Towards Distributed Flow Scheduling in IEEE 802.1Qbv Time-Sensitive Networks

MIAO GUO, Zhejiang University, China
SHIBO HE, Zhejiang University, China
CHAOJIE GU, Zhejiang University, China
XIUZHEN GUO, Zhejiang University, China
JIMING CHEN, Zhejiang University, China
TAO GAO, Huawei Technologies Co., Ltd., China
TONGTONG WANG, Huawei Technologies Co., Ltd., China

Flow scheduling plays a pivotal role in enabling Time-Sensitive Networking (TSN) applications. Current flow scheduling mainly adopts a centralized scheme, posing challenges in adapting to dynamic network conditions and scaling up for larger networks. To address these challenges, we first thoroughly analyze the flow scheduling problem and find the inherent locality nature of time scheduling tasks. Leveraging this insight, we introduce the first distributed framework for IEEE 802.1Qbv TSN flow scheduling. In this framework, we further propose a multi-agent flow scheduling method by designing Deep Reinforcement Learning (DRL)-based route and time agents for route and time planning tasks. The time agents are deployed on field devices to schedule flows in a distributed way. Evaluations in dynamic scenarios validate the effectiveness and scalability of our proposed method. It enhances the scheduling success rate by 20.31% compared to state-of-the-art methods and achieves substantial cost savings, reducing transmission costs by 410× in large-scale networks. Additionally, we validate our approach on edge devices and a TSN testbed, highlighting its lightweight nature and ease of deployment.

CCS Concepts: • **Networks** → **Cyber-physical networks**; *In-network processing*; **Packet scheduling**.

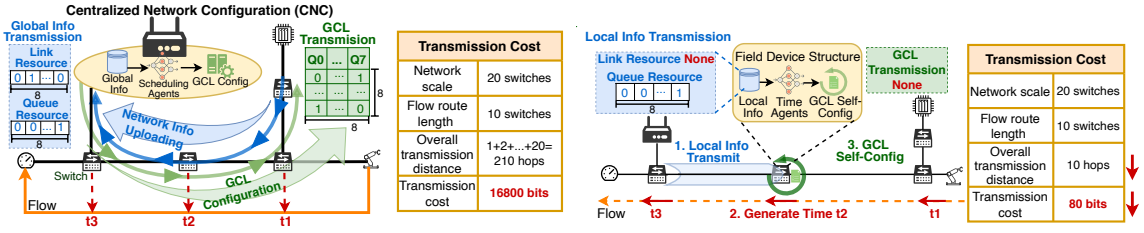
Additional Key Words and Phrases: Time-Sensitive Networking, distributed scheduling, deep reinforcement learning

1 INTRODUCTION

Time-Sensitive Networking (TSN) has arisen as a promising networking technology in response to the advancement of the Industrial Internet of Things (IIoT) and cyber-physical systems [38]. Evolving from traditional Ethernet, TSN is proposed in IEEE 802.1 Standard [23] to offer deterministic, low-latency, and low-jitter transmission [21] for time-critical applications [3, 9]. To achieve real-time and deterministic communications, TSN specifies the gate mechanism Time-Aware Shaper (TAS) in IEEE 802.1Qbv [14] for flow scheduling, which controls the route and time schedules of flows using the Gate Control List (GCL) in each switch.

The GCL is computed using various network factors, including network topology, flow model, and scheduling constraints specified by the TSN standard. Since the GCL is loaded into the switch before operation, even minor changes in the network status trigger a recomputation of the GCL to align with the updated network conditions. Thus, various flow scheduling algorithms have been proposed to efficiently compute GCLs in IEEE 802.1Qbv time-sensitive networks. Solver-based methods [6, 16, 25] model flow transmission rules as a set of constraints and feed them into Satisfiability

Authors' addresses: Miao Guo, Zhejiang University, Yuquan Campus, 38 Zheda Road, Hangzhou, China, gm_oct@zju.edu.cn; Shibo He, Zhejiang University, Yuquan Campus, 38 Zheda Road, Hangzhou, China, s18he@zju.edu.cn; Chaojie Gu, Zhejiang University, Yuquan Campus, 38 Zheda Road, Hangzhou, China, gucj@zju.edu.cn; Xiuzhen Guo, Zhejiang University, Yuquan Campus, 38 Zheda Road, Hangzhou, China, guoxz@zju.edu.cn; Jiming Chen, Zhejiang University, Yuquan Campus, 38 Zheda Road, Hangzhou, China, cjm@zju.edu.cn; Tao Gao, Huawei Technologies Co., Ltd., 156 Beiqing Road, Beijing, China, tao.g@huawei.com; Tongtong Wang, Huawei Technologies Co., Ltd., 156 Beiqing Road, Beijing, China, tongtong.wang@huawei.com.



(a) **Centralized**: CNC centrally gathers global network information (8×210 bit-link and 8×210 bit-queue resource information) and configures the GCLs ($8 \times 8 \times 210$ bits) of all the field devices.

(b) **Distributed**: Each field device on the flow route gathers local information (i.e., 8-bit queue resource information) from its adjacent device to calculate and configure its own GCL.

Fig. 1. Centralized v.s. Distributed DRL scheduling schemes for a 20-switch linear TSN network. Assuming the length of the resource information and GCL is 8, the distributed scheme significantly reduced the volume of exchanging information by 210×.

Modulo Theories (SMT) or Integer Linear Programming (ILP) solvers to search for the optimal solution. However, they usually take a long time to search for solutions due to computational complexity. Heuristic methods [35, 41] reduce the search time by narrowing down the search space. However, the designing processes are handcrafted and highly dependent on domain knowledge and professional expertise [7, 11]. Recently, Deep Reinforcement Learning (DRL) has emerged as a promising approach for flow scheduling due to its self-exploiting capability without any laborious efforts. For instance, DRLS [42] calculates the flow routes by designing a DRL-based scheduler. Similarly, TTDeep [18] proposes a DRL-based framework to jointly determine the route and time plans of flows.

Fig. 1a illustrates the typical scheduling process of the existing approaches, which adopt a centralized scheme. The Centralized Network Configuration (CNC) collects global network information to calculate the GCL, and then configure the GCLs on all filed devices. However, such a scheme has two critical scalability issues regarding: *i) Network condition:* Current approaches share a common prerequisite that CNC has comprehensive knowledge of the network resources and flows. As a result, in situations where network conditions are dynamic, i.e., emergency flow arrival and network resources change due to manufacturing requirements, existing approaches need to stop the network transmission and perform the above scheduling process again [29]; *ii) Network size:* Existing methodologies need to gather information from the whole network for computation and disseminate GCLs to all field devices. The communication overhead and configuration complexity are exacerbated by the expansion of the network size.

To address the scalability issue, our paper takes an important step in this domain — we introduce the first distributed framework for IEEE 802.1Qbv TSN flow scheduling. As shown in Fig. 1b, our framework empowers individual field devices to conduct GCL computation and configuration locally. This is achieved by utilizing local network information obtained from the device within one hop. In such a distributed framework, even when flow variations and network resources undergo frequent changes, each field device can swiftly update its GCL to align with the evolving network conditions. Consequently, this approach results in a significant reduction in both communication overhead and configuration complexity.

Our design is based on a key observation: **the time planning process of flow scheduling exhibits inherent locality**, challenging the common belief that TSN flow scheduling is a global coupling combinatorial optimization problem. Specifically, there is a mutual interdependence between time schedule and resource state. The offset time assigned to each link impacts the resource state, which subsequently influences the selection of the offset time to avoid finding resource conflicts. This interdependence involves two key resources: i) the link resource on the current

departure link, and ii) the queue resource on the next link along the route, as the flow enters the queue of the next link upon departing from the current link. Consequently, for the time planning process, the relevant resource information within a two-link range is sufficient, indicating that global knowledge is unnecessary.

Following this insight, our design further addresses three key challenges.

Searching Space Reduction: First, as flow scheduling requires both route and time solutions, jointly optimizing them in a single step leads to vast search space and slow convergence. We tackle this by splitting scheduling into sub-problems, i.e., route and time planning to reduce the search space of each problem. To solve them, we propose a multi-agent DRL scheduling method by designing both route agent and time agent for route and time planning tasks, respectively.

Effective Pattern Representation: Second, given the intricacies of flow scheduling, exploiting only link resources leads to low schedulability due to its insufficiency to depict network features and constraints. We overcome it by introducing queue resources to the system model and incorporating it into the DRL features, enhancing the exploration ability and schedulability of agents.

Efficient Training Strategy: Third, on account of the inherent locality of the time planning task, we offload it to field devices, enabling the time agent to explore and exploit the network locally. Unfortunately, the limited local knowledge and stringent scheduling constraints often result in model instability and poor generalization. From experiments, we observe a gradual convergence of model weights when training it in a complexity-increasing order of scheduling constraints. Thus, we propose a general model training framework by customizing training curriculums with different levels of scheduling constraint complexity, yielding model stability and desirable scheduling performance.

We implement our framework DiRTS (Distributed Reinforcement Learning for Flow Scheduling) and compare it with other state-of-the-art methods. The results from experiments in dynamic scenarios show that our scheduling success rate, on average, is 20.31% higher than DRLS method and DiRTS saves transmission costs by 410× in large-scale networks compared to the centralized scheme. The contributions of this paper can be summarized as follows:

- We introduce the first distributed reinforcement learning framework for IEEE 802.1Qbv TSN scheduling by offloading the time planning task to field devices, which guarantees scalability and low transmission cost.
- We propose a multi-agent DRL scheduling method by designing both route agent and time agent to jointly determine the route and time schedule of flows. We find the inherent locality property of time planning tasks and deploy time agents to field devices to perform scheduling distributedly.
- We develop an open-source dataset generation tool to ease the evaluation efforts on TSN scheduling research. It can generate flows and network topologies allowing user-defined features. We evaluate the performance of the proposed method with a network simulator, and validate its applicability on industry-level field devices and TSN testbed.

The rest of this paper is organized as follows. Section 2 reviews the related works. Section 3 discusses the background and system models of TSN, and elaborates on the scheduling problem and constraints. Section 4 presents the DiRTS framework and analysis. Section 5 and Section 6 present the design of the route agent and time agent, respectively. In Section 7, we provide details on the evaluation setup and results. We present the simulation results and the TSN testbed results in Section 8. Section 9 discusses related issues. Finally, Section 10 concludes the paper.

