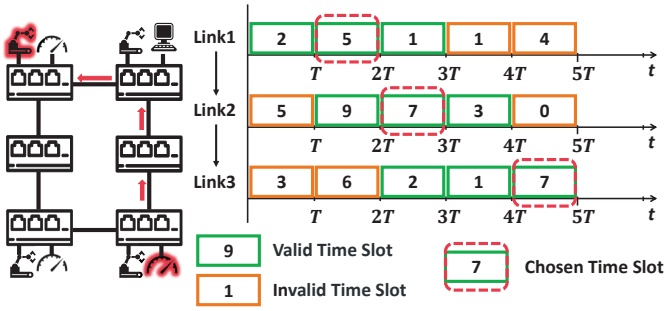


Fig. 5: Max Remaining Resource Scheduling Diagram.

within the deadline, which is assumed to be five time slots. During the transmission process, the traffic passes through three links. The available resources for each link in these five time slots are shown in the figure. For example, the five time slots for link1 are $(2, 5, 1, 1, 4)$. We then reduce this problem to finding the maximum remaining resources, and we use dynamic programming (DP) for quick solution. This leads to a relaxed solution. It should be noted that the time slot of the previous link we choose must be earlier than the subsequent continuous time slots. Alg. 3 shows the specific process of DP. We can simply analyze the complexity of the algorithm to be $\mathcal{O}(ktl)$, where k represents the number of traffic flows, t is the deadline time and l is the path length.

Importantly, we have made specific modify to the IEEE 802.1 Qch standard to optimize throughput, as the time slots for ECT on adjacent links are not sequential. Specifically, we have engineered buffer queues in hardware to accommodate those ECT that does not require immediate transmission. Despite these modifications, the updated switch remains compliant with the TSN shaping mechanism.

Algorithm 3: Max Remaining Resource Scheduling

Input: Resource Matrix R
Output: max_sum and $optimal\_path$
Define dp and $path$ as two-dimensional arrays with the same size as Resource Matrix;
for *each column* j *from 1 to columns* **do**
  for *each row* i *from 0 to rows* **do**
    $dp[i][j] \leftarrow max\_val + matrix[i][j]$;
    $path[i][j] \leftarrow max\_index$;
  end
end
$max\_sum \leftarrow max(dp[-1])$;
$max\_row \leftarrow maxIndex(dp[-1])$;
Define $optimal\_path \leftarrow []$;
Append max_row to $optimal\_path$;
for *each column j from columns-1 to 1 (backward)* **do**
  $max\_row \leftarrow path[max\_row][j]$;
  Append max_row to $optimal\_path$;
end
Reverse optimal_path;
return max_sum *and* $optimal\_path$;

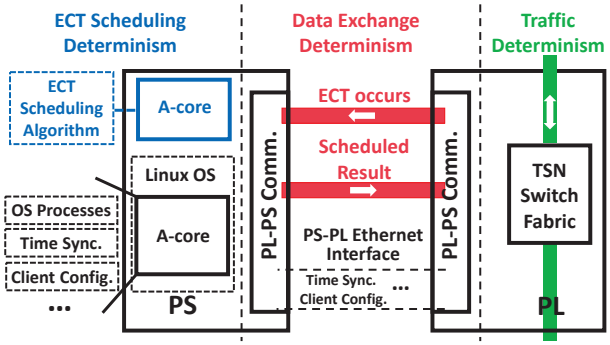


Fig. 6: End-to-End Determinism Guarantee.

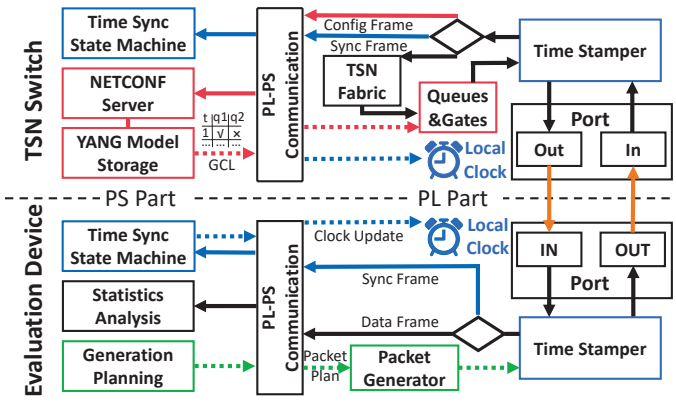


Fig. 7: Architecture of TSN testbeds.

