# MSS: Exploiting Mapping Score for CQF Start Time Planning in Time-Sensitive Networking

Miao Guo, *Student Member, IEEE,* Chaojie Gu, *Member, IEEE,* Shibo He, *Senior Member, IEEE,* Zhiguo Shi, *Senior Member, IEEE,* and Jiming Chen, *Fellow, IEEE*

***Abstract*—Time-Sensitive Networking (TSN), an emerging network technology, requires high-performance scheduling mechanisms to deliver deterministic service in Industry 5.0. Cyclic Queuing and Forwarding (CQF) is launched to simplify the configuration complexity of the early-stage mechanism Time-Aware Shaper (TAS) in TSN flow scheduling. Previous CQF studies adopt an inflexible incremental flow scheduling scheme, which consists of flow sorting, offset search, and resource judgment. However, we observe that flow sorting and offset search are mutually interdependent. The offset of a flow helps determine the resource status on the flow path, which can guide flow sorting. By utilizing the interaction between flow and offset, we design a novel scheduling approach that achieves high scheduling performance and time efficiency. Specifically, the proposed approach combines flow sorting and offset search together to select flow and its offset (i.e., (*flow, offset*)) simultaneously. To effectively determine the selecting priority and select the potential optimal flow-offset combination, we define a unified metric, $Mapping\ Score$, to quantify the schedulability of different flow and offset combinations. The extensive experiments demonstrate that the scheduling success rate of our proposed approach is on average 31.69% higher than the baseline and 4.57% higher than the state-of-the-art FLJ method. Moreover, it outperforms the state-of-art FLJ method by 7.62% in large-scale linear topologies, indicating its great scalability in different network scales and complex topologies.**

***Index Terms*—Industrial Internet of Things, Time-Sensitive Networking, CQF model, scheduling, mapping score.**

## I. Introduction

INDUSTRY 5.0 envisions increasing interconnectivity and intelligent automation in IIoT, which has stringent Quality of Service (QoS) requirements [1]. Timeliness, one of the most critical QoS metrics in IIoT networks [2], affects the production efficiency of many time-sensitive industries, including energy [3], smart grid [4], and telemedicine [5]. Thus, it is desirable to have a deterministic network with bounded delay and jitter in industrial systems [6].

Time-Sensitive Networking (TSN) is proposed in IEEE 802.1 Standard [7] to improve the real-time capabilities that previous Ethernet [8] lacks. TSN aims at delivering deterministic services, making it suitable for time-sensitive applications [9]. To achieve deterministic transmission, the IEEE 802.1Qbv standard [10] specifies a gate mechanism called Time-Aware Shaper (TAS) to schedule flows in TSN. TAS utilizes two switch signals (i.e., 0 and 1) to control each queue connected to the associated egress port. These control signals are generated according to the predefined scheduling scheme—Gate Control List (GCL), which requires dynamic configuration on each queue of all the network hosts and switches [8]. Especially, the configuration cost would increase significantly in large-scale networks. To simplify the configuration complexity of GCL, Cyclic Queuing and Forwarding (CQF), a simplified model, is proposed in IEEE 802.1Qch [11] standard. CQF achieves this by statically configuring the GCL [7] (see Section III).

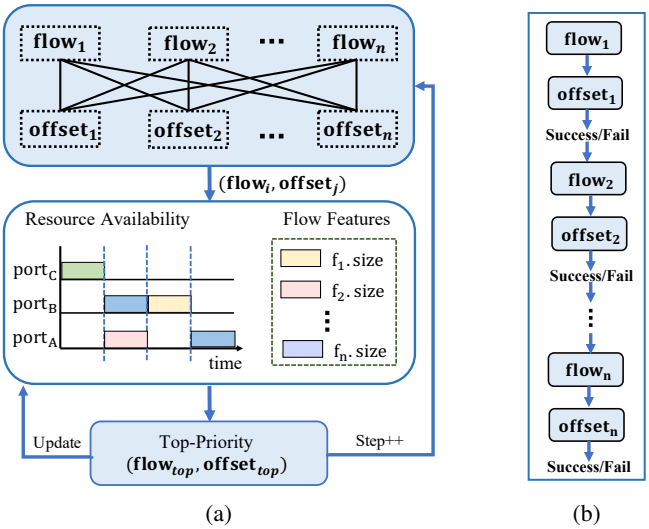



Fig. 1: Comparison between (a) the proposed approach and (b) the incremental scheduling scheme. The proposed approach utilizes the mutual interaction between flow and offset to jointly leverage the resource availability and flow features.

However, the traffic scheduling and practical implementation remain open issues in CQF. For the sake of time efficiency, the idea of incremental scheduling [6] is adopted to ease the computation intensity of solver-based algorithms. The incremental scheduling scheme is — (1) flow sorting, (2) offset search, (3) resource judgement [12]. Whereas, this serialized and inflexible scheme neglects the mutual interaction between flow sorting and offset search — offset helps determine the resource availability, which can guide flow sorting. Resource availability provides information on the load that the network can accommodate in the target flow path, for this reason, flow sorting leveraging both flow features and resource availability will exploit this information and

This work was partially supported by National Natural Science Foundation of China under Grant U1909207 and the State Key Laboratory of Industrial Control Technology, Zhejiang University, China (No.ICT2022A01). *Corresponding author: Chaojie Gu.*

M. Guo, C. Gu, S. He, Z. Shi and J. Chen are with the State Key Laboratory of Industrial Control Technology, Zhejiang University, Hangzhou, Zhejiang, 310027, China. E-mail: {gm_oct, gucj, s18he, shizg, cjm}@zju.edu.cn.

thus lead to a more even allocation of the global resource than only leveraging flow features. Since the prerequisite of resource availability is offset, offset can guide flow sorting and contribute to even resource allocation. Thus we should utilize the mutual interaction between flow sorting and offset search in scheduling for load balance level improvement. However, the incremental scheduling scheme separates flow sorting and offset search. That is, it first selects flow according to the flow feature sorting sequences and then assigns offset for this flow. The resource availability derived from offset at the second step can not guide the flow sorting at the first step due to the serialization feature of the incremental scheduling scheme. As a result, it is impossible for flow sorting of traditional incremental scheduling to leverage both flow feature and resource availability, which decreases the load balance level. Many studies that adopt the serialized and inflexible incremental scheduling scheme neglect this mutual interaction. For example, CQF and DIP joint scheduling [13] is conducted with greedy algorithm, which sorts flows in size ascending order but neglects the guidance of offset to assist flow sorting, leading to low scheduling performance in heavy-traffic scenarios. ITP [14] proposes Tabu heuristic to find sub-optimal solutions but the initial solution is generated with greedy algorithm. As a result, it inevitably neglects the influence of offset on flow sorting and decreases the load balance level. The state-of-the-art FLJ-VB [12] utilizes position diversity to guide flow sorting but fails to utilize offset, which leads to marked scheduling performance degradation in resource-constrained scenarios. We fill this gap by utilizing the interaction between flow and offset. Specifically, we combine flow sorting and offset search together to form a (*flow, offset*) selecting step. The former provides information on flow features, i.e., the load that the flow will add to the network. The latter helps determine the resource availability, i.e., the load that the network can accommodate in the target flow path. These two factors jointly decide the impact of each-flow scheduling on the network load balance, which can guide the scheduling to minimize the impact on the network load balance in each step and reach a high load balance level. The comparison of previous incremental scheduling and our proposed approach is shown in Fig. 1.