2 RELATED WORK

TSN Scheduling: TSN scheduling has been studied since the release of IEEE 802.1Qbv in 2015. The previous works can be divided into three categories: solver-based, heuristic-based, and DRL-based scheduling. For solver-based scheduling,

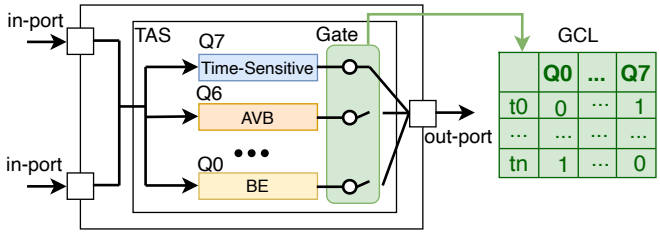


Fig. 2. Switch model of TAS. The gates of queues are controlled by the GCL.

Craciunas *et al.* [6] propose a SMT model over linear integer arithmetic based on their prior studies on Time-Triggered Ethernet (TTE) [5, 27, 28]. Thereafter, they formalize the scheduling problem as a set of constraints expressed with the first-order theory of arrays [25] and then solve it with SMT solvers. Another aspect of solver-based studies formalizes scheduling problems as ILP and adopts the ILP-based method [8, 26, 32, 40]. Solver-based methods generate the optimal schedule by exploring the whole search space, which has exponential time complexity. If they are utilized to perform scheduling in dynamic manufacturing scenarios that require frequent schedule computations, the time consumption will be significant, degrading production efficiency. For heuristic-based scheduling, Nayak *et al.* [7] propose the No-wait Packet Scheduling Problem (NW-PSP) to calculate TSN schedules, which is solved in Tabu search algorithm. Tabu heuristic is also applied in Injection Time Planning (ITP) [41] to solve the scheduling problem in CQF-based TSN. Besides the widely used Tabu heuristic, Pop *et al.* introduce a scheduling algorithm based on Greedy Randomized Adaptive Search Procedure (GRASP) [11, 31]. For DRL-based scheduling, the related works are as follows.

DRL-based Scheduling: Thanks to the great generalization of DRL, many prior works exploit it in network scheduling. Stampa *et al.* [34] propose an actor-critic policy gradient algorithm for routing optimization in SDN. Mao *et al.* [22] utilizes graph neural network (GNN) and policy network to schedule data processing jobs on the computing clusters. Chinchali *et al.* [4] present a reinforcement learning (RL) based scheduler to optimally schedule IoT traffic in mobile networks. Huang *et al.* [17] use DRL to solve TDMA route and link scheduling problems in wireless sensor networks. Sellami *et al.* [33] employ DRL to solve computing task scheduling problems in software-defined Fog-IoT networks. However, the network protocol and flow transmission pattern of the above studies are totally different from TSN, which can not be applied in TSN scheduling directly.

For DRL-based TSN scheduling, DRLS [42] and DeepScheduler [15] apply DRL to calculate the route of flows in TSN. However, the DRL is only used in route planning, leaving the time planning task solved heuristically. TTDeep [18] applies DRL to solve route and time planning problems in TSN. DeepCQF [2] utilizes DDQN to schedule flows in CQF-based TSN, which is inapplicable in TAS.

The DRL-based methods mentioned above employ a centralized approach, which struggles to adapt to dynamic network conditions and poses challenges for large-scale deployment. In contrast, our approach embraces a distributed scheme, ensuring scalability in scheduling.

3 PRELIMINARY

3.1 Switch Model

In TSN, flows are categorized into three classes: time-sensitive flows, Audio/Video Bridging (AVB) flows, and best-effort flows depending on their criticality and Quality-of-Service (QoS) requirements. Among them, time-sensitive flows have

the stringent determinism requirement and the highest priority. To guarantee their deterministic, low-latency, and low-jitter transmission, TSN specifies TAS for flow scheduling. As Fig. 2 shows, flows of different classes enter the corresponding queues, which enable/disable the flow transmission with the open/close state of the gates to control the departure time of time-sensitive flows from each switch. Since the gate state follows the predefined GCL, we calculate the GCLs, i.e., determine the transmission time of flows on each switch to schedule the deterministic time-sensitive flows. The scheduling of other classes of flows is of lower priority than time-sensitive flows and orthogonal to our work.

3.2 Network and Flow Model

The topology of a TSN network is modeled as a directed graph $G = \{V, E\}$. The set of vertices V represents all the devices in the network, including the set of all end systems (end devices) denoted as ES and the set of all switches denoted as SW. Hence $V = ES \cup SW$. End systems are responsible for sending and receiving flows while switches are responsible for forwarding flows. The set of edges E represents data links in the network. The network supports full-duplex transmission. Two vertices $v_a, v_b \in V$ determine two edges $[v_a, v_b] \in E$ and $[v_b, v_a] \in E$. The transmission rate of link $[v_a, v_b]$ is $[v_a, v_b].s$.

We use F to denote a set of time-sensitive flows. In TSN, the flow patterns are complicated but fixed and can be denoted by a 5-tuple consisting of source, destination, period, frame length, and maximum end-to-end delay:

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

For a flow starting from v_a and ending at v_b, its route Rt is an ordered sequence : $[[v_a, v_{a+1}], \ldots, [v_{b-1}, v_b]]$. Since the flow is periodic, we use $f_{i,m}^{[v_a,v_b]}$ to denote the transmission of the m-th frame of flow f_i on link $[v_a, v_b]$. Frame $f_{i,m}^{[v_a,v_b]}$ is associated with the attribute pair (L, ϕ), which refers to transmission duration and offset on the link. The transmission duration of frame on a link $f_{i,m}^{[v_a,v_b]}.L$ is calculated by $\frac{f_i.size}{[v_a,v_b].s}$. $f_{i,m}^{[v_a,v_b]}.\phi$ is the start time (offset) for the frame transmission on link $[v_a, v_b]$ in the period. If there is no proper offset for any frame $f_{i,m}^{[v_a,v_b]}$, f_i is unscheduable, denoted as $S(i) = 0$, otherwise, $S(i) = 1$.

3.3 Scheduling Problem and Constraints

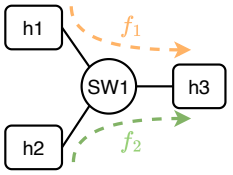
In this section, we introduce the scheduling problem and the constraints. For each TSN flow, the scheduling algorithm outputs a route from its source to the destination and assigns offset for all the links along the route to form the GCLs on switches. To guarantee the deterministic transmission of all the flows, the selection of frame offset ϕ and its route Rt should comply with the following constraints within a hyper period containing all the frame occurrences, which is the least common multiple (LCM) of all the periods of flows as in Eq. 2.

$$T_{sched} = \mathrm{LCM}(F.periods). \tag{2}$$

To stress it clearly, we utilize a simple scenario with two time-sensitive flows f_1 and f_2 in Fig. 3a. The constraints of flow transmissions are shown accordingly in Fig. 3b.

Frame Constraint[1]: The offset of any frame has to be greater than or equal to 0. The entire transmission window (offset plus transmission duration) has to fit within the period.

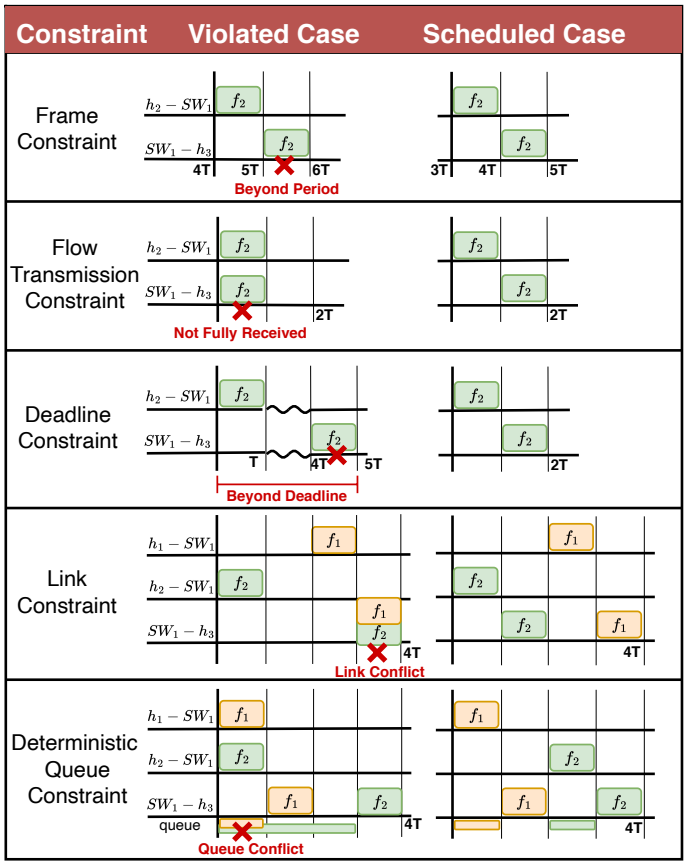
[1]The formulas of constraints are available in Appendices.



Flow Feature

Flow	Src	Dst	Period	ddl	Rt
f1	h1	h3	5T	4T	h1-SW1-h3
f2	h2	h3	5T	4T	h2-SW1-h3

(a) Example scenario. 1T represents a time slot.



(b) Scheduling constraints illustration of f_1 and f_2.

Fig. 3. Scheduling problem and constraints illustration.

Flow Transmission Constraint: The frame propagation of a flow must follow the sequential order along the routed path.

Deadline Constraint: The end-to-end latency cannot exceed the deadline.

Link Constraint: Two frames routed through the same physical link in the network can not overlap in the time domain. As Fig. 3a shows, the hyper period of f_1 and f_2 is 5T, thus the length of resource state is 5 time slots. In the violated case of Link Constraint in Fig. 3b, the link resource state of $SW_1 - h_3$ impacted by $f1$ and f_2 are both [0, 0, 0, 1, 0], which conflict with each other at time slot 4T, indicating potential frame loss. After scheduling, the ultimate link resource of

$SW_1 - h_3$ is [0, 1, 0, 1, 0]. To choose a proper offset on a link ($SW_1 - h_3$), the link resource on the current link is required to examine the Link Constraint.