D. End-to-End Determinism Guarantee

The TSN system strives for ultimate determinism, necessitating that each of its components, including the scheduling algorithm, must exhibit deterministic behavior. Given the inherent resource management and task scheduling logic of the operating system, merely integrating the scheduling algorithm within the operating system does not assure determinism. Furthermore, data exchange between the PS and PL via the network port can induce substantial latency and jitter.

In InNetScheduler, we tackle the above challenge and ensure the determinism of scheduling tasks through handcrafted and detailed resource and computation management. As seen in Fig. 6, we categorize the end-to-end delay determinism into the ECT scheduling computation determinism, data exchange determinism and traffic determinism. For these parts, we propose isolated core for ECT scheduling computation determinism, bypassing uncertain OS scheduling, and Twin DMA to ensure data exchange determinism. TSN Traffic Shaping Mechanism ensures the traffic determinism in the network.

1) ECT Scheduling Computation Determinism Guarantee: Guaranteeing the execution latency of time-sensitive tasks operating on Linux OS poses a challenge. Given the diversity of computational tasks, OS on the TSN chip frequently opts for TSOS (Time Sharing Operating System, *i.e.*, Debian), thereby introducing an element of uncertainty into the computation. This is primarily due to numerous background processes concurrently running and vying for computational resources with time-sensitive tasks within the operating system. To mitigate the influence of these processes, we have reserved one A-Core (indicated by the blue box in Fig. 6) for the execution of each time-sensitive task. In our implementation, we achieved A-Core isolation by employing the boot parameter `isolcpus=<cpu number>` during the construction of the Linux OS.

2) Data Exchange Determinism Guarantee: We employ Direct Memory Access (DMA) technology to manage the process of intermediate data exchange. In contrast to existing techniques, such as the PL-PS Ethernet interface-based solution [21], our proposed Twin DMA provides a dedicated data transmission channel for the scheduling data of ECT, thereby ensuring deterministic internal PL-PS data exchange within the switch. Additionally, the utilization of the DMA channel facilitates data transfer to or from the device, resulting in significantly less CPU overhead compared to network-based solutions. Should DMA not be utilized for data transmission, the task would fall entirely to the CPU. However, the CPU's speed in accessing memory does not match that of DMA, and it would be unable to execute other computational tasks concurrently.

IV. Implementation

To validate the accuracy, real-time responsiveness, and determinism of InNetScheduler, we employed a Xilinx 7000 FPGA board to develop a suite of TSN switches and evaluation devices with hardware-software co-design, and incorporating InNetScheduler into the system. As shown in Fig. 7, the hardware component comprises modules for gating and high-precision packet transmission, while the software component encapsulates complex processing logic, such as time synchronization and user configuration. Our system supports 802.1Qch, 802.1Qbv, 802.1Qcc [22], and 802.1AS. Precise time stamper and GCL were executed on the hardware. So it can facilitated time synchronization at the nanosecond level as well as sub-microsecond-level packet jitter. This TSN system also supports the collection, statistics, and analysis of data packets, meeting the system's network detection and performance evaluation needs. This work bridges the gap between theoretical understanding and practical application of TSN.