To effectively determine the scheduling priority, i.e., select the most suitable (*flow, offset*) combination in each scheduling step, FLJ-VB [12] utilizes position diversity depending on flow period and path but it neglects the flow size and resource availability, leading to marked scheduling performance degradation in resource-constrained scenarios. Greedy algorithm [13] exploits flow size to determine schedule priority, which fails to consider resource availability and leads to unbalanced resource distribution. To tackle this problem, we design a uniform metric $Mapping\ Score$. It comprehensively exploits the information of both flows and global resources, contributing to high load balance.

In this paper, we improve the classical incremental scheduling scheme and propose an efficient approach to schedule time-sensitive flows in CQF-based TSN. The proposed approach combines flow sorting and offset search together to select flow and its offset combination (i.e., (*flow, offset*)) simultaneously in each step of scheduling. To determine the scheduling priority, we design a unified metric $Mapping\ Score$ to quantify the influence of different (*flow, offset*) combinations on the global resource (i.e., the schedulability of each combination). The $Mapping\ Score$ not only takes the flow sizes into account but also considers the availability of resources on the flow path. Following that, we design a heuristic scheduling algorithm ensuring each selected combination makes the least influence on the global resource load balance with $Mapping\ Score$. We compare the proposed algorithm with other prevalent algorithms (i.e., Naive, Naive Greedy, Tabu, and FLJ) under various settings. The experimental results show that the scheduling success rate with the proposed algorithm is, on average $31.69\%$ higher than with the Naive algorithm, and $4.57\%$ higher than with the most state-of-the-art FLJ. Meanwhile, it saves more than half the time of the heuristic computation-intensive Tabu algorithm. The contributions of this paper can be summarized as follows:

- We observe that flow sorting and offset search steps are mutually interdependent in the inflexible incremental scheduling. We design a novel approach by well utilizing this mutual interaction and coupling these two steps to form a (*flow, offset*) selecting step, which improves scheduling performance.
- A unified metric $Mapping\ Score$ is introduced to quantify the schedulability of each (*flow, offset*) combination and thus determine the scheduling priority. The $Mapping\ Score$ can assist the scheduling process to keep load balance in each step and improve the global load balance level.
- A $\underline{M}apping\ \underline{S}core$-based Scheduling (MSS) algorithm is designed as a general solver. MSS algorithm can generate potentially better (*flow, offset*) sequences and guarantee the load balance of the global resource. Meanwhile, it achieves a better trade-off between computation time and scheduling performance.

The rest of this paper is organized as follows. Section II presents the related work. Section III introduces the system models of switches, network topology, and data flows in CQF-based TSN. Section IV states the target optimization problem. Section V introduces $Mapping\ Score$ and its utility mechanism. Based on it, the designed algorithm is proposed. Section VI shows the experiment setup and evaluation results, and Section VII concludes this paper.

## II. Related Work

In past years, various scheduling algorithms under different scheduling models of TSN have been proposed. Craciunas *et al.* offer a comprehensible description of the IEEE 802.1 Qbv standard and a detailed investigation of the issues related to flow and frame isolation [15]. The authors propose a Satisfiability Modulo Theories (SMT) model over linear integer arithmetic that finds an exact transmission offset for each frame to minimize the number of used queues. They further find that the GCL without length limitation is not feasible for real-world devices. Hence, they propose a set of constraints for window-based scheduling of GCLs and

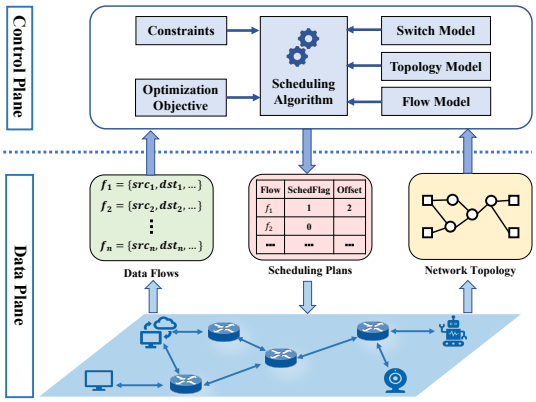



Fig. 2: Scheduling architecture in CQF. It is composed of a control plane and a data plane.

mapping the frames to the windows [16]. This model requires non-linear arithmetic due to the multiplication of variables. Experimental results of this model are given in [17], showing the trade-off between the computation time and the maximal number of windows per queue. Oliver *et al.* [18] take the formalization of constraints from [16] and turn it into an SMT model using the theory of arrays as the background theory, with jitter minimization objective. The Z3 solver is used for experimental evaluation and comparison between this window-based synthesis algorithm and the frame-based synthesis algorithm from [15]. Integer linear programming (ILP) is also used to solve scheduling tasks [19], [20]. Previous study [20] addresses a special case called no-wait scheduling, where the frames are not allowed to wait at switches in the queues. The proposed heuristic based on Tabu search was already used for no-wait job shop scheduling problems [21] and aimed to leave as much contiguous space as possible for the best-effort traffic.

However, the execution time of the solver-based scheduling significantly increases when the network scale enlarges. In CQF scheduling, the idea of incremental scheduling is adopted for better time efficiency in both research domains, i.e., offline scheduling and online scheduling. Offline scheduling solves static scheduling problems and has been widely applied in industrial systems [22]. Injection Time Planning (ITP) [14] is the first to solve traffic scheduling problems in CQF-based TSN. The authors propose the ITP mechanism to optimize the network throughput of time-sensitive flows. Tabu heuristic algorithm with domain-specific optimizing strategies is designed to schedule time-sensitive flows. In large-scale deterministic networks, CQF and DIP (Deterministic IP) are converged to schedule time-sensitive flows. The joint scheduling is formulated as ILP and is conducted in a greedy algorithm [13]. FLJ-VB [12], which is based on divisibility theory, is built to improve the schedule performance and reduce runtime. For online scheduling in CQF, FITS [23] is proposed to solve the online flow scheduling problem by incrementally generating traffic schedules for new time-sensitive flows. The above algorithms all adopt incremental scheduling. However, the classical incremental scheduling process can be further improved to increase the scheduling success rate and load balance level by utilizing the mutual interaction between flow sorting and offset search. In this paper, we mainly solve the offline scheduling problem and combine flow sorting and offset search together to form a (*flow, offset*) selecting step. To efficiently determine the scheduling priority, a unified metric $Mapping\ Score$ is introduced to quantitatively describe the impact of (*flow, offset*) on the network resource. To the best of our knowledge, we are the first to improve the incremental scheduling process.

## III. System Model

In this section, we build a global view of the scheduling architecture in TSN and elaborate core models in the architecture (i.e., switch, topology and data flow models) respectively.

### A. Scheduling Architecture

As Fig. 2 shows, TSN is composed of data plane and control plane [24]. The data plane consists of physical network elements, i.e., TSN switches and hosts. It offers real-time information on data flows and network topology of the industrial field to the control plane. The control plane stores global information [25] and plans the network traffic scheduling according to the information provided by the data plane. The control plane is composed of 3 parts.