Deterministic Queue Constraint: Qbv standard reserves eight queues for flows of different priorities, among which time-sensitive flows are assigned to one queue that is of the highest priority. To guarantee the deterministic arrival order, queue-sharing flows must be scheduled so their arrival times are far apart to avoid interleaving. Thus, the constraint isolates two different flows such that one flow can be transmitted to a shared queue only after the other flow is dispatched from the queue. As Fig. 3b shows, $f1$ and $f2$ shares the same queue of SW_1 on link $SW_1 - h_3$. Here we introduce the queue resource to depict this constraint. In the violated case of Deterministic Queue Constraint, the queue resource of SW_1 (also say link $SW_1 - h_3$) impacted by f_1 and f_2 are [1, 0, 0, 0, 0] and [1, 1, 1, 0, 0], respectively, which conflict each other at time slot 1T, meaning f_2 starts to occupy the queue before f_1 leaves it. Taking f_2 as an example, to determine the offset on the current link $h_2 - SW_1$, the queue resource information on the next link $SW_1 - h3$ ([1, 0, 0, 0, 0]) is needed to examine the Deterministic Queue Constraint as the offset on link $h_2 - SW_1$ determines the start time of queue occupation on the next link. As a result, the offset of f_2 on $h_2 - SW_1$ is adjusted from 0 to 2.

Objective: The optimization objective is maximizing the number of successfully scheduled flows formalized as Eq. 3:

$$\textbf{Maximize} \sum_{f_i \in F} S(i). \tag{3}$$

4 DIRTS FRAMEWORK

4.1 Overview

Fig. 4 depicts the overview of DiRTS. The scheduling framework is implemented on two components: CNC and field devices including both end devices and switches. CNC gathers the flow requirements and network topology, which are fed into the route agent to generate flow routes. Each field device with a time agent deployed conducts time planning and GCL configuration autonomously when the flow arrives. The training of agents is guided by the scheduling constraints and the optimization objective to guarantee the quality of generated schedules.

4.2 Locality Analysis

To identify tasks suitable for distributed processing, we conduct an analysis of the locality characteristics of route and time planning tasks.

Route Planning Task: To generate routes for flows, various attributes of edges between the source and destination nodes need to be compared such as the edge-to-edge reachability and distance. Since the above attributes are not self-discovered by a single node[36], they need to be transmitted and aggregated iteratively among edges in the distributed scheme. As Fig. 5a shows, the transmission distance is 3 for device N_6 to get the information of N_1. As the network enlarges, the largest transmission distance between any two nodes increases, raising the transmission cost. Thus the distribution scheme is not suitable for route planning tasks. To avoid the increasing transmission cost, the route planning agent works in CNC centrally, collecting the knowledge of network topology via topology discovery[24] to generate the route.

It is worth mentioning that the centralized deployment of the route planning task does not contradict the distributed reinforcement learning framework for 2 reasons: 1) For transmission cost, the requisite network information for the route planning task is the spacial knowledge in topology connection of network nodes, which is limited and fixed compared to the time-varying knowledge in network resource distribution required by the time planning task. It only

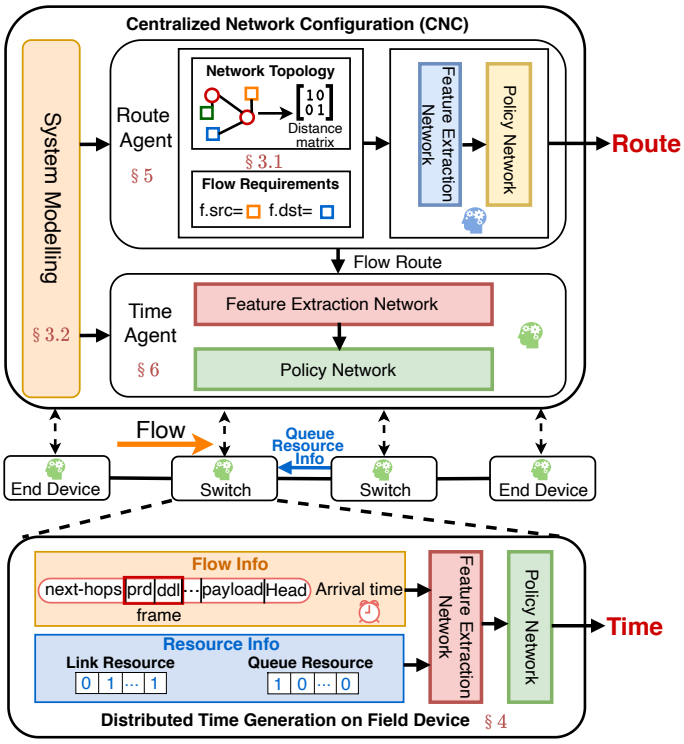


Fig. 4. Overview of DiRTS. The route agent is deployed in CNC to generate routes for flows. The time Agent is deployed in both CNC and field devices. The time agent in CNC is used for module updating in the training phase. The time agents in field devices are used for action generation, i.e., distributed time schedule generation in both training and scheduling phases.

needs a one-time initiation and triggers no extra transmission cost in dynamic flow variations. Besides, the topology knowledge is available with the topology discovery function of CNC specified in IEEE 802.1Qcc[13]. 2) For configuration complexity, the calculation of the centralized route planning task triggers no additional configuration. Additionally, its results serve as the input of the time agent, which promotes the execution of the distributed time planning task to avoid the complex GCL configurations by CNC.

Time Planning Task: The offset time of time-sensitive flows on each hop must satisfy 5 constraints as demonstrated in Sec. 3.3. Among them, the Frame Constraint, Flow Transmission Constraint, and Deadline Constraint are all related to flow attributes, i.e., period, arrival time, and deadline, which are carried with flow itself. The Link Constraint and Deterministic Queue Constraint are related to the link resource on the current link and the queue resource on the next link respectively, which are available within two links. Thus, the requisite information to examine the constraints on time planning tasks is locally accessible on each hop, meaning the offset per hop can be inferred distributedly at the corresponding field device.

The information accessing pattern for time calculation is demonstrated in Fig. 5b. When the flow reaches a certain device, e.g., N_1, the queue resource of the next link ($N_3 - N_5$) and the link resource of the current link ($N_1 - N_3$) are required on device N_1 to calculate the offset time on link $N_1 - N_3$. Since the queue resource on the next link is accessible

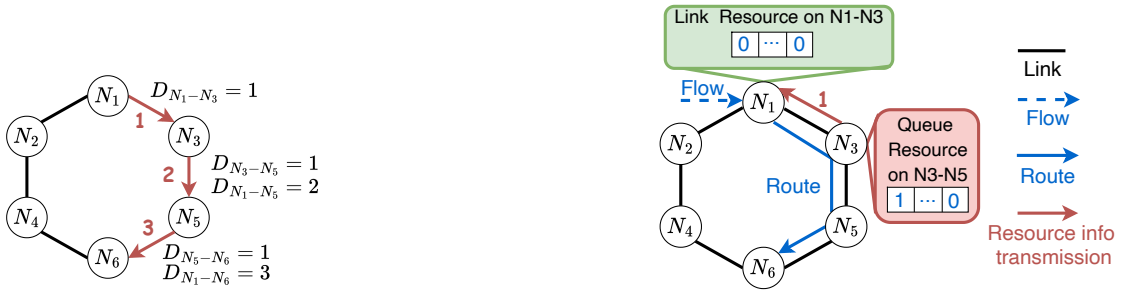


Fig. 5. Information transmission and aggregation needed for (a) distributed route planning task and (b) distributed time planning task. In (a), the longest transmission distance is 3. In (b), the largest transmission distance remains constant at 1.

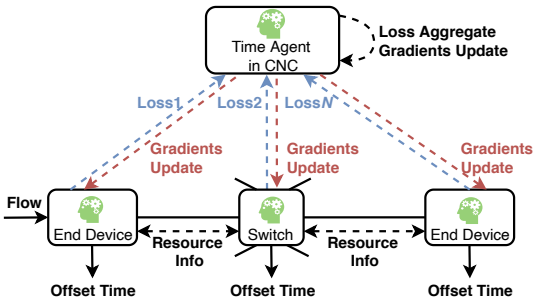


Fig. 6. Training process of DRL time agents. The time agent on the top is deployed in CNC. 1) Parameter Broadcasting refers to the Gradients Update from the time agent in CNC to agents in field devices. 2) Local play is the process of offset time inference and loss generation by the local time agent. The time agents in the field devices treat flow information and resource information as input and output offset time and loss on each hop. 3) Central Update refers to the loss aggregation and gradients update of the time agent in CNC based on the losses.

on the adjacent node ($N3$) and the link resource is already available on N_1, the largest transmission distance remains constant at 1 irrespective of network expansion, demonstrating the great locality of the time planning task.

4.3 Distributed Working Pattern

To save transmission costs, we employ the distributed scheme for time planning tasks. The working pattern can be divided into two phases: training and scheduling.

Training Phase: As Fig. 6 shows, the training process of distributed reinforcement learning for the time planning task contains 3 steps: *1) Parameter Broadcasting:* In the kth training round, denoting the model parameter of the time agent in CNC as θ_k, CNC transmits θ_k to all the filed devices. *2) Local Play:* Each device forwards flows. Specifically, when a flow reaches the device D_i ($i = 1, 2, ..., |D|$) along its route, the device gathers the local information and feeds it into the time agent θ_k. The time agent then generates an offset time when the device forwards this flow to the next hop. Upon the generation of the offset time, the loss $L_i(\theta_k)$ (The opposite of reward expectation, see Sec. 5.3) is derived by the local time agent to evaluate the quality of this output. *3) Central Update:* At the end of round k, the time agent in CNC aggregates all the losses generated by the field devices and updates the model parameter to θ_{k+1} based on Gradient

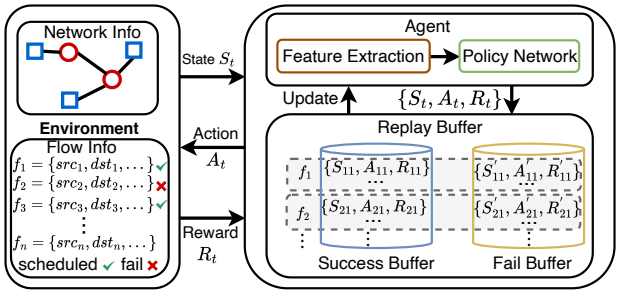


Fig. 7. The structure of reinforcement learning for route planning task in DiRTS. The agent generates an action according to the current state of the environment. A reward is generated by the environment as the feedback of the action. The history data of the triples are stored in a replay buffer to update the agent till convergence.