V. Evaluation

A. Experiment Setup

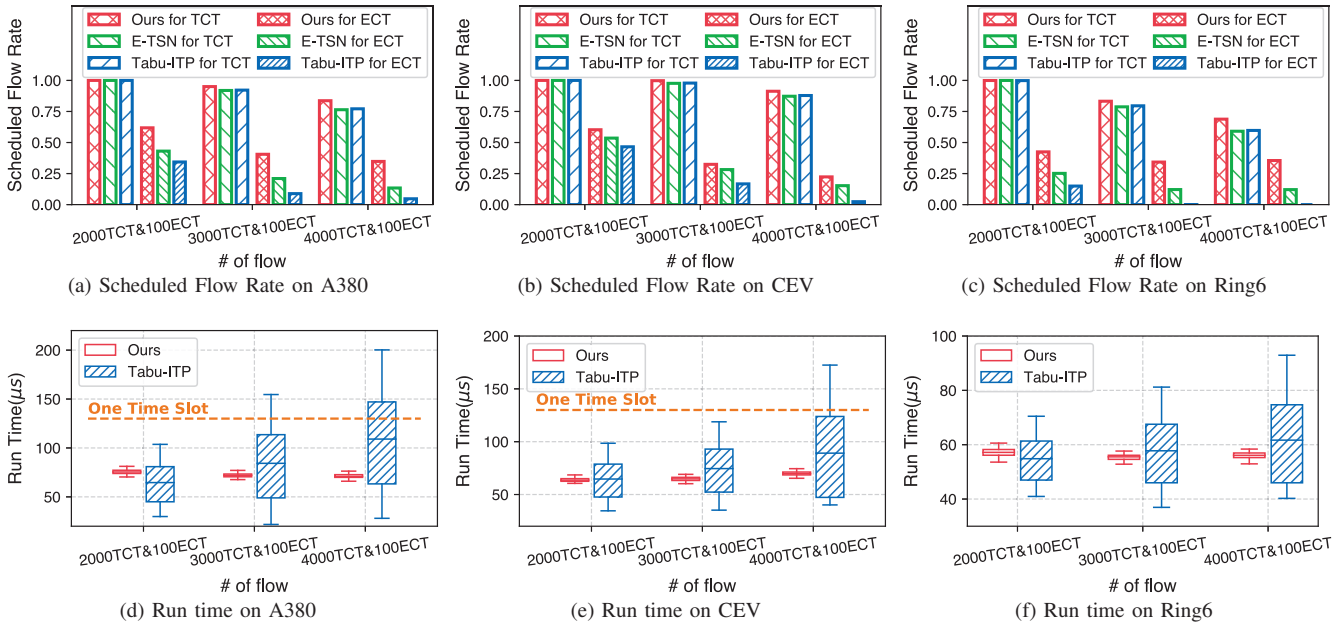
1) Network Topology: We select three typical topologies to represents complex use cases in industrial scenarios:

A380 topology is a simplification of the control network of the Airbus A380 [23]. The network topology consists of 9 switches and 8 devices.

CEV topology is an abstraction of the control network of the Orion spacecraft, a Crew Exploration Vehicle from NASA, consisting of 9 switches and 9 devices [24].

Ring6 topology is also common in industrial control networks [25], [26]. It consists of 6 switches and 6 devices.

The aforementioned three representative topologies are shown in Fig. 9, where the ellipses represent switches and the rectangles represent devices.



(a) Scheduled Flow Rate on A380 (b) Scheduled Flow Rate on CEV (c) Scheduled Flow Rate on Ring6

(d) Run time on A380 (e) Run time on CEV (f) Run time on Ring6

Fig. 8: Overall performance of InNetScheduler on different network topologies.

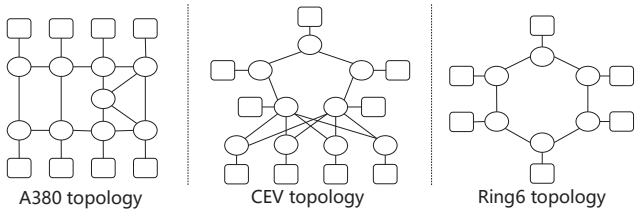


Fig. 9: Topologies under test.

2) Flow Requirement: The requirements for TSN flows are generated randomly in compliance with the IEC/IEEE 60802 [27], the TSN profile designated for industrial automation. We randomly select two devices for each flow as the source and destination. TCT's period and maximum end-to-end delay are random chosen from $\{8\,\text{ms},\ 16\,\text{ms}\}$. ECT is non-periodic, and its maximum end-to-end delay are random chosen from $\{4\,\text{ms},\ 8\,\text{ms},\ 16\,\text{ms}\}$. The payload length is uniformly distributed between 1 and 2 packets.

3) Resource Setting: We choose CQF as the traffic shaping protocol. Resource settings in our experiments include link bandwidth, queue length, and global time slot size. Similar to most studies, we set the link bandwidth to 1000Mb/s, the length of each CQF queue to 10 and time slot size to $250\,\mu\text{s}$.