- Models: Topology and flow models provide mathematical descriptions and abstraction for network topology and data flows. The switch model specifies the forwarding mode of data frames in CQF. The generated scheduling plans must follow the principles in the switch model. The detailed descriptions of these models are in the following part of Section III.
- Scheduling Algorithm: It generates scheduling plans which are issued to the hosts and switches in the data plane. Section V expands the algorithm in detail.
- Constraints and Optimization Objective: Constraints restrict the transmission of data flows in the network. The optimization objective is the goal of the scheduling algorithm. The scheduling algorithm must satisfy the constraints and achieve the optimization objective. They are elaborated in Section IV.

### B. Switch Model Supporting CQF

The frame forwarding scheme of CQF is presented in Fig. 3. Time is divided into time slots of equal length. The frame delivery follows two rules:

- If a switch received a frame at a time slot, the frame should be forwarded at the next time slot to the next hop.
- The frame delivery must be done in a time slot between two neighboring switches.

In this way, the data transmission in the TSN can realize bounded delay and jitter. Generally, the maximum and minimum end-to-end delay of a frame can be expressed by:

$$maxDelay = (hop_{num} + 1)T_{slot}, \tag{1}$$

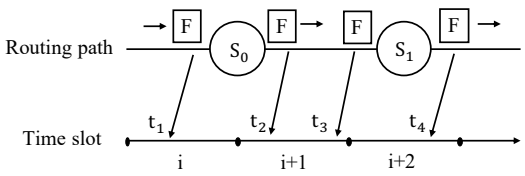



Fig. 3: Frame transmission in CQF. Time is divided into slots with equal duration and the frames received in slot $i$ are sent out in slot $i+1$.

$$minDelay = (hop_{num} - 1)T_{slot}, \tag{2}$$

where $hop_{num}$ is the number of switches on the routing path of the frame, and $T_{slot}$ is the length of time slot.

The switch model supporting CQF is depicted in Fig. 4. All the elements are synchronized in time. There are two queues responsible for the receiving and forwarding of time-sensitive flows for each port. At one time slot, one of the queues is in receiving mode while the other is in forwarding mode. At the next time slot, the modes of the two queues are reversed. The modes of both queues are controlled by $R_x$ GCL and $T_x$ GCL. As Fig. 4 shows, the contents of the GCLs are fixed and are 0 and 1 flip over time to avoid extra computation and configuration overhead.

### C. Network Topology

The network topology in the data plane can be abstracted as a graph $G = \{V, E\}$, which means the graph consists of vertices and directed edges. The directed edges $E$ represent transmission links in the network, and the vertices $V$ are composed of hosts and switches, which can be denoted as $V = \{H, S\}$. The switch's each port hosts 8 queues, 2 of which are used to buffer time-sensitive flows.

The host is the source and destination of flows. It generates data flows and determines when to send it out. Each host configures the starting time slot of each flow. When the flow can not be scheduled due to congestion and resource overflow, the host will not send out the frames belonging to the flow.

### D. Data Flow

In TSN, the minimum sending interval between two frames belonging to the same flow is the period of the time-sensitive flow. The amount of data sent in each period is fixed. The flow sends one frame in each cycle and generates data at time 0. (A flow with several frames can be taken as multiple single-frame flows.) We use $F$ to denote a set of time-sensitive flows. In $F$, there are $n$ time-sensitive flows. For time-sensitive flow $f_i$, we can describe it using a 7-tuple consisting of data source, destination, period, the data size of the frame within each period, routing path, the maximum allowable end-to-end delay, and the offset of the flow from the source.

$$\begin{aligned} &\forall f_i \in F, i \in [0, n-1], \\ &f_i = \{src, dst, period, size, path, deadline, \textit{offset}\}. \end{aligned} \tag{3}$$

We use $f_{i,j}$ to represent the $(j+1)$th ($j$ starts from 0) frame in flow $f_i$. *Offset* is the time when source host sends out the first frame of the flow. The sending time of frame $f_{i,j}$ at the source node is $f_i.\textit{offset} + j * f_i.period$. In the problem formulation, *offset* is the decision variable and all the other attributes of $f_i$ is known as prior.

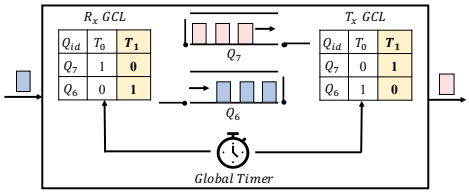



Fig. 4: Switch model in CQF. The current time slot is $T_1$.

## IV. Problem Statement

In this paper, we study the start time planning problem of time-sensitive flows. If the start time of flows is not well scheduled, they tend to converge in certain hotspots in the network, which causes queue overflow. In this section, we formulate the scheduling of time-sensitive flows into an optimization problem with four constraints and an optimization goal.

When allocating offset to time-sensitive flows, the transmission of these flows must comply with the constraints within a period containing all the frame occurrences. This period is the scheduling hyper period, which is the least common multiple ($LCM$) of all the periods of flow. We use $T_{sched}$ to denote the scheduling hyper period as in Eq. 4.

$$T_{sched} = \text{LCM}(F.periods) \tag{4}$$

We propose the constraints and optimization goal as follows.

### A. Offset Constraint

The offset constraint stipulates that the starting time of data flow should be within its period. If not, multiple frames of the same flow would accumulate in the host, which will consume the limited buffer space of the host. Moreover, the offset constraint helps reduce the searching complexity since the search space of offset is limited to a reasonable extent.

$$\begin{aligned} &\forall f_i \in F, i \in [0, n-1] \\ &0 \le f_i.\textit{offset} < \frac{f_i.period}{T_{slot}} \end{aligned} \tag{5}$$

where $T_{slot}$ is the length of the time slot. Since the granularity of offset is a time slot, flow period $f_i.period$ must be divided by the length of a time slot.

### B. Time Slot Constraint

The time slot constraint restricts the maximum and minimum length of a time slot. Since the unit of offset is the time slot in CQF, we must ensure the periods of all the flows can be divided by the length of the time slot. Consequently, the maximum length is the greatest common divisor (GCD) of the periods of all flows, as Eq. 6 shows.

$$Max(T_{slot}) = \text{GCD}(F.periods) \tag{6}$$

According to the CQF forwarding principle, a frame must be sent and received in the same time slot. Thus, the minimum

length of a time slot should have enough length for finishing frame forwarding. In the worst case, the queue is full of frames, and all the frames must arrive at the next hop within the same time slot at which they are sent out. The minimum length of a time slot is expressed by Eq. 7

$$Min(T_{slot}) = \frac{Queue_{length}}{BW} + hop_{delay} + \delta, \tag{7}$$

where $Queue_{length}$ is the maximum amount of bytes that each queue can hold. $BW$ is the link bandwidth, $hop_{delay}$ is the internal processing delay and propagation delay of a switch and $\delta$ is the clock synchronization precision. In practice, $Min(T_{slot})$ is much smaller than $Max(T_{slot})$. $T_{slot}$ is a property of the network in CQF-based TSN and requires to be determined as a constant in advance.