Descend (GD) as in Eq. 4.

$$\theta_{k+1} = \theta_k - \alpha \nabla \sum_{i=1}^{|D|} L_i(\theta_k), \tag{4}$$

where α refers to the learning rate. The training process repeats till the time agent converges. Note the training contributes no communication cost to the scheduling process since training is a one-time effort independent of scheduling.

Scheduling Phase: For the scheduling phase, agents on different devices work hop-by-hop, following the order of the flow route. The scheduling computation on each hop utilizes the remaining time of frame forwarding within one time slot, without consuming extra time slots. In this paper, each flow arrives incrementally in dynamic scenarios, thus each agent schedules one flow at a time.

5 ROUTE AGENT

Route agent aims to generate the route planning for flows. As Fig. 7 shows, the reinforcement learning structure for the route planning task is composed of two parts: agent and environment. In the process of reinforcement learning, according to the current state S_t given by the environment, the agent outputs an action A_t, and then a reward R_t is generated by the environment to update the agent, which aims to maximize the reward. The agent keeps on updating its model till convergence through the exchange of S_t, A_t, and R_t. As is mentioned in Sec. 4.2, the requisite network information for the route planning task is the fixed spatial knowledge in the topology connection of network nodes, rather than the time-varying information of network resources. Thus the state modeling of the route agent excludes the knowledge of network resources, which further reduces network information uploading cost as Fig. 1 shows. By transmitting and utilizing the network resource information locally, it enables the distributed implementation of the time planning tasks and decouples the route planning and time planning tasks.

5.1 State Modeling

As Fig. 8 shows, the state modeling is achieved by the feature extraction network, which initially extracts the raw feature from the environment, and leverages Graph Neural Network (GNN) to generate the GNN feature. Then, the above two features are concatenated to represent the State S_t.

Raw Feature: For each edge e, we utilize r_e to represent its raw feature, which is composed of raw space feature α_e and raw resource feature β_e. Raw space feature α_e depicts the space characteristic of e in the network topology. It consists of

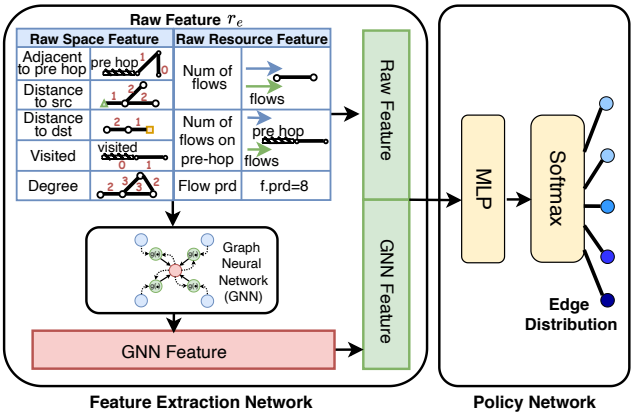


Fig. 8. The model of route agent. The model of composed of a feature extraction network and a policy network.

5 parts: 1) Whether this edge is adjacent to the edge of the last action. 2) The distance between this edge and the source node of the current flow. 3) The distance between this edge and the destination node of the current flow. 4) Whether this edge has been visited. 5) The number of adjacent edges. Raw resource feature β_e depicts the resource state of edge e and features of the current flow. It consists of 3 parts: 1) The degree of congestion on this edge, which is the number of scheduled flows on this edge. 2) The degree of congestion on the previous-hop edge. 3) The period of current flow.

GNN Feature: To efficiently adapt to network topologies and fully extract topology features, we apply GNN to get the embedding vector of each edge. For each link e, GNN takes its raw feature vector r_e as input and performs per-edge embedding. Edge e absorbs information from all its neighbors and the output embedding vector is denoted by Eq. 5:

$$r_e^k = g[\sum_{u \sim \epsilon(e)} f(r_u^{k-1}) + r_e^{k-1}],\ k = 1, 2, ... \tag{5}$$

where f and g are both activation functions, and $\epsilon(e)$ denotes all neighbor edges of e. Each edge collects all information of its k-order neighbors by iterating this computation k times. The ultimate feature vector v_e representing any edge e of state S_t is $v_e = concat(r_e || r_e^k)$.

5.2 Action Modeling

The action is generated by the policy network. As Fig. 8 illustrates, the policy network is composed of two parts: multiple layer perception (MLP) and a softmax layer. The input of the policy network can be denoted as a matrix $M_{route} = [v_1^T, v_2^T, \ldots, v_m^T] \in R^{m \times n}$, where m is the number of edges and $v_i \in R^n$ is the feature vector of the ith edge. The MLP takes M_{route} as input and generates score vector $\mathbf{s} = [s_1, s_2, \ldots, s_m]$ for all the edges. Thereafter, softmax layer turns $\mathbf{s}$ into probability distribution of the edges with softmax function:

$$p_i = \frac{exp(s_i)}{\sum_{j=1}^{m} exp(s_j)}. \tag{6}$$

The edge is sampled by the probability distribution.

5.3 Model Updating

Policy Gradient: To update the policy network, we use the policy gradient algorithm by calculating the gradient of the expected reward relative to the parameters. In the policy gradient algorithm, the goal of the route agent is to find an

Algorithm 1: Route Agent Training Procedure.

Input: Flow set F, Network topology G, Sample number N
Initialize agent parameter θ, replay buffer Bf
for $episode \leftarrow 1 : K$ **do**
  $F_{fail\_set} = \emptyset$
  for *each* $f_i \in F$ **do**
    $f_i.state = InSchedule, Trajectory_{set} = \emptyset$
    while $f_i.state == Inschedule$ **do**
      Get state S_t, sample link $link$ by $p(link|S_t)$
      Take action $A_t = link$, get reward R_t and $f_i.state$
      Store transition (S_t, A_t, R_t) in $Trajectory_{set}$
      if $f_i.state == Success$ or $Fail$ **then**
        break
      end
    end
    for *each* $(S_t, A_t, R_t) \in Trajectory_{set}$ **do**
      Decay and update reward $R_t = \sum_{q=t}^{T} \gamma^{q-t} R_q$
    end
    if $f_i.state == Fail$ **then**
      $F_{fail\_set} \leftarrow f_i$
      $Store\_in\_buffer(Trajectory_{set}, Bf.fail\_buffer)$
    else
      $Store\_in\_buffer(Trajectory_{set}, Bf.suc\_buffer)$
    end
  end
  // Learn from failed flows in replay buffer
  for *each* $f_j \in F_{fail\_set}$ **do**
    $Sample\_transitions(N, f_j, Bf.fail\_buffer)$
    $Sample\_transitions(4N, f_j, Bf.suc\_buffer)$
    Exploit Eq. 7 and Eq. 8 to update parameter θ.
  end
end

optimal strategy $\pi : S \times A \rightarrow [0, 1]$, to maximize the reward value expectation $R_\pi = E[\sum_{t=0}^{n-1} \gamma^t R(S_t, A_t)]$, where S, A and γ refer to state, action and reward decay factor respectively. The gradient of the policy network is derived as Eq. 7.

$$\nabla_\theta R(\pi_\theta) = E_{A\sim\pi_\theta}[\nabla_\theta log(\pi_\theta(A|S))R(S, A)]. \tag{7}$$

Here θ refers to all parameters of the policy network, which is updated as Eq. 8:

$$\theta = \theta + \alpha \nabla_\theta R(\pi_\theta), \tag{8}$$

where α is the learning rate.

In the above analysis, $R(S_t, A_t)$ refers to the reward function. Since we aim to find the route path for as many flows as possible, we set the reward function as follows:

$$R(S_t, A_t) = \zeta \times R_{finish} + \eta \times N_{hop}, \tag{9}$$

$$R_{finish} = \begin{cases} 2, flow\ reaches\ destination \\ 0, in\ progress \\ -1, schedule\ failed \end{cases} \tag{10}$$

where N_{hop} refers to the number of hops the flow goes through at state S_t, which avoids flow picking up redundant edges. ζ and η are both hyperparameters.

Algorithm Design: Alg. 1 shows the training procedure of the route agent. We train the route agent for K episodes. In each episode, we find routes for all the flows in lines 4-12. During flow routing, we get a transition (S_t, A_t, R_t) by finding an edge in each hop until the flow is successfully routed or fails. Lines 14-16 perform reward decay and then all the transitions are stored in the replay buffer according to the flow routing state (i.e., Success or Fail) in lines 17-22. In the training procedure (lines 25-29), we first sample training data of unsuccessfully routed flows from the replay buffer as in Fig. 7 and then exploit the data to update the parameter θ of the policy network. Specifically, as shown in Fig. 7, the replay buffer is composed of a success buffer and a fail buffer. A flow would be successfully scheduled or fail to be scheduled in different episodes. The trajectories of each flow in different episodes are stored in the success or fail buffer according to the scheduling results in different episodes. Thus, trajectories would be in both success buffer and fail buffer for a specific flow. We take the training of route agent which schedules three flows $f_1 - f_3$ as an example. At the start of the k-th episode, we first initiate the flow set of unscheduled flows F_{fail_set}. Then we schedule each flow with the current agent parameter. Taking the scheduling of f_1 as an example, the selection of each link is an action till scheduling success (finding the feasible path) or failure of f_1 occurs (a specific link selection fails to constitute a feasible path). The scheduling process of f_1 till success or failure generates a sequential trajectory of all the transitions $[(S_1, A_1, R_1), (S_2, A_2, R_2), ..., (S_t, A_t, R_t)]$, which is stored in $Trajectory_{set}$. Then we perform reward decay for all the reward values $R_n (1 \leq n \leq t)$ in $Trajectory_{set}$. We store $Trajectory_{set}$ of f_1 in success buffer or fail buffer according to its scheduling result in the k-th episode. After the scheduling of all the flows $f_1 - f_3$ is finished, we store the unscheduled flows into F_{fail_set}. In the learning phase of the k-th episode, we learn from the historical trajectories of these failed flows. Assuming f_1 fails to be scheduled, first, we sample transitions (S_n, A_n, R_n) of f_1 from f_1's historical success buffer and fail buffer. Since we intend to let the route agent learn from the successful experiences. We sample 4N transitions from the successful buffer and N transitions from the fail buffer. Then we exploit Eq. 7 and Eq. 8 with the sampled transitions to update the parameters of the route agent. We keep repeating the above process for K times.