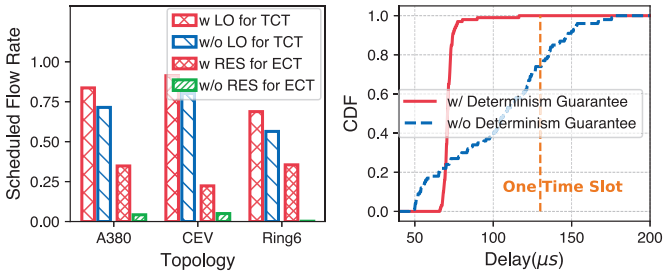
4) Metrics: We employ scheduled flow rate, defined as the proportion of successfully scheduled traffic in a given set of flows, as a metric to evaluate the efficacy of the scheduling algorithm. Additionally, we measure the run time of scheduling algorithm on a real TSN switch to assess its real-time and deterministic performance.

5) Baselines: We compare the InNetScheduler 's performance with two most relevant state-of-the-art schedule algorithm, E-TSN [7] and Tabu-ITP [18], to evaluate the algorithm performance. E-TSN models ECT as TCT, and reserves some resources for ECT based on the model when scheduling in CNC. For comparison purposes, we use Tabu-ITP, which use both Tabu Search for TCT scheduling in CNC and ECT scheduling in TSN switch, and also include End-to-End Determinism Guarantee technology for Tabu-ITP.

B. Overall Performance

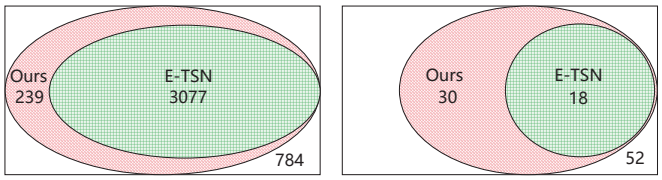
Fig. 8 demonstrates the overall performance of InNetScheduler. For each experiment, we randomly generate 100 scheduling problems and then record the scheduled flow rate and run time on a TSN switch among different methods. Each scheduling problem consists of {2000, 3000, 4000} TCT and 100 ECT. To sum up, InNetScheduler achieves the highest scheduled flow rate among the counterparts and ensures real-time and deterministic performance which is not guaranteed by other methods.

Fig. 8a, 8b, 8c display the scheduled flow rate of different method. At the highest load of the three topologies, the TCT scheduled flow rate of InNetScheduler is on average 7.0% higher than E-TSN and 6.7% higher than Tabu-ITP. When comparing ECT flow schedules, the advantage of InNetScheduler is more pronounced, with its scheduled flow rate at the highest load being 17.3% and 28.5% higher than E-TSN and Tabu-ITP, respectively. It is noteworthy that all three methods exhibit relatively subpar performance on the Ring6 topology. This is attributed to the fact that the Ring6 topology, comprising merely six switches, is incapable of managing a load exceeding 3000 flows. E-TSN and Tabu-ITP each exhibit unique strengths regarding the scheduling capabilities for TCT and ECT. For TCT, E-TSN sacrifices a degree of scheduled flow rate due to its reservation of resources for ECT. However, these reserved time slots prove to be pivotal in ECT scheduling, making E-TSN significantly more effective than



(a) Breakdown on software (b) Breakdown on hardware

Fig. 10: ablation study



(a) Comparison on TCT (b) Comparison on ECT

Fig. 11: Scheduled flows on the same set of problems.

Tabu-ITP in this aspect. Thus, a trade-off exists between E-TSN and Tabu-ITP in this context.

Fig. 8d, 8e, 8f illustrates the run time results in different topologies. Given that E-TSN takes a modeling approach and totally deploy in CNC, its scheduling run time for ECT is not taken into account. We also based on the CQF protocol, the deadline for ECT computation can be determined: $250\mu s - 10 * \frac{1500B}{1000Mb/s} = 130\mu s$ (Maximum Transmission Unit size is 1500B). That is, the lower time limit for sending background traffic. If the scheduling time is within 130 µs, then the next forwarding result is computed in every time slot, and the real-time nature of the scheduling ensures deterministic data forwarding. InNetScheduler consistently meets the requirements for ECT scheduling computation. However, a significant portion of computational tasks in the Tabu-ITP method fails to complete within the specified time, potentially causing critical data frame to miss a time slot, so can't be used in actual production.