### C. Deadline Constraint

In the data flow model, each flow has a deadline. If the starting time of a flow is delayed, the flow cannot arrive at the destination on time. To avoid such a situation, the constraint restricts all flows arriving before the deadline.

$$\begin{aligned} &\forall f_i \in F, i \in [0, n-1] \\ &f_i.\textit{offset} + hop_{num}(f_i) \leq \frac{f_i.deadline}{T_{slot}} \end{aligned} \tag{8}$$

where $hop_{num}(f_i)$ is the number of switches in the path of flow $i$.

### D. Queue Resource Constraints

A resource block is the queue resource on one port of a switch at a time slot. We use $Q^{T(t)}_{S(j,k)}$ to denote the resource block in port $k$ of switch $j$ at the time slot $t$. Each resource block can be occupied by multiple flows. We denote the occupation state of a resource block by a flow as $O(f_i, Q^{T(t)}_{S(j,k)})$. If flow $f_i$ occupies the resource block $Q^{T(t)}_{S(j,k)}$, the value of $O(f_i, Q^{T(t)}_{S(j,k)})$ is 1, otherwise, it is 0. The condition for $O(f_i, Q^{T(t)}_{S(j,k)})$ to be 1 is shown in Eq. 9.

$$\begin{aligned} &\forall f_i \in F, i \in [0, n-1], j \in [0, m-1] \\ &\forall k \in [0, S_j.P_{num} - 1], \forall t \in [0, \frac{T_{sched}}{T_{slot}} - 1] \\ &O(f_i, Q^{T(t)}_{S(j,k)}) = 1 \\ &\textbf{s.t.} \\ &S(j,k) \in f_i.path, \alpha \in [0, \frac{T_{sched}}{f_i.period} - 1] \\ &t = (f_i.\textit{offset} + \frac{\alpha \times f_i.period}{T_{slot}} + hop(S_j, f_i)) \text{mod} (\frac{T_{sched}}{T_{slot}}) \end{aligned} \tag{9}$$

where $m$ is the number of switches in the network, and $S_j.P_{num}$ is the number of ports in the switch $j$. $S(j,k)$ is port $k$ of switch $j$. $hop(S_j, f_i)$ starts from 0 and indicates which hop switch $j$ is in the path of flow $i$. There may exist several frames of the same flow during $T_{sched}$. $\alpha$ indicates which period the frame belongs to. $t$ is the time slot that the flow occupies within $T_{sched}$.

TABLE I: Flow features.

| Flow | Size | Period (slot) | Routing path |
|---|---|---|---|
| $f_1$ | 25 | 2 | $port_A \longrightarrow port_B$ |
| $f_2$ | 26 | 4 | $port_A \longrightarrow port_C$ |
| $f_3$ | 27 | 3 | $port_A \longrightarrow port_D$ |

Based on the above denotation of resource block occupation by flows, the resource constraint is expressed in Eq. 10. $M(i)$ represents whether the flow $i$ could be mapped onto the network resource or not. $M(i)$ is 1 if the flow can be mapped successfully. Otherwise, it is 0. This constraint states that the sum of all the frames in the same resource block can not surpass the queue length.

$$\begin{aligned} &\forall f_i \in F, i \in [0, n-1], j \in [0, m-1] \\ &\forall k \in [0, S_j.P_{num} - 1], \forall t \in [0, \frac{T_{sched}}{T_{slot}} - 1] \\ &\sum_{i=0}^{n-1} M(i) \times O(f_i, Q^{T(t)}_{S(j,k)}) \times f_i.size \leq Queue_{length} \end{aligned} \tag{10}$$

In this paper, the optimization objective is maximizing the number of successfully mapped time-sensitive flows, which is formulated as Eq. 11. The decision variables are the *offset* of each flow in the network.

$$\textbf{Maximize} \sum_{i=0}^{n-1} M(i). \tag{11}$$

## V. Algorithm Design

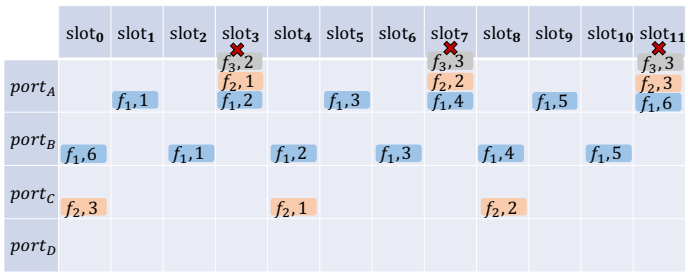
In order to schedule flows as many as possible and achieve load balance, we need to couple flow sorting and offset search together and jointly consider the flow features and resource availability to avoid flow congestion. In this section, we first introduce $Mapping\ Score$ to quantify the impact of each flow scheduling on the network resource and explain its necessity. Then we provide a utility analysis of coupling flow sorting and offset search with $Mapping\ Score$. Finally, the detailed scheduling algorithm based on $Mapping\ Score$ to solve the problem in Section IV is presented.

### A. Flow scheduling with $Mapping\ Score$

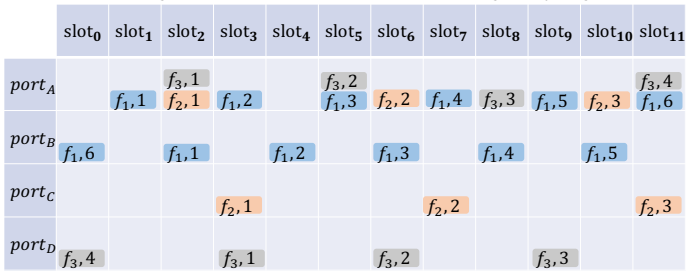
To demonstrate the necessity of jointly considering both flow features and resource availability, we take the scheduling of three flows as an example. Table I summarizes the features of the three flows. The queue length of switches is set to 60.

As Fig. 5a shows, the traditional greedy algorithm first sorts flows according to their frame size. Flows taking up fewer resources are mapped first. Thus, the flow sequence is ① $f_1$, ② $f_2$, ③ $f_3$. When conducting offset search, the algorithm maps the selected flow from the largest offset and reserve the relatively earlier time slot for the later-mapped flows.

When scheduling flows, we refer to the choice of a certain flow and its corresponding offset (i.e., the start time from its source) as a scheduling step. For example, assigning value 1 to the offset of flow $f_2$ can be depicted with $(f_2, 1)$. According




(a) Scheduling Gantt chart with the traditional greedy algorithm.




(b) Scheduling Gantt chart with $Mapping\ Score$.

Fig. 5: The number within each rectangle represents the sequence number of the frame, e.g., $f_1, 1$ means the 1st frame of flow $f_1$.

to the traditional incremental scheduling greedy algorithm, the scheduling steps are ① $(f_1, 1)$, ② $(f_2, 3)$. The scheduling of flow $f_3$ will cause resource overflow of $port_A$ at $slot_3$, $slot_7$ and $slot_{11}$. Consequently, this algorithm can only schedule two flows.

However, the scheduling performance will improve when we couple flow sorting and offset search together, and jointly consider the flow features and resource availability. We use $Mapping\ Score$ to quantify the impact of flow scheduling on the network resource. When scheduling flows, we tend to impact the network resource as little as possible by choosing the scheduling combination (*flow, offset*) whose $Mapping\ Score$ is the highest. We first elaborate on the definition of $Mapping\ Score$, then give the detailed scheduling process based on $Mapping\ Score$ in the example scenario.