6 TIME AGENT

Time agent aims to generate offset time on each hop of the flow. The structure of DRL process for time planning tasks is similar to that of route planning tasks shown in Fig. 7, which takes the network information and flow requirements as state and generates the offset time slot at the current hop as an action.

6.1 State Modeling

As Fig. 9 shows, the feature extraction network initially extracts raw features from the environment and then adds positional encoding to it, which then forms the current state S_t.

Raw Feature: As analyzed in Sec. 4, offset determination shows great locality by considering flow attributes, the resource on the current link and the queue resource on the next link. Thus the raw feature of time determination is categorized in these three aspects as Table 1. As both link and queue resources of the current link are available on the current hop, we incorporate both of them in the feature to fully extract the network information. In the raw feature, the resource of the link or queue is represented as a vector with a length of the hyper period. For example, if flow f_i passes through link a, with the offset, the period and the hyper period being 2, 4, and 8, respectively, the link resource of link a is $[0, 0, 1, 0, 0, 0, 1, 0]$, with each element representing a resource block. Each feature shown in Table 1 is depicted in a $0/1$ vector vec_i, thus the raw time feature on each hop is $Feature_{raw} = [vec_1, vec_2, \ldots, vec_7]$, the size of which is $7 \times hyperperiod$.

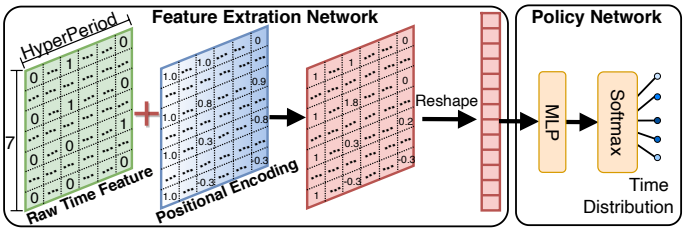


Fig. 9. The model of time agent. The feature extraction network is composed of raw time features and positional encoding.

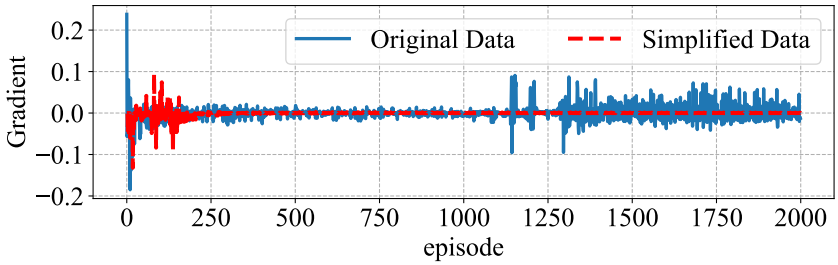


Fig. 10. The training gradients of two scenarios: using 1) original data and 2) simplified data to train agent.

Positional Encoding: To enable the model to make use of different feature vectors, we inject information about the relative position of the feature vectors[37] by adding positional encodings to the raw time feature $Feature_{raw}$. The positional encodings have the same size as $Feature_{raw}$. The computation of positional encodings is shown in Eq. 11:

$$PE(pos, 2i) = sin(\frac{pos}{10000^{2i/d_{model}}})$$
$$PE(pos, 2i+1) = cos(\frac{pos}{10000^{2i/d_{model}}}), \tag{11}$$

where pos is the position of the encodings (i.e., the row number), d_{model} is the column width, and $i = 0, 1, \ldots, d_{model}/2-1$.

6.2 Action Modeling

For the time agent, each action corresponds to an offset decision on the current link. As illustrated in Fig. 6, the time agent works in a distributed learning scheme. Each device generates its action by choosing a proper offset on the current link. The policy network of the time agent is composed of a MLP and a softmax layer. The input of the policy network can be denoted as a matrix $M_{time} = Feature_{raw} + PE \in R^{7*hyperperiod}$. The MLP takes M_{time} as input and generates score vector $s = [s_1, s_2, \ldots, s_{hyperperiod}]$ for all the time slots within hyperperiod. Thereafter, the softmax turns s into a probability distribution, according to which the time slot is sampled.

6.3 Model Updating

Training Framework Design: To update the model of the time agent, we use policy gradient as described in Sec. 5.3. However, the complex constraints and limited local knowledge lead to model instability. As Fig. 10 shows, if we feed the original network and flows into the model, the gradients of the model oscillate, indicating that the agent learns

Table 1. Raw Feature of Time Agent.

Feature Class	Feature Content	Relevant Constraint
Flow Attributes	Flow Period	Frame Const
	Flow Deadline	Deadline Const
	The Number of Left Hops to Dst	Frame & Deadline Const
	Offset Time on the Previous Hop	Flow Transmission Const
Resource on the Current Link	The Link Resource	Link Const
	The Queue Resource	Deterministic Queue Const
Resource on the Next Link	The Queue Resource	Deterministic Queue Const

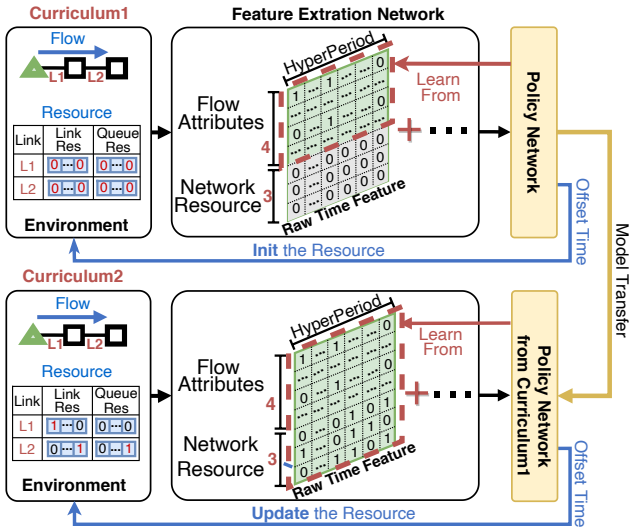


Fig. 11. Training curriculums for time agents. Curriculum 1 inits all the resource vectors to 0. The policy network only learns valid information from the flow attributes in the raw time feature. Curriculum 2 updates resource vectors after each scheduling action. The policy network learns from both valid information of flow attributes and network resources.

nothing from the original data. Surprisingly, we observe that the gradients will converge if we simplify the training data by removing the resource conflict among flows to make the agent only learn part of the scheduling constraints. It gives us a hint that we can divide training into different phases based on the complexity of constraints rather than make the agent learn all the constraints once and for all to avoid model instability.

As a result, we customize curriculums for the agent in different phases as Fig. 11. *In Curriculum 1*, we train the time agent to obey the constraints relevant to the flow attributes. To achieve that, each flow in the training set is scheduled after the network resource is set to the initial state, i.e., no link or queue in the network is occupied. *In Curriculum 2*, after the agent learns flow attributes, we train the agent to satisfy the Link and Deterministic Queue Constraints relevant to the network resource. To this end, after each flow is scheduled, the queue and link resources of

Table 2. Reward Design.

Reward	Condition
$\xi - [(offset + N_{leftHop}) - period]$	$offset > period - N_{leftHop}$
$\xi - (offset_{previousHop} - offset)$	$offset < offset_{previousHop}$
$\xi - [(offset - offset_{firstHop}) - ddl]$	$offset - offset_{firstHop} > ddl$
$\xi - N_{link\_conflict}$	$link\ resource\ conflicts$
$\xi - N_{queue\_conflict}$	$queue\ resource\ conflicts$

each link must be updated accordingly, from which the agent learns to avoid resource conflict, and thus to obey Link and Deterministic Queue Constraints.

Reward Design: The formula of model updating also follows Eq. 7 and 8. The difference between route agent and time agent updating is the reward definition as the two tasks comply with different rules. Since the offset determination considers the constraints in Table 1, the rewards are defined accordingly as Table 2, in which the reward is generated based on the degree to which different constraints are followed or violated. For example, if the generated offset is less than the offset on the previous hop (i.e., violating Flow Transmission Constraint), the reward must deduct from the difference to penalize it. Here, the $N_{leftHop}$ is the number of left hops to the destination, $offset$ is the action taken on the current link, $offset_{previousHop}$ and $offset_{firstHop}$ are the offset time on the previous link and the first link, $N_{link\_conflict}$ and $N_{queue\_conflict}$ are the numbers of resource blocks where conflict happens on link and queue respectively, and ξ is the hyperparameter.

7 EVALUATION

7.1 Experimental Setup

Baseline: We compare DiRTS with six centralized algorithms, i.e., SPF_early [12], DRLS+LD [42], DeepScheduler (DS) [15], TTDeep [18], ALAP [30], and ILP [1]. SPF_early searches the shortest path and allocates the earliest valid slot to each hop. DRLS and DS both search routes with DRL while the time slot is assigned in Low Degree (LD) and As Soon As Possible (ASAP) heuristic, respectively. TTDeep searches both route and time slot with DRL. ALAP searches the shortest path and allocates the time slot as late as possible to minimize transmission delay. ILP generates the optimal schedules with ILP solvers by traversing the whole search space.

Network Setting: We conduct experiments on three typical industrial topologies: ring, linear, and tree [41]. The network scale range is 6-40 nodes. The number of end devices connected to each switch is randomly selected from the set $\{1, 2, 3\}$. The link bandwidth is 1Gbps and the time slot length is set to $250\mu s$ to allow the transmission of one Maximum Transmission Unit (MTU)-sized frame and the scheduling computation on each hop. For flow generating, the source and destination of each flow are randomly selected. The frame size is randomly generated from 64 to 1500 Bytes. The period is selected from 2, 4, 8, 16 time slots. The deadlines are restricted randomly from 20-30 ms. We only consider the case of unicast since the multicast flows can be split into multiple unicasts. The combinations of network topologies and flows make up the training set used to train the agents, which is generated with a self-designed TSN dataset generation tool[2]. For the experiment setup in Sec. 7.2, the network parameter and flow attributes are in accordance with the above setting, unless stated otherwise.