In comparison to Tabu-ITP, InNetScheduler demonstrates superior deterministic performance, with jitter confined within 10 µs, thus marking an improvement by an order of magnitude over Tabu-ITP, whose jitter is above 180 µs. Between A380, CEV and Ring6 topology, the median scheduling run time for Tabu-ITP roughly exhibits a decreasing relationship. Meanwhile, the jitter of Tabu-ITP's run time for ECT is also a similar relationship. This may be due to the algorithmic decision of Tabu search, which requires frequent domain selection and movement when the topology is complex and the number of feasible solutions is low, resulting in a prolonged and fluctuating computation time. In contrast, the complexity of our algorithm is completely determined by the path length of the flow, the maximum allow delay, and the number of flows, so run time of our algorithm is highly deterministic.

C. Ablation Study

We conduct several experiments to understand the effectiveness of each module in InNetScheduler.

1) Software: We conducted experiments by separately eliminating Load-aware Optimizer (LO) from the TCT scheduling and Relaxated ECT Scheduler (RES) from ECT scheduling, both are replaced with greedy scheduling. Fig. 10a illustrates the efficacy of the two component. The findings reveal that the removal of the sorter results in a decrease in the scheduled flow rate of TCT by 12.2%, 11.1%, and 12.4% across the three topologies. Conversely, the elimination of the RES component leads to a reduction in the scheduled flow rate of ECT by 30.5%, 21.8%, and 35.3%. These results underscore the significant role of software algorithm design in enhancing the scheduled flow rate of the TSN system. Similar to the overall experimental results in §V-B , after removing RES we designed, ECT on Ring6 topology is almost unschedulable.

2) Hardware: We evaluate the run time of InNetScheduler's ECT scheduling with and without End-to-End Hardware Guarantee. As a control group, we replace the sensing DMA with a PS-PL Ethernet interface and turn off the core isolation, put the interface and CPU at 50% load. Fig. 10b indicates a significant increase in computational delay and jitter in the control group's experiment, potentially compromising the overall system availability. Given the previously established benchmark of 130 µs, approximately 26% of the data fails to complete computation within the allotted time. The reason for these task failures is likely to be due to the lack of exclusive resources, which caused the transmission between PS-PL to be blocked or waited for a long time before the CPU's time slice. This underscores the efficacy of the hardware module in ensuring the determinism of ECT scheduling.

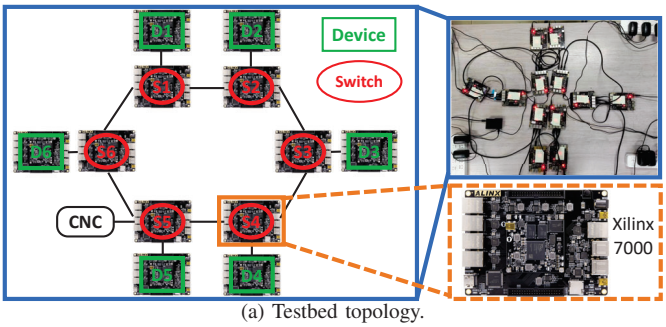
D. Deep Dive of InNetScheduler

To better understand the scheduled flow rate of InNetScheduler, we conduct a more detailed analysis of scheduling results. We conduct an experiment using a representative scheduling task, consisting of 4000 TCT and 100 ECT, and record the scheduling results on InNetScheduler and E-TSN .

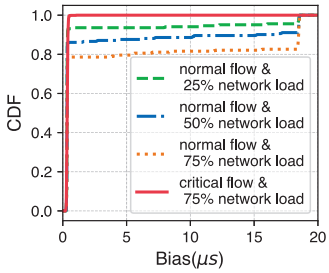
Fig. 11a and Fig. 11b depict the relationships between the sets of scheduled flows for both TCT and ECT. In the comparison of scheduled flow rate between these two types of flows, InNetScheduler's scheduled flow set fully encompasses that of E-TSN. This implies that all flows solvable by E-TSN can invariably be resolved by InNetScheduler, and issues that InNetScheduler cannot address are likewise unsolvable by E-TSN, indicating that InNetScheduler fully covers the capacity of E-TSN. Following similar experiments, we discovered that InNetScheduler also fully encompasses the capacity of Tabu-ITP. Consequently, InNetScheduler can replace other existing methods in industrial networks without any adverse effects.

E. Testbed Deployment

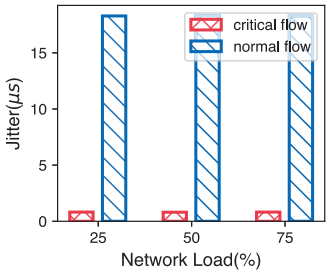
To verify the schedule calculated by InNetScheduler, we deploy its TCT scheduling into CNC and ECT scheduling into switches. We configured the switches and devices in the previously mentioned Ring6 topology as shown in Fig. 12a



(a) Testbed topology.