As is mentioned in Section IV, the resource block in the $k$th port of $j$th switch at the $t$th slot is denoted as $Q^{T(t)}_{S(j,k)}$. If a flow occupies a resource block, the value $O(f_i, Q^{T(t)}_{S(j,k)})$ is 1, otherwise, it is 0. We define a flow set $F^{T(t)}_{S(j,k)} = \{f | O(f, Q^{T(t)}_{S(j,k)}) = 1\}$, meaning the flow set $F^{T(t)}_{S(j,k)}$ is composed of flows taking up the same resource block $Q^{T(t)}_{S(j,k)}$. The available resource of $Q^{T(t)}_{S(j,k)}$ is denoted as $Res^{T(t)}_{S(j,k)}$ in Eq. 12.

$$Res^{T(t)}_{S(j,k)} = Queue_{length} - \sum_{f \in F^{T(t)}_{S(j,k)}} f.size \tag{12}$$

For each combination of a flow and its possible offset ($f_i$, *offset*), there exists a $Mapping\ Score$, which can be computed with Eq. 13.

TABLE II: $Mapping\ Scores$ generated in the second scheduling step.

| Combination | $Mapping\ Score$ |
|---|---|
| $(f_2, 3)$ | $\frac{60-25}{26}$ |
| $(f_2, 2)$ | $\frac{60}{26}$ |
| $(f_2, 1)$ | $\frac{60-25}{26}$ |
| $(f_2, 0)$ | $\frac{60}{26}$ |
| $(f_3, 2)$ | $\frac{60-25}{27}$ |
| $(f_3, 1)$ | $\frac{60-25}{27}$ |
| $(f_3, 0)$ | $\frac{60-25}{27}$ |

$$Mapping\ Score = \frac{\text{Min } Res^{T(t)}_{S(j,k)} + f_i.size}{f_i.size}$$
$$s.t.$$
$$O(f_i, Q^{T(t)}_{S(j,k)}) = 1, \tag{13}$$

where Min $Res^{T(t)}_{S(j,k)} + f_i.size$ is the minimum available resource on the routing path of flow $f_i$ before $f_i$ is scheduled. It can only be worked out after the offset of $f_i$ is determined because we need offset to locate certain resource blocks occupied by $f_i$ according to Eq. 9. A large $Mappping\ Score$ means that the flow makes little impact on the network. Thus, we choose the combination with the highest $Mapping\ Score$ as the top-priority (*flow, offset*) to schedule.

Here we elaborate on the detailed scheduling process based on $Mapping\ Score$ in the above example. For the first step, we compute $Mapping\ Scores$ of all the possible scheduling combinations. Since the available resource of all the resource blocks is 60, a small flow size generates a high $Mapping\ Score$. So, we select flow $f_1$ to schedule. 0 and 1 are both possible values for the offset of $f_1$. We choose 1 as its offset to leave the earlier time slots for other flows. For the second step, the $Mapping\ Scores$ of all the possible combinations are listed in Table II. We choose combination $(f_2, 2)$ as the current scheduling step because its $Mapping\ Score$ is the largest. Likewise, we schedule the flow $f_3$ and set its offset as 2. As Fig. 5b shows, the method with $Mappping\ Score$ can successfully schedule all three flows. Under the guide of $Mapping\ Score$, the scheduling process of the example is ① $(f_1, 1)$, ② $(f_2, 2)$, ③ $(f_3, 2)$.

### B. Algorithm Design based on $Mapping\ Score$

The computation of mapping time-sensitive flows onto underlying queue resources is equivalent to the bin packing problem and is NP-hard [26]. The search space is $\prod_{i=0}^{n-1} \frac{f_i.period}{T_{slot}}$, where $n$ is the number of flows.

In order to reduce the complexity while improving scheduling performance, we propose a $Mapping\ Score$ based heuristic algorithm. The proposed algorithm is shown in Algorithm. 1. In each iteration, the flow selection and its offset are determined simultaneously. We denote this combination as (*flow, offset*). We choose the combination (*flow, offset*) whose

**Algorithm 1** $Mapping\ Score$ Based Scheduling (MSS) Algorithm

**Input:** Flow set $F$, Network topology $G$
**Output:** $F.offset$

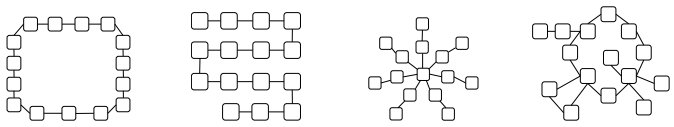
1: Initialize available queue resource $Q[port, t]$
2: Initialize mapping score table $Score[f, offset]$
3: **for** $i = 1 : F.num$ **do**
4:  **for** each $f_i \in F$ **do**
5:  **for** each $offset \in [0, f_i.period - 1]$ **do**
6:  $latency = count\_e2e\_latency(f_i, offset)$
7:  **if** $latency > f_i.deadline$ **then**
8:  $Score(f_i, offset) = -1$
9:  // Deadline constraint cannot be met
10:  **else if** $QueueOverflow(f_i, offset, Q) == true$ **then**
11:  $Score(f_i, offset) = -1$
12:  // Resource constraint can not be met
13:  **else**
14:  $Score = computeMappingScore(f_i, offset, Q)$
15:  **end if**
16:  **end for**
17:  **end for**
18:  $[f, offset] = findMaxScore(Score)$
19:  $f.offset = offset$
20:  $f.schedFlag = SUCCESS$
21:  $updateQueueResource(Q, f, offset)$
22:  $Score(f, :) = -1$
23:  Update score table $Score[f, offset]$
24: **end for**

$Mapping\ Score$ is the largest to be current scheduling step in each iteration.

In line 4-17, the $Mapping\ Score$ of each (*flow, offset*) combination is updated. The *Offset* $Constraint$ is satisfied in line 5, and the $Deadline\ Constraint$ is satisfied in line 6-7. Line 10 checks whether the $Queue\ Resource\ Constraint$ is met. Line 14 computes the $Mapping\ Score$ of each feasible combination (*flow, offset*). If the $Mapping\ Scores$ of combination (*flow, offset*) is -1, the value of *offset* in the combination is not feasible for the scheduling of flow $f$. After each $Mapping\ Score$ is updated, we choose the combination with the highest score as the current scheduling step, i.e., the flow $f$ is scheduled, and the starting time on its host is set as the value *offset*. Then the resource pool is updated in line 20 because of the flow scheduling. The $Score$ table is updated in line 23 because all the combinations of scheduled flow $f$ can be out of consideration in the next iteration. The algorithm ends until there is no available flow to schedule. This MSS heuristic algorithm generates near-optimal solutions by fully exploiting the knowledge of both flows and resource availability [14]. The time complexity of MSS algorithm is $O(n^2)$ [27].