[2]https://github.com/gimmyy/TSN-data-set-generation

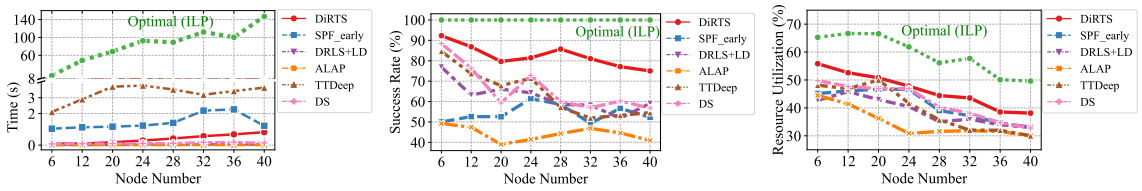


Fig. 12. Performance of algorithms in different network scales. (a) Execution time. The inference time of ILP exceeds DiRTS by 200×. (b) Scheduling success rate. DiRTS remains the highest among time-saving algorithms. (c) Resource utilization.

Network Condition: We focus on dynamic scenarios where each flow is dynamically injected into the network and scheduled incrementally. Once a flow is scheduled, its route and time schedule are fixed, i.e., the resources it takes remain unchanged.

Parameter Setting: The number of iterations in GNN is 8. The lengths of the raw feature and GNN feature of the route agent are both 8. The MLP in the route agent contains 4 FC layers with dimensions 32, 16, 8, and 1. The parameters ζ and η of routing reward are set to 10 and -0.1 respectively. The MLP in time agent includes 4 FC layers with dimensions 128, 64, 32, 16. The parameter ξ in time reward is -40. The activation function is `Leaky_Relu`. Adam optimizer with a learning rate of 0.002 is adopted. The reward decay factor is 0.8 and we train agents for at least 2000 episodes.

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

Evaluation metrics: 1. *Scheduling success rate* is the proportion of successfully scheduled flows relative to the total number of flows. A high scheduling success rate indicates that most flows are scheduled successfully. Note that we set all network flows as time-sensitive flows in our setup to understand the performance of various approaches. In practice, time-sensitive flows typically occupy a small proportion of all network flows. Therefore, even achieving a success rate of less than 100% can still effectively handle all flows, depending on the proportion of time-sensitive flows [39]. 2. *Resource utilization* is the proportion of the allocated time slots over the total time slots on all the links. A high resource utilization means the devices are fully utilized. 3. *Execution time* is the time spent to infer the schedule results of all the flows, which is expected to be minimized.

7.2 Experimental Results

7.2.1 *Performance in Different Network Scales.*

Setup: We set the node number of the network from 6 to 40. We fix the flow number at 200 and adopt ring topology to test the scheduling performance of all the algorithms. Since ILP can produce the optimal solution, we normalize the outcome of other algorithms with the results of ILP.

Results: Fig. 12a shows the execution time of different algorithms. The execution time of ILP grows exponentially with the network scale, which is intolerable in time-critical industrial scenarios. In contrast, the execution time of DiRTS is below 1s. In the 40-node scenario, the overall execution time is 0.82s, meaning the scheduling time cost for a single flow is 4.1ms on average. Since the interval of dynamic flow requirement should be no less than the flow deadline (i.e., 20-30ms), the least flow interval is 30ms. For this interval value per new flow requirement, the scheduling time of 4.1ms is feasible. DRLS+LD takes less time because the time planning part exploits the heurist method which selects the time slot with the most adequate resources. This heuristic calculation is a one-step judgment that gets rid of iteration and

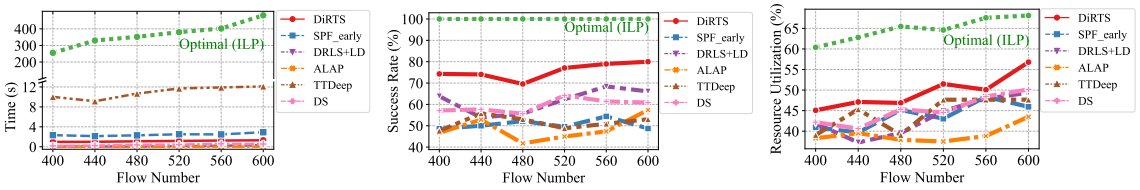


Fig. 13. Performance of algorithms in different traffic volumes. (a) Execution time. The inference time of ILP exceeds DiRTS by 200×. The execution time of all algorithms increases with the flow number but only that of ILP is recognizable in the time scale. (b) Scheduling success rate. DiRTS remains the highest among time-saving algorithms. (c) Resource utilization.

search. DS uses more time because, in the time planning task, it conducts an iterative search in case of the infeasibility of the earliest time slot, which is more time-consuming. As Fig. 12b shows, the average success rate of the proposed DiRTS is the highest among the time-saving algorithms, exceeding DRLS+LD by 20.31% on average. The reason is that DiRTS dynamically learns the generic scheduling rules and the current network state instead of following the solid heuristic, leading to great generalization and scalability in different scenarios. Meanwhile, DiRTS integrates queue resources into the features of the time agent, making the network representation more comprehensive and thus enhancing its exploration ability. For all the centralized methods, only ILP outperforms DiRTS since it takes a global view of the whole network to generate schedules at the cost of large time consumption. Although other centralized methods collect global knowledge, they use it separately by solving the scheduling problem hop by hop and utilizing only local knowledge to guide the scheduling at each hop, which fails to take advantage of the global view. As mentioned before, the scheduling success rate does not have to be 100% because, in practical scenarios, time-sensitive flows typically occupy a relatively modest proportion, and a lower success rate than 100% could sufficiently cover all the flows [39]. As Fig. 12c shows, the resource utilization of DiRTS remains the highest among the time-saving algorithms. The average increases of DiRTS compared to SPF_early, DRLS+LD, DS, ALAP, and TTDeep are 4.32%, 6.97%, 3.94%, 11.86%, and 4.02%. The resource utilization increase is contributed by the increase of successfully scheduled flows, which occupies more idle resources. The reason for the resource utilization degradation of all the algorithms as the network enlarges is that the total flow number is fixed while the overall network resource keeps growing.

7.2.2 *Performance in Different Traffic Volumes.*

Setup: We set the flow number from 400 to 600 and adopt the 40-node ring topology to test the scheduling performance of all the algorithms.

Results: Fig. 13a shows the execution time of algorithms in different traffic volumes. At flow number 600, the execution time of ILP exceeds the DiRTS by 200×, which is unacceptable in time-critical industrial scenarios. In contrast, the execution time of DiRTS is below 2s. Fig. 13b depicts the scheduling success rate of different algorithms. The average gap between DiRTS and DRLS+LD is 13.49%. At flow number 600, the scheduling success rate of DiRTS reaches 80%, outperforming DRLS+LD by 13.91%. DiRTS shows great scalability to dynamic flow requirements as the scheduling success rate increases with the dynamic flow number. The reason is that the RL agent acquires robust feature extraction capability to cope with diverse flow sets. As Fig. 13c shows, the resource utilization of DiRTS remains the highest among the time-saving algorithms. The average increases of DiRTS compared to SPF_early, DRLS+LD, DS, ALAP and

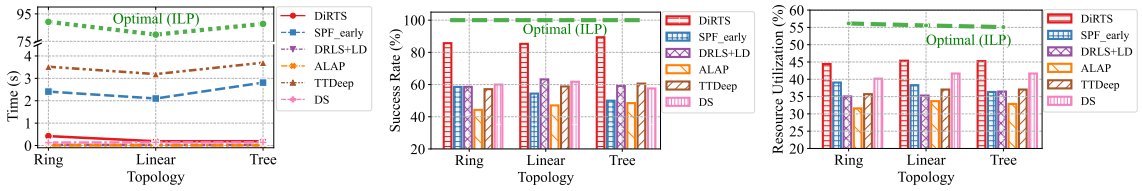


Fig. 14. Performance of algorithms in different topologies. (a) Execution time. The inference time of ILP exceeds DiRTS by 200×. (b) Scheduling success rate. DiRTS remains the highest among time-saving algorithms. (c) Resource utilization.

Table 3. Scheduling success rate comparison of RA+TA, SPF+TA, and RA+ASAP.

Node	**DiRTS(RA+TA)** (%)	SPF+TA (%)	RA+ASAP (%)
24	**80.00**	55.00	62.50
28	**84.85**	60.61	69.70
32	**70.59**	55.89	50.00
36	**79.41**	55.88	61.76
40	**80.56**	58.33	63.89

TTDeep are 5.09%, 5.28%, 3.72%, 9.68% and 5.17%. The reason for the resource utilization increases with the flow number is that the scheduled flows increase while the total network scale and resources remain constant.

7.2.3 Performance in Different Topologies.

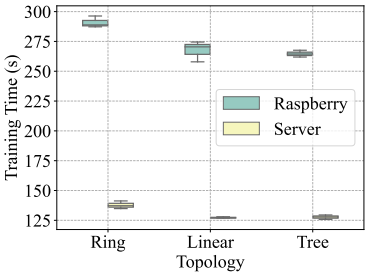
Setup: We set the network topology as Ring, Linear, and Tree respectively to test the robustness of DiRTS. We fix the node number and flow number at 28 and 200, respectively.

Results: Fig. 14a shows the execution time of algorithms in different topologies. The execution time of ILP exceeds 80s in all topologies while that of DiRTS still remains below 1s. In Fig. 14b, the scheduling success rate of DiRTS is 27.14%, 22.06%, and 30.30% higher than that of DRLS+LD in Ring, Linear, and Tree topology, respectively. It demonstrates that the proposed DiRTS shows great generalization because of complete training data and the adoption of constraints-based curriculum learning. As Fig. 14c shows, the resource utilization of DiRTS remains the highest among all the time-saving algorithms. The average increases of DiRTS compared to SPF_early, DRLS+LD, DS, ALAP, and TTDeep are 7.11%, 9.41%, 3.84%, 12.30% and 8.44%. The resource utilization increase is contributed from the increase of successfully scheduled flows, which occupies more idle resources.

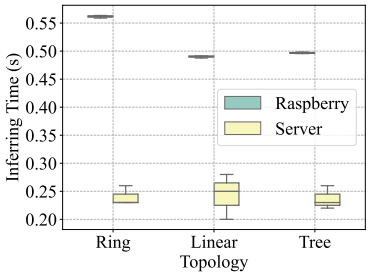
7.2.4 Effectiveness of Route Agent and Time Agent.

Setup: To test the effectiveness of Route Agent (RA) and Time Agent (TA) respectively, we compare the scheduling success rate of DiRTS (RA+TA) to ShorestPathFirst+TA (SPF+TA) and RA+AsSoonAsPossible (RA+ASAP). We choose the flow number from 100-300 randomly and set the network scale to 24-40 nodes.