(b) Bias of different flows.



(c) Jitter of different flows.

Fig. 12: Testbed deployment results.

and performed real scheduling tests. The comparative scheme involved normal traffic, where we opted to transmit all critical traffic as normal traffic at a lower priority. During each cycle, we dispatched 20 specific data frames, accompanied by background traffic at rates of 250Mb/s, 500Mb/s, and 750Mb/s. For each set of conditions, we tested data over 1000 cycles and documented the bias (difference between scheduled delay) and jitter (delay's variation).

Fig. 12b illustrates the cumulative distribution function of traffic bias under different network loads. Across the three scales of background traffic, the average delay is 2.5%, 4.9%, and 8.0% lower than that of regular traffic. Furthermore, 6.3%, 13.9%, and 21.4% of data frames within the regular traffic are delayed due to the background traffic. Fig. 12c illustrates the jitter associated with the transmission of two types of traffic data. Experimental results indicate that the critical traffic, managed by the scheduling algorithm, remains entirely undisturbed by the background traffic. The network jitter for the scheduled critical traffic is confined within $1\,\mu$s, whereas it reaches up to $18\,\mu$s for normal traffic. The experiment demonstrates that both types of critical traffic, when scheduled via InNetScheduler, can ensure a transmission delay at the microsecond level and transmission jitter at the sub-microsecond level.

VI. Related Work

The issue of TSN scheduling has been extensively researched. Existing scheduling algorithms can be categorized into two types based on whether computation commences before the arrival of traffic: Offline and Online Scheduling. From this perspective, InNetScheduler is equivalent to an offline-online fusion algorithm, where TCT scheduling belongs to offline and ECT scheduling belongs to online.

A. Offline TSN Scheduling

Offline Scheduling necessitates the assumption that both the network topology and critical traffic information are static. Conventional Offline Scheduling algorithms typically model the scheduling problem as a standard Satisfiability Modulo Theories (SMT) [28]–[30] or Integer Linear Programming (ILP) problem [31], [32]. which are then solved using a solver, as demonstrated in the research conducted by scholars such as Craciunas *et al.* [33]. While the use of SMT or ILP provides the highest network throughput, it often requires a comprehensive exploration of the entire solution space, resulting in slower execution speeds. The second category of commonly used algorithms includes heuristic algorithms, such as Tabu Search and Genetic Algorithms [18], [34], [35]. These algorithms enhance execution speed by trimming the solution space using artificial rules, albeit at the expense of some network throughput. With the advancement of Deep Learning, numerous methods for addressing scheduling problems have emerged [36]–[38]. This type of method also adopts the cutting of the solution space, but the cutting is carried out by the knowledge learned by the neural network.

B. Online TSN Scheduling

In certain instances, both the network topology and data transmission requirements are subject to dynamic changes. Consequently, scheduling algorithms must incorporate newly emerged data flows into the existing timetable as swiftly as possible. Some approaches rely on specific network characteristics and application scenarios to design algorithms [39]–[41]. For instance, Huang *et al.* proposed an Online Scheduling and routing method specifically designed for industrial automation scenarios [42]. Other approaches employ heuristic search to maximize the quantity of data flows that can be added to the entire network [43], [44]. However, these approaches also face the challenge of overly stringent assumptions about sequential data flow emergence, resulting in lower network throughput compared to offline methods.

VII. Conclusion

In this work, we propose InNetScheduler, a novel paradigm for TSN scheduling. InNetScheduler decomposes the ECT scheduling task from the CNC and places it to TSN switches. By reducing potential ECT conflicts, the Load-aware Optimizer ensures efficient operations. With the Relaxated ECT Scheduler, in-network computation for ECT scheduling is accelerated, and the End-to-End Determinism Guarantee assures low jitter for reliable performance. The evaluations demonstrate that InNetScheduler is fast and scalable, and improves network throughput significantly. We envision InNetScheduler as a critical step on TSN practice in the real industrial production.

Acknowledgment

This work is supported in part by the National Key Research Plan under grant No. 2021YFB2900100, the NSFC under grant No. 62372265, No. 62202262, No. 62302254, No. 62272462 and No. 62202263.