## VI. Evaluation

To demonstrate the effectiveness of the proposed algorithm, we compare the proposed algorithm with four algorithms, i.e., Naive algorithm (NV) [14], Naive Greedy algorithm (NG) [13], Tabu algorithm (Tabu) [14] and FLJ-VB (FLJ) [12]. NV randomly chooses a flow from the flow set and schedules it as soon as it is generated from the source. NG algorithm first sorts flows by their frame sizes in ascending order. When offsetting the time slot at the source, the greedy strategy of decreasing offset is adopted. For Tabu parameters, the size of the tabu list is 50. We set the maximum iteration and the maximum repetition to 300 and 20, respectively. In each iteration, 10 candidate solutions are generated. For FLJ, the PDV-descending order is adopted.



(a) Ring (b) Linear (c) Tree (d) Mix

Fig. 6: Experiment topologies. The rectangles represent switches. The host number connecting to each switch ranges from 1 to 3 randomly.

We conduct experiments on a server with an Intel Xeon Silver 4210R Processor (10 cores, 2.40 GHz with 13.75 MB cache) and 252 GB RAM.

### A. Experiment Setup

*1) Network Setting:* We conduct experiments on three topologies: ring, linear, and tree, which are typical topologies in industry. To evaluate the scalability of the proposed approach, we use two network scales, denoted by $S_1$ and $S_2$, which consist of 7 and 15 switches, respectively, according to typical industrial scenarios [28]. Especially, we use a complex network topology for the $S_2$ network to evaluate the performance of the proposed algorithm. Fig. 6 presents the layouts of these topologies. The number of hosts connected to each switch is randomly selected from the set $\{1, 2, 3\}$. We set the link bandwidth to 1000 Mb/s and time slot length to 125 µs.

*2) Flow Features:* To evaluate the robustness of the proposed algorithm, we use three sets of periods:[2, 7], [2, 9] and [2, 11] time slots. (i.e., The flow period is randomly generated from the corresponding set.) We evaluate the proposed algorithm with different flow numbers, i.e., 200 and 500. For different flow numbers, we set different queue lengths for them. When the flow number is 200, the queue length is set to 5 KB. When the flow number is 500, the queue length is set to 8 KB. The queue length selection is based on two reasons. First, Restricted queue length will decrease the network capacity and the scheduling performance, so the queue length can not be too short. Second, the on-chip memory in most TSN switches is limited, so the queue length can not be too long for cost savings [14]. The frame size is randomly chosen from 64 to 1500 bytes. The deadlines of all the flows are milliseconds in general. They are generated from 2 to 5 milliseconds randomly. For each parameter setting, we run the experiment 50 times.

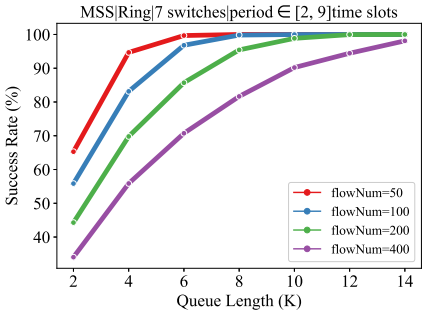



Fig. 7: Success rate under different queue lengths. Different lines represent different flow numbers.

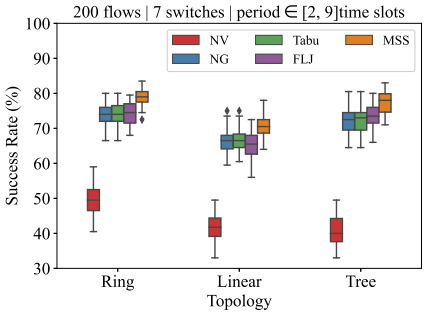



Fig. 8: Scheduling success rate in light traffic.

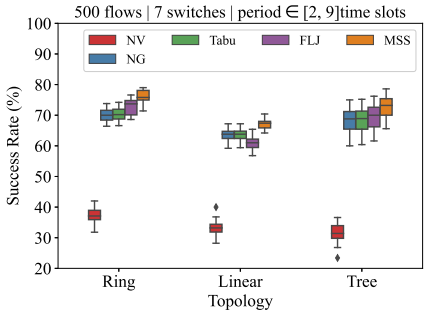



Fig. 9: Scheduling success rate in heavy traffic.

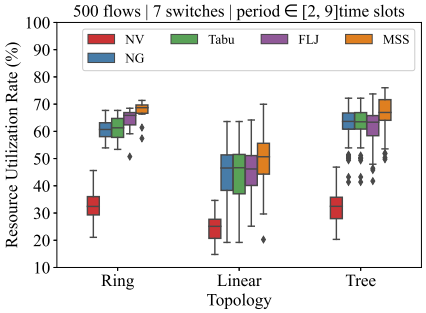



Fig. 10: Resource utilization. It is the proportion of the allocated queues over the total queues of all ports.

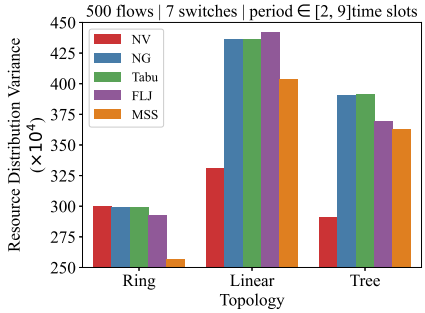



Fig. 11: Resource distribution variance. Load balance level is measured by the resource distribution variation.

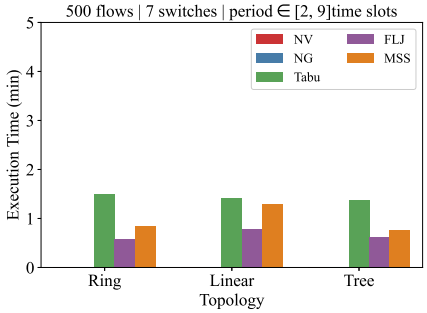



Fig. 12: Execution time. The execution time of NV and NG are below 3s.

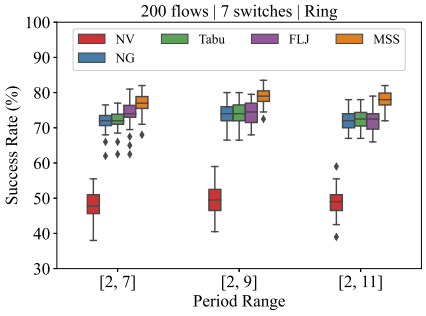



Fig. 13: Scheduling success rate with different flow period ranges.

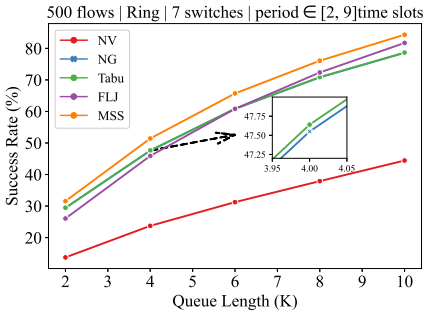



Fig. 14: Success rate of different algorithms under different queue lengths. NG and Tabu almost coincide into the same line due to similar performance.

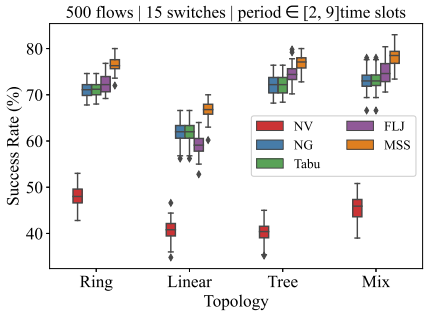



Fig. 15: Scheduling success rate in complex network scenarios with a large flow number and more switches.