Results: The performance of DiRTS, SPF+TA, and RA+ASAP are shown in Table 3. Averagely, the scheduling success rate of DiRTS exceeds SPF+TA by 21.94% because the route agent learns to avoid congestion by incorporating the number of scheduled flows in the link feature while SPF lacks the ability to recognize link congestion dynamically. DiRTS outperforms RA+ASAP by 17.51% on average. The reason is that the time agent of DiRTS learns the generic scheduling



(a)



(b)

Fig. 15. (a) The training time comparison and (b) the inferring time comparison of DiRTS on Raspberry Pi and a server. In (a), the three bars below 150s are all tested on Server. In (b), the three bars above 0.45s are all tested on Raspberry Pi.

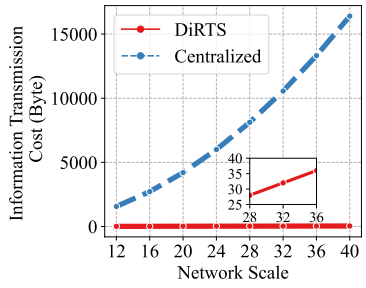


Fig. 16. The information transmission cost of DiRTS and the centralized method in the scheduling process.

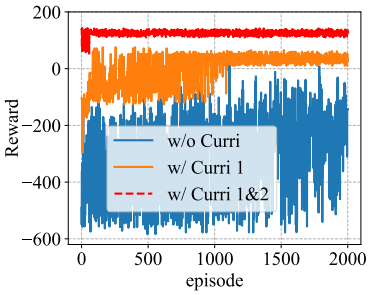


Fig. 17. Training performance comparison of time agents with/without curriculum design.

rules and the current network state instead of following the solid ASAP policies, leading to great generalization and scalability in different network scales.

7.2.5 *Training and Running Overheads of Agents.*

Setup: We train the agents in Raspberry Pi (4 model B) and servers to compare their training and execution costs. Both executions are conducted on CPUs because the majority of field devices are configured with CPUs but no GPU due to financial constraints. The training time is spent for 200 episodes. The numbers of flows and nodes are set to 200 and 28, respectively.

Results: The training and execution costs of agents in Raspberry Pi (Pi) and the server are shown in Fig. 15a. The training time on Pi is on average 2.10 times longer than that on the server due to the limited computational resources of Pi. The training time of Pi is within 300s, which is acceptable for the training process because it is a one-time effort. As for the execution cost shown in Fig. 15b. The total inference time on Pi is below 0.6s, which demonstrates the applicability of DiRTS on resource-constrained industry-level devices.

7.2.6 *Transmission Cost Comparison.*

Setup: To prove the ability of DiRTS to save information transmission cost, we compare the transmission cost of DiRTS and the centralized method (Centralized). The network scale is set to 12-40 switches of linear topology. The average

Table 4. Algorithm Comparison. Extra GCL configuration requires transmitting GCLs from CNC to devices. Uploading resource information means transmitting link and queue resource information from devices to CNC.

Algorithm	Time Efficiency		Functionality		Deployability		
	Execution Time (s)	Success Rate (%)	Route Planning	Time Planning	Avoid Heuristic Expertise	Avoid Extra GCL Configuration	Avoid Uploading Resource Information
SPF_early	2.44	54.33	✗	✓	✗	✗	✗
DRLS+LD	0.03	60.30	✓	✓	✗	✗	✗
DS	0.13	59.78	✓	✓	✗	✗	✗
TTDeep	3.47	58.85	✓	✓	✓	✗	✗
ALAP	0.01	32.71	✗	✓	✗	✗	✗
ILP	86.61	100	✗	✓	✓	✗	✗
DiRTS (Ours)	0.27	86.80	✓	✓	✓	✓	✓

route length of flows is set to half of the network size. For example, the average route length of flows is 10 in a 20-switch network.

Results: The transmission costs of DiRTS and the centralized method are shown in Fig. 16. At the network scale 40, the centralized method incurs 410× the cost of information transmission compared to DiRTS, restricting its scalability. This is due to the global information requirement and per-device GCL configuration of the centralized scheme, which triggers massive communications between CNC and devices before flow admission. In contrast, DiRTS is free of GCL transmission and network resource information uploading thanks to the local computing and configuration capabilities of devices. Meanwhile, each device requires only the local resource information within one link, resulting in a total transmission distance equivalent to the length of the flow route irrespective of the network scale.

7.2.7 *Effect of Training Curriculums for Time Agents.*

Setup: We display the comparison of time agents with/without curriculum adoption. We train the agents in the 20-node ring topology and the flow number is set as 100.

Results: As shown in Fig. 17, Agent1 taking no curriculum is unstable and its model keeps fluctuating. Agent 2 only taking curriculum 1 converges to local optimum in episode 1094 and the fluctuation intensity declines compared to Agent1. Agent 3 taking Curriculum 1 and 2 converges in episode 22 and the average reward exceeds Agent 2 by 71.01%. It demonstrates the effectiveness of training curriculum customization in a constraints complexity-increasing manner.

To summarize, the comparison of algorithms is as Table 4. Different from other DRL-based methods, the route agent of DiRTS only requires fixed spatial knowledge in topology connection of network nodes, rather than the time-varying information of network resources. Thus the state modeling of the route agent excludes the knowledge of network resources, which reduces network information uploading cost as Fig. 16 shows. By transmitting and utilizing the network resource information locally, it enables the distributed implementation of the time planning tasks and decouples the route planning and time planning tasks.

8 VALIDATION ON SIMULATOR AND TESTBED

Validation on Network Simulator: In order to evaluate the feasibility and correctness of DiRTS, we use the network simulator ns.py[19] to verify the schedule calculated by DiRTS. We simulate the schedule generated by DiRTS in ring and tree topologies, which are shown in Fig. 18a and Fig. 18c, respectively. In each network topology, 50 flows are tested.

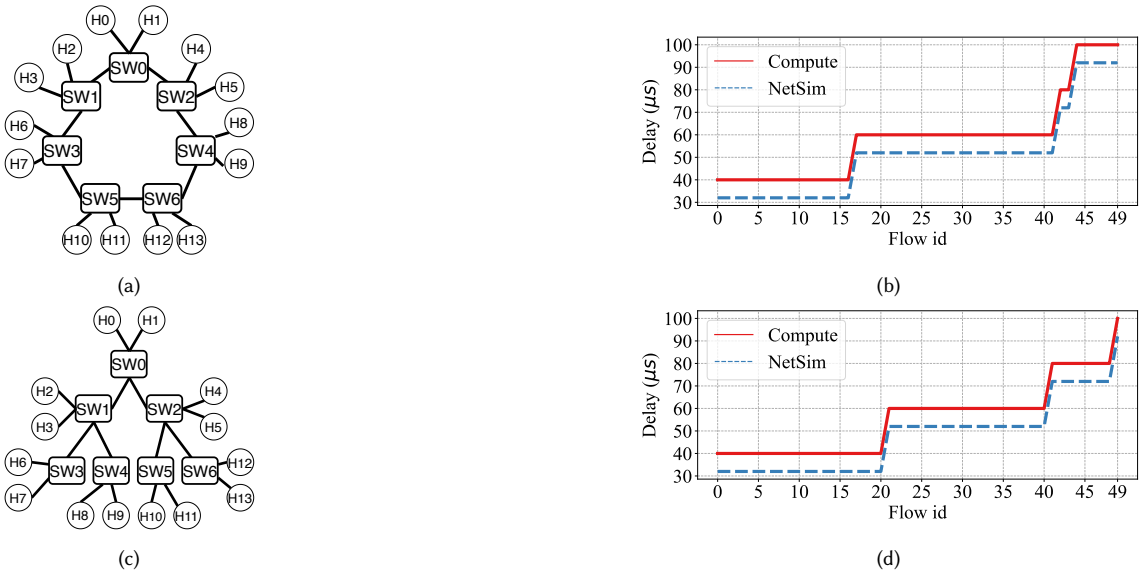


Fig. 18. The simulation topology of (a) ring and (c) tree. $H_0 - H_{13}$ are the hosts, i.e., end devices. The delay of successfully scheduled 50 flows generated in algorithm computation and simulation of (b) ring and (d) tree topology. The flows are sorted by their delays.

The simulation results in ring and tree topology are shown in Fig. 18b and Fig. 18d, respectively. The simulation delays of flows are less than the computation delays because the computation delay is the integer multiples of time slot which is the upper bound of the delay and guarantees the reachability of flows in the worst case. Since the network simulation is more ideal than realistic scenario, the simulation delay is below the computation worst-case delay, which meets the deadline constraints of all the flows and is in line with our expectations. The simulation results show that all flows arrived at destinations on time within their deadlines, demonstrating the feasibility of DiRTS.

Validation on Testbed: The experiment setup is shown in Fig. 19a. The network is composed of 4 end devices (hosts) and 2 switches. The Spirent SPT-N4U Compact Chassis serves as the hosts, which supports TSN standards and can generate user-defined flows. Meanwhile, It can provide statistics for the flows. The version of switches is Moxa TSN-G5004. We randomly choose 10 flows and the schedule is calculated by DiRTS, which is then translated to the GCLs and configured in switches. These flows are then generated and monitored by the Spirent SPT-N4U Compact Chassis.

The experimental results are shown in Fig. 19b. The computation results provide the worst-case delays of all the flows, which exceed the simulation and experimental delays. However, the experimental delays are not so ideal as the simulation since the flows would go through ingress filtering and link propagation delay in real scenarios, causing longer delays compared to the simulation. As a result, the delays in experimental scenarios lie between the computation results and the simulation results, which accords with theoretical analysis and demonstrates the flows in realistic scenarios all meet their deadlines. This experiment shows that the schedule computed by DiRTS is applicable to TSN devices.

9 DISCUSSION

GCL Coordination and Admission Control: If the scheduling of flow f fails on a device, it drops the frame and notifies the devices on the previous hops with a backtracking flow f_b. f_b has a lower priority to ensure existing TSN

(a) (b)

Fig. 19. (a) The experimental setup. (b) The delay of successfully scheduled flows of computation, simulation, and experiment. The flows are sorted by their delays.

flows are not delayed. Upon receiving f_b, previous devices recover the GCL set for f, and the source device stops sending f. We conducted Response Time Analysis [20] to analyze the worst-case delay of f_b, which is assigned Class A priority and transmitted along 10 hops. The worst-case delay was 968.64us (<1ms). In dynamic scenarios, each flow incrementally injects into the network. The injection time interval of flows can be easily controlled larger than this for potential GCL recovery and admission control of unscheduled flows.