REFERENCES

[1] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," *Business & information systems engineering*, vol. 6, pp. 239–242, 2014.

[2] D. Bruckner, M.-P. Stănică, R. Blair, S. Schriegel, S. Kehrer, M. Seewald, and T. Sauter, "An introduction to opc ua tsn for industrial communication systems," *Proceedings of the IEEE*, vol. 107, no. 6, pp. 1121–1131, 2019.

[3] T. J. Burke. (2023) The semiconductor industry's future aligns with tsn. [Online]. Available: https://www.electronicdesign.com/technologies/industrial/article/21260404/cclink-partner-association-the-semiconductor-industrys-future-aligns-with-tsn

[4] A. Albert *et al.*, "Comparison of event-triggered and time-triggered concepts with regard to distributed control systems," *Embedded world*, vol. 2004, pp. 235–252, 2004.

[5] X. Ge, Q.-L. Han, X.-M. Zhang, and D. Ding, "Dynamic event-triggered control and estimation: A survey," *International Journal of Automation and Computing*, vol. 18, no. 6, pp. 857–886, 2021.

[6] T. Li, Q. Qiu, and C. Zhao, "A fully distributed event-triggered communication strategy for second-order multi-agent systems consensus," *arXiv preprint arXiv:2011.11882*, 2020.

[7] Y. Zhao, Z. Yang, X. He, J. Wu, H. Cao, L. Dong, F. Dang, and Y. Liu, "E-tsn: Enabling event-triggered critical traffic in time-sensitive networking for industrial applications," in *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2022, pp. 691–701.

[8] S. Yin, S. Wang, Y. Huang, R. Huo, T. Huang, and Y. Liu, "Critical event-triggered flows tolerance in time-sensitive networks," in *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, 2021, pp. 1–6.

[9] H. Inc. (2023) Ie4320 comware v7 series rackmount industrial switches. [Online]. Available: https://www.h3c.com/cn/Products_And_Solution/InterConnect/Products/Switches/Products/IE_MS/H3C_IE/IE/IE4320/

[10] N. Semiconductors. (2023) Layerscape 1028a applications processor. [Online]. Available: https://www.nxp.com/products/processors-and-microcontrollers/arm-processors/layerscape-processors/layerscape-1028a-applications-processor:LS1028A

[11] B. Inc. (2023) The broadcom strataconnect bcm53570. [Online]. Available: https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm53570

[12] S. Kianpisheh and T. Taleb, "A survey on in-network computing: Programmable data plane and technology specific applications," *IEEE Communications Surveys & Tutorials*, 2022.

[13] *Timing and Synchronization for Time-Sensitive Applications*, IEEE Std. 802.1AS, 2020.

[14] *Enhancements for Scheduled Traffic*, IEEE Std. 802.1Qbv, 2015.

[15] *Cyclic Queuing and Forwarding*, IEEE Std. 802.1Qch, 2015.

[16] W. Tan and B. Wu, "Long-distance deterministic transmission among tsn networks: Converging cqf and dip," in *2021 IEEE 29th International Conference on Network Protocols (ICNP)*. IEEE, 2021, pp. 1–6.

[17] Z. Wang, J. Xu, X. Wang, X. Zhuge, X. He, and Z. Yang, "Industrial knee-jerk: In-network simultaneous planning and control on a tsn switch," in *Proceedings of the 21st Annual International Conference on Mobile Systems, Applications and Services*, 2023, pp. 356–369.

[18] J. Yan, W. Quan, X. Jiang, and Z. Sun, "Injection time planning: Making cqf practical in time-sensitive networking," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 616–625.

[19] Xilinx. (2021, Jul.) Socs with hardware and software programmability. [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html

[20] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep., 2008.

[21] X. Inc. (2022) Mpsoc ps and pl ethernet example projects. [Online]. Available: https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/478937213/MPSoC+PS+and+PL+Ethernet+Example+Projects

[22] *Stream Reservation Protocols (SRP) Enhancements and Performance Improvements*, IEEE Std. 802.1Qcc, 2018.

[23] F. Boulanger, D. Marcadet, M. Rayrole, S. Taha, and B. Valiron, "A time synchronization protocol for a664-p7," in *2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC)*. IEEE, 2018, pp. 1–9.