### *B. Experimental Results*

*1) Impact of Flow Number and Queue Length:* The scheduling success rate with different queue lengths and flow numbers is shown in Fig. 7. We can see that, when the flow number in the network is fixed, the success rate grows as the queue resource increases. When the queue length is fixed, the scheduling success rate decreases as the flow number increases since a large number of flows increases the probability of congestion and conflict. In practice, field operators of the factory should select switches with suitable queue lengths according to the actual flow number in the network.

*2) Performance of Algorithms under Different Topologies:* To qualitatively evaluate the performance of the proposed algorithm, we use four metrics, i.e., scheduling success rate, resource utilization, resource distribution variance, and execution time. We compare the proposed algorithm with four algorithms (i.e., NV, NG, Tabu, and FLJ).

The scheduling success rate is the proportion of the scheduled flow number to the total flow number in the network. A high scheduling success rate indicates that most flows are scheduled successfully. Results in Fig. 8 present the scheduling success rate of five scheduling algorithms under different network topologies when the flow number is 200. In this light-traffic scenario, the success rate of MSS is $31.69\%$ higher than that of NV on average. MSS outperforms the FLJ algorithm by $4.57\%$ averagely, which is state-of-the-art.

Likewise, the scheduling rates of five algorithms when the flow number is 500 are displayed in Fig. 9. In this heavy-traffic scenario, the success rate of MSS is $38.10\%$ higher than that of NV on average. MSS outperforms the state-of-the-art FLJ algorithm by $4.22\%$ averagely. The MSS improves the scheduling success rate because it couples flow sorting and offset search together under the guidance of $Mapping\ Score$. Meanwhile, the MSS is robust under both light-traffic and heavy-traffic scenarios.

Resource utilization is the proportion of the allocated queues over the total queues. A high resource utilization means the devices are fully utilized. As Fig. 10 shows, compared with the other four algorithms, the proposed algorithm MSS improves the resource utilization by $30.97\%$, $5.11\%$, $4.88\%$ and $4.10\%$, respectively. This is because MSS takes resource availability into consideration compared with NV, NG, and Tabu, and MSS considers the flow size when sorting flows compared with FLJ. Thus MSS avoids overusing resource blocks and outperforms other algorithms in resource utilization.

The resource distribution variance is shown in Fig. 11. Load balance level is measured by the resource distribution variance over all the queues. A low resource distribution variance value means a high load balance level, which can make sure the hardware resource is fairly used. The average variance of FLJ is $7.95\%$ higher than that of MSS. This is because MSS jointly considers resource availability and flow features with $Mapping\ Score$ while other approaches either neglect the resource distribution or the resource consumption of flow itself when scheduling flows.

The execution time is presented in Fig. 12. Tabu, FLJ, and MSS take much longer time than NV and NG because of more complex searching space. Tabu algorithm consumes $2.2\times$ more time than MSS. The execution time of MSS is slightly more than that of FLJ due to the computation and sorting of $Mapping\ Score$. On average, MSS costs $18.68$s more than FLJ, which is affordable in real-time industrial systems.

To conclude, MSS can improve the scheduling performance while significantly reducing the execution time. Contributing to $Mapping\ Score$, MSS can make flows take up available resource blocks instead of aggregating to one hot spot.

*3) Impact of Different Flow Periods:* In this experiment, we compare the scheduling success rate with different flow period ranges, as shown in Fig. 13. When the flow period is fixed at [2, 11], the scheduling success rate of MSS is $5.88\%$ higher than FLJ. The improvement is $4.77\%$ and $2.70\%$ When the flow period range is [2, 9] and [2, 7], respectively. The period range increment leads to the explosive growth of the hyper period, thus the search space is enlarged. The performance promotion demonstrates the effectiveness of MSS in the case of large search space.

*4) Impact of Different Queue Length:* Fig. 14 shows the scheduling success rate of different algorithms under different queue lengths. Different queue lengths of switches indicate different adequacy of network resources. Fig. 14 depicts the scheduling capabilities of these algorithms under different network resource adequacy. We can see that, when the queue length is less than 6K, Tabu and NG outperform FLJ, i.e., Tabu and NG have better scheduling capabilities in resource-constrained scenarios while FLJ performs well in resource-sufficient scenarios. MSS remains the best in both scenarios. This is because the sorting policy of FLJ fails to consider the flow size. Thus it is possible that the flows scheduled first may have a large size and almost fill the resource block when queue length is insufficient, resulting in the scheduling failure of other flows. In resource-sufficient scenarios, the defect of flow sorting in FLJ can be negligible, however, whether the offset select considers the resource availability matters. NG and Tabu fail to consider the most restricted resource block in the flow path and miss the optimal offset, thus occupying the resource block which is almost filled and hindering other flows from being scheduled. MSS jointly considers both factors so its scheduling capability remains the best in both scenarios.

*5) Scalability:* To evaluate the robustness and scalability of MSS, we test the performance of five algorithms under complex scenarios as Fig. 15 shows. In the complex scenarios with more switches and more flows, MSS outperforms FLJ by $4.38\%$ and it ourperforms Tabu by $5.01\%$ on average. Especially, the performance improvement of MSS versus FLJ is $7.62\%$ in the linear topology. Consequently, the MSS algorithm can be well extended in complex scenarios.

## VII. Conclusion

In this paper, we utilize the mutual interaction of flow and offset to design a novel scheduling approach, which couples flow sorting and offset search together to form a (*flow, offset*) combination selecting step. Moreover, we introduce $Mapping\ Score$ to quantitatively describe the impact of the combination on the network resource and propose a $Mapping\ Score$ based scheduling (MSS) algorithm to schedule time-sensitive flows. The proposed MSS algorithm jointly considers the flow features and the availability of resources on the flow path. MSS generates (*flow, offset*) sequences based on $Mapping\ Score$ to achieve better load balance. The experimental results show that the scheduling success rate with the MSS algorithm is on average $31.69\%$ higher than with the Naive algorithm and $4.57\%$ higher than the state-of-the-art FLJ. Especially, it outperforms the state-of-art by 7.62% in large-scale linear topologies. Meanwhile, it saves more than half the time of the compute-intensive Tabu heuristic algorithm. The MSS algorithm achieves great time efficiency and scalability for complex network scenarios.

## References


[1] S. A. Bhat, N.-F. Huang, I. B. Sofi, and M. Sultan, "Agriculture-food supply chain management based on blockchain and iot: A narrative on enterprise blockchain interoperability," *Agriculture*, vol. 12, no. 1, 2022. [Online]. Available: https://www.mdpi.com/2077-0472/12/1/40

[2] A. W. Colombo, S. Karnouskos, X. Yu, O. Kaynak, R. C. Luo, Y. Shi, P. Leitao, L. Ribeiro, and J. Haase, "A 70-year industrial electronics society evolution through industrial revolutions: The rise and flourishing of information and communication technologies," *IEEE Industrial Electronics Magazine*, vol. 15, no. 1, pp. 115–126, 2021.