Necessity of additional flows for attributes information: As described in Sec 4.2, the scheduling of each distributed time agent gets flow attributes from the flow itself instead of additional flows carrying attributes information. Though flow attributes profiling for TSN remains an open issue, according to UA Specification [10], the QoS requirement could be encoded in the OPC UA Ethernet frame, eliminating the extra cost for additional flows carrying attribute information.

Solution to link/switch failure: As evaluated in Fig. 12 and Fig. 14, DiRTS shows great robustness in different network topologies and conditions. When link/switch failure occurs, topology information is required to be uploaded in CNC for the route agent to update the spatial feature in state modeling. The route agent regenerates another feasible route with the new topology, which allows the time agent to circumvent the problem of link failure. Following that, time agents exploit the local knowledge on the feasible route for offset generation without additional configuration.

Comparison to equivalent decentralized algorithms: As evaluated in Fig. 12, for all the centralized comparison methods, only ILP outperforms DiRTS since it takes a global view of the whole network to generate schedules at the cost of large time consumption. Although other centralized methods collect global knowledge, they use it separately by solving the scheduling problem hop by hop and utilizing only local knowledge to guide the scheduling at each hop, which fails to take advantage of the global view and conducts in a decentralized way intrinsically. Thus, the decentralized version of these methods would maintain the same scheduling performance and inferring time. The performance of traditional Qcc-based approaches that do not use AI can refer to the scheduling results of SPF_early and ALAP algorithms.

10 CONCLUSION

In this paper, we present a distributed reinforcement learning framework for flow scheduling in IEEE 802.1Qbv time-sensitive networks. Specifically, we propose a multi-agent DRL scheduling method by designing DRL-based route and time agents to perform route and time planning tasks. We find the inherent locality property of time scheduling tasks in TSN and distribute time planning tasks to field devices, enabling the time agent to explore the network locally. Further,

we propose a general model training framework to promote the convergence of distributed agents. We also develop a TSN dataset generation tool to ease the evaluation efforts on TSN scheduling research.

The experimental results show that the scheduling success rate with the proposed method is on average 20.31% higher than with the state-of-the-art DRLS, demonstrating its effectiveness. Moreover, it saves the transmission cost by 410× compared with the centralized method, showing great scalability. We successfully implement the designed agents in Raspberry Pi to validate the deployability of the proposed method in resource-constrained industrial devices. At last, simulations and experimental validations on the TSN testbed demonstrate the applicability of the proposed method. For future work, we plan to introduce OPC UA into the current DRL framework to provide a cross-level configuration solution in the OPC UA over TSN architecture.

ACKNOWLEDGMENTS

Thank the editors and reviewers for the valuable comments. This work is supported by the National Science Foundation of China (NSFC) under Grant No. (62302439, U23A20296), and in part by the Fundamental Research Funds for the Central Universities (226-2024-00004). Chaojie Gu is the corresponding author.

REFERENCES

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

[2] Zongrong Cheng, Dong Yang, Weiting Zhang, Jie Ren, Hongchao Wang, and Hongke Zhang. 2022. DeepCQF: Making CQF Scheduling More Intelligent and Practicable. In *ICC 2022 - IEEE International Conference on Communications*. 1–6. https://doi.org/10.1109/ICC45855.2022.9882280

[3] Hao Ran Chi, Maria de Fátima Domingues, Konstantin. I. Kostromitin, Ahmad Almogren, and Ayman Radwan. 2021. Spatiotemporal D2D Small Cell Allocation and On-Demand Deployment for Microgrids. *IEEE Access* 9 (2021), 116830–116844. https://doi.org/10.1109/ACCESS.2021.3105750

[4] Sandeep Chinchali, Marco Pavone, and Sachin Katti. [n. d.]. Cellular Network Traffic Scheduling using Deep Reinforcement Learning. ([n. d.]).

[5] Silviu S Craciunas and Ramon Serna Oliver. 2016. Combined task-and network-level scheduling for distributed time-triggered systems. *Real-Time Systems* 52 (2016), 161–200.

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

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

[8] Jonathan Falk, Frank Dürr, and Kurt Rothermel. 2018. 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, 136–146.

[9] Bernard Fong, A. C. M. Fong, and Kim-Fung Tsang. 2021. Capacity and Link Budget Management for Low-Altitude Telemedicine Drone Network Design and Implementation. *IEEE Communications Standards Magazine* 5, 4 (2021), 74–78. https://doi.org/10.1109/MCOMSTD.0001.2100010

[10] OPC Foundation. 2023. OPC UA Specification Part 14. https://reference.opcfoundation.org/Core/Part14/v105/docs/7.3.3.

[11] Voica Gavriluț and Paul Pop. 2018. Scheduling in time sensitive networks (TSN) for mixed-criticality industrial applications. In *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*. IEEE, 1–4.

[12] OSPF Working Group. 2008. Open Shortest Path First. https://en.wikipedia.org/wiki/Open_Shortest_Path_First.

[13] TSN Group. 2015. IEEE 802.1Qcc Standard. https://www.ieee802.org/1/files/public/docs2015/cc-cummings-topology-discovery-v1.pdf

[14] TSN Group. 2016. IEEE 802.1Qbv Standard. https://www.ieee802.org/1/pages/802.1bv.html

[15] Xiaowu He, Xiangwen Zhuge, Fan Dang, Wang Xu, and Zheng Yang. 2023. Deep-Scheduler: Enabling Flow-Aware Scheduling in Time-Sensitive Networking. In *IEEE INFOCOM*.

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

[17] Huang Huang and Jingjing Li. 2020. Distributed Energy-Aware Reliable Routing and TDMA Link Scheduling in Wireless Sensor Networks. In *2020 29th International Conference on Computer Communications and Networks (ICCCN)*. 1–10. https://doi.org/10.1109/ICCCN49398.2020.9209621

[18] Hongyu Jia, Yu Jiang, Chunmeng Zhong, Hai Wan, and Xibin Zhao. 2021. TTDeep: Time-Triggered Scheduling for Real-Time Ethernet via Deep Reinforcement Learning. In *2021 IEEE Global Communications Conference (GLOBECOM)*. 1–6. https://doi.org/10.1109/GLOBECOM46510.2021.9685850

[19] Baochun Li, Li Chen, and Xi Peng. 2022. ns.py. https://github.com/TL-System/ns.py.

[20] Yuhan Lin, Xi Jin, Tianyu Zhang, Meiling Han, Nan Guan, and Qingxu Deng. 2021. Queue assignment for fixed-priority real-time flows in time-sensitive networks: Hardness and algorithm. *Journal of Systems Architecture* 116 (2021), 102141. https://doi.org/10.1016/j.sysarc.2021.102141

[21] Lucia Lo Bello and Wilfried Steiner. 2019. A Perspective on IEEE Time-Sensitive Networking for Industrial Communication and Automation Systems. *Proc. IEEE* 107, 6 (2019), 1094–1120. https://doi.org/10.1109/JPROC.2019.2905334

[22] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. 2019. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM special interest group on data communication*. 270–288.

[23] John L. Messenger. 2018. Time-Sensitive Networking: An Introduction. *IEEE Communications Standards Magazine* 2, 2 (2018), 29–33. https://doi.org/10.1109/MCOMSTD.2018.1700047

[24] Sanaz Mohammadi, Didier Colle, and Wouter Tavernier. 2022. Latency-aware Topology Discovery in SDN-based Time-Sensitive Networks. In *2022 IEEE 8th International Conference on Network Softwarization (NetSoft)*. 145–150. https://doi.org/10.1109/NetSoft54395.2022.9844085

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

[26] Paul Pop, Michael Lander Raagaard, Marina Gutierrez, and Wilfried Steiner. 2018. Enabling fog computing for industrial automation through time-sensitive networking (TSN). *IEEE Communications Standards Magazine* 2, 2 (2018), 55–61.

[27] Francisco Pozo, Guillermo Rodriguez-Navas, Hans Hansson, and Wilfried Steiner. 2015. SMT-based synthesis of TTEthernet schedules: A performance study. In *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*. 1–4. https://doi.org/10.1109/SIES.2015.7185055

[28] Francisco Pozo, Wilfried Steiner, Guillermo Rodriguez-Navas, and Hans Hansson. 2015. A decomposition approach for SMT-based schedule synthesis for time-triggered networks. In *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*. 1–8. https://doi.org/10.1109/ETFA.2015.7301436

[29] Wei Quan, Jinli Yan, Xuyan Jiang, and Zhigang Sun. 2020. 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)*. 369–376. https://doi.org/10.1109/HPCC-SmartCity-DSS50907.2020.00045

[30] Michael Lander Raagaard and Paul Pop. 2018. Optimization algorithms for the scheduling of IEEE 802.1 Time-Sensitive Networking (TSN).

[31] Mauricio GC Resende and Celso C Ribeiro. 2014. GRASP: Greedy randomized adaptive search procedures. *Search methodologies: introductory tutorials in optimization and decision support techniques* (2014), 287–312.

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

[33] Bassem Sellami, Akram Hakiri, Sadok Ben Yahia, and Pascal Berthou. 2022. Energy-aware task scheduling and offloading using deep reinforcement learning in SDN-enabled IoT network. *Computer Networks* 210 (2022), 108957.

[34] Giorgio Stampa, Marta Arias, David Sánchez-Charles, Victor Muntés-Mulero, and Albert Cabellos. 2017. A deep-reinforcement learning approach for software-defined networking routing optimization. *arXiv preprint arXiv:1709.07080* (2017).

[35] Ammad Ali Syed, Serkan Ayaz, Tim Leinmüller, and Madhu Chandra. 2021. Dynamic Scheduling and Routing for TSN based In-vehicle Networks. In *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*. 1–6. https://doi.org/10.1109/ICCWorkshops50388.2021.9473810

[36] Yujie Tang, Nan Cheng, Wen Wu, Miao Wang, Yanpeng Dai, and Xuemin Shen. 2019. Delay-Minimization Routing for Heterogeneous VANETs With Machine Learning Based Mobility Prediction. *IEEE Transactions on Vehicular Technology* 68, 4 (2019), 3967–3979. https://doi.