[24] D. Tămaş-Selicean and P. Pop, "Optimization of ttethernet networks to support best-effort traffic," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*. IEEE, 2014, pp. 1–4.

[25] P.N.American. (2022) Industrial topology options and profinet. [Online]. Available: https://us.profinet.com/wp-content/uploads/2019/08/Topology.pdf

[26] W.Voss. (2022) Industrial ethernet guide-network topologies. [Online]. Available: https://copperhilltech.com/blog/industrial-ethernet-guide-network-topologies

[27] *TSN Profile for Industrial Automation*, IEC/IEEE Std. 60 802, 2018.

[28] F. Pozo, G. Rodriguez-Navas, H. Hansson, and W. Steiner, "Smt-based synthesis of ttethernet schedules: A performance study," in *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*. IEEE, 2015, pp. 1–4.

[29] F. Pozo, W. Steiner, G. Rodriguez-Navas, and H. Hansson, "A decomposition approach for smt-based schedule synthesis for time-triggered networks," in *2015 IEEE 20th conference on emerging technologies & factory automation (ETFA)*. IEEE, 2015, pp. 1–8.

[30] Z. Yang, Y. Zhao, F. Dang, X. He, J. Wu, H. Cao, Z. Wang, and Y. Liu, "CaaS: Enabling Control-as-a-Service for Time-Sensitive Networking," in *IEEE INFOCOM*, 2023.

[31] 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.

[32] 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.

[33] S. S. Craciunas and R. S. Oliver, "Combined task-and network-level scheduling for distributed time-triggered systems," *Real-Time Systems*, vol. 52, pp. 161–200, 2016.

[34] M. Pahlevan and R. Obermaisser, "Genetic algorithm for scheduling time-triggered traffic in time-sensitive networks," in *2018 IEEE 23rd international conference on emerging technologies and factory automation (ETFA)*, vol. 1. IEEE, 2018, pp. 337–344.

[35] D. Hellmanns, A. Glavackij, J. Falk, R. Hummen, S. Kehrer, and F. Dürr, "Scaling tsn scheduling for factory automation networks," in *2020 16th IEEE International Conference on Factory Communication Systems (WFCS)*. IEEE, 2020, pp. 1–8.

[36] X. He, X. Zhuge, F. Dang, W. Xu, and Z. Yang, "Deep-scheduler: Enabling flow-aware scheduling in time-sensitive networking," in *IEEE INFOCOM*, 2023.

[37] H. Jia, Y. Jiang, C. Zhong, H. Wan, and X. Zhao, "Ttdeep: Time-triggered scheduling for real-time ethernet via deep reinforcement learning," in *2021 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2021, pp. 1–6.

[38] L. Yang, Y. Wei, F. R. Yu, and Z. Han, "Joint routing and scheduling optimization in time-sensitive networks using graph-convolutional-network-based deep reinforcement learning," *IEEE Internet of Things Journal*, vol. 9, no. 23, pp. 23 981–23 994, 2022.

[39] D. Yang, K. Gong, J. Ren, W. Zhang, W. Wu, and H. Zhang, "Tc-flow: Chain flow scheduling for advanced industrial applications in time-sensitive networks," *IEEE Network*, vol. 36, no. 2, pp. 16–24, 2022.

[40] Z. Pang, X. Huang, Z. Li, S. Zhang, Y. Xu, H. Wan, and X. Zhao, "Flow scheduling for conflict-free network updates in time-sensitive software-defined networks," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 3, pp. 1668–1678, 2020.

[41] Z. Feng, Q. Deng, M. Cai, and J. Li, "Efficient reservation-based fault-tolerant scheduling for ieee 802.1 qbv time-sensitive networking," *Journal of Systems Architecture*, vol. 123, p. 102381, 2022.

[42] Y. Huang, S. Wang, T. Huang, B. Wu, Y. Wu, and Y. Liu, "Online routing and scheduling for time-sensitive networks," in *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2021, pp. 272–281.

[43] Q. Yu, H. Wan, X. Zhao, Y. Gao, and M. Gu, "Online scheduling for dynamic vm migration in multicast time-sensitive networks," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 6, pp. 3778–3788, 2019.

[44] Q. Yu and M. Gu, "Adaptive group routing and scheduling in multicast time-sensitive networks," *IEEE Access*, vol. 8, pp. 37 855–37 865, 2020.