[3] M. Ahmad, N. Javaid, I. A. Niaz, A. Almogren, and A. Radwan, "A cost-effective optimization for scheduling of household appliances and energy resources," *IEEE Access*, vol. 9, pp. 160 145–160 162, 2021.

[4] H. R. Chi, M. d. F. Domingues, K. I. Kostromitin, A. Almogren, and A. Radwan, "Spatiotemporal d2d small cell allocation and on-demand deployment for microgrids," *IEEE Access*, vol. 9, pp. 116 830–116 844, 2021.

[5] B. Fong, A. C. M. Fong, and K.-F. Tsang, "Capacity and link budget management for low-altitude telemedicine drone network design and implementation," *IEEE Communications Standards Magazine*, vol. 5, no. 4, pp. 74–78, 2021.

[6] N. G. Nayak, F. Dürr, and K. Rothermel, "Incremental flow scheduling and routing in time-sensitive software-defined networks," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 5, pp. 2066–2075, 2017.

[7] J. L. Messenger, "Time-sensitive networking: An introduction," *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 29–33, 2018.

[8] M. Vlk, Z. Hanzálek, K. Brejchová, S. Tang, S. Bhattacharjee, and S. Fu, "Enhancing schedulability and throughput of time-triggered traffic in ieee 802.1qbv time-sensitive networks," *IEEE Transactions on Communications*, vol. 68, no. 11, pp. 7023–7038, 2020.

[9] Z. Feng, M. Cai, and Q. Deng, "An efficient pro-active fault-tolerance scheduling of ieee 802.1qbv time-sensitive network," *IEEE Internet of Things Journal*, pp. 1–1, 2021.

[10] *IEEE 802.1Qbv Standard*, Std. [Online]. Available: https://www.ieee802.org/1/pages/802.1bv.html

[11] *IEEE 802.1Qch Standard*, Std. [Online]. Available: https://1.ieee802.org/tsn/802-1qch/

[12] Y. Zhang, Q. Xu, L. Xu, C. Chen, and X. Guan, "Efficient flow scheduling for industrial time-sensitive networking: A divisibility theory based method," *IEEE Transactions on Industrial Informatics*, pp. 1–1, 2022.

[13] 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)*, 2021, pp. 1–6.

[14] 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*, 2020, pp. 616–625.

[15] 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.

[16] S. S. Craciunas, R. S. Oliver, and W. Steiner, "Formal scheduling constraints for time-sensitive networks," *arXiv preprint arXiv:1712.02246*, 2017.

[17] W. Steiner, S. S. Craciunas, and R. S. Oliver, "Traffic planning for time-sensitive communication," *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 42–47, 2018.

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

[19] 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, 2019.

[20] 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.

[21] R. Macchiaroli, S. Mole, and S. Riemma, "Modelling and optimization of industrial manufacturing processes subject to no-wait constraints," *International Journal of Production Research*, vol. 37, no. 11, pp. 2585–2607, 1999.

[22] L. L. Bello and W. Steiner, "A perspective on ieee time-sensitive networking for industrial communication and automation systems," *Proceedings of the IEEE*, vol. 107, no. 6, pp. 1094–1120, 2019.

[23] W. Quan, J. Yan, X. Jiang, and Z. Sun, "On-line traffic scheduling optimization in ieee 802.1qch based time-sensitive networks," in *2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, 2020, pp. 369–376.

[24] N. G. Nayak, F. Dürr, and K. Rothermel, "Time-sensitive software-defined network (tssdn) for real-time applications," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, ser. RTNS'16. New York, NY, USA: Association for Computing Machinery, 2016, p. 193–202. [Online]. Available: https://doi.org/10.1145/2997465.2997487

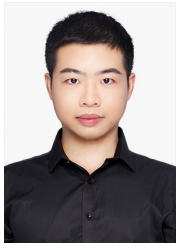
[25] Z. Li, H. Wan, Z. Pang, Q. Chen, Y. Deng, X. Zhao, Y. Gao, X. Song, and M. Gu, "An enhanced reconfiguration for deterministic transmission in time-triggered networks," *IEEE/ACM Transactions on Networking*, vol. 27, no. 3, pp. 1124–1137, 2019.

[26] E. Falkenauer and A. Delchambre, "A genetic algorithm for bin packing and line balancing," in *Proceedings 1992 IEEE International Conference on Robotics and Automation*, 1992, pp. 1186–1192 vol.2.

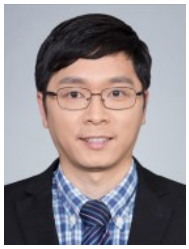
[27] M. L. Raagaard and P. Pop, "Optimization algorithms for the scheduling of ieee 802.1 time-sensitive networking (tsn)," 2018.

[28] S. S. Craciunas and R. S. Oliver, "Combined task- and network-level scheduling for distributed time-triggered systems," *Real-Time Syst.*, vol. 52, no. 2, p. 161–200, mar 2016. [Online]. Available: https://doi.org/10.1007/s11241-015-9244-x


**Miao Guo** (Student Member, IEEE) received the B.Eng. degree from Zhejiang University, Hangzhou, China, in 2020. She is currently working toward the Ph.D. degree with the College of Control Science and Engineering, Zhejiang University, Hangzhou, China. Her research interests include industrial IoT, deterministic network and Time-sensitive Networking.



**Chaojie Gu** (Member, IEEE) received the B.Eng. degree from Harbin Institute of Technology, Weihai, China, in 2016, and the Ph.D. degree in computer science and engineering from Nanyang Technological University, Singapore, in 2020. He was a Research Fellow with Singtel Cognitive and Artificial Intelligence Lab for Enterprise (SCALE) in 2021. He is an Assistant Professor with the College of Control Science and Engineering, Zhejiang University, Hangzhou, China. His research interests include IoT, industrial IoT, edge computing, and low power wide area network.



**Shibo He** (Senior Member, IEEE) received the Ph.D. degree in control science and engineering from Zhejiang University, Hangzhou, China, in 2012. He is currently a Professor with Zhejiang University. He was an Associate Research Scientist from March 2014 to May 2014, and a Postdoctoral Scholar from May 2012 to February 2014, with Arizona State University, Tempe, AZ, USA. From November 2010 to November 2011, he was a Visiting Scholar with the University of Waterloo, Waterloo, ON, Canada. His research interests include Internet of Things, crowdsensing, big data analysis, etc.

**Zhiguo Shi** (Senior Member, IEEE) received the B.S. and Ph.D. degrees in electronic engineering from Zhejiang University, Hangzhou, China, in 2001 and 2006, respectively. Since 2006, he has been a Faculty Member with the Department of Information and Electronic Engineering, Zhejiang University, where he is currently a Full Professor. From 2011 to 2013, he visited the Broadband Communications Research Group, University of Waterloo, Waterloo, ON, Canada. His current research interests include signal and data processing.

**Jiming Chen** (Fellow, IEEE) received the B.Sc. and Ph.D. degrees in control science and engineering from Zhejiang University, Hangzhou, China, in 2000 and 2005, respectively. He is currently a Professor with the College of Control Science and Engineering, and the Deputy Director of the State Key Laboratory of Industrial Control Technology, Zhejiang University. His research interests include the Internet of Things, sensor networks, networked control, and control system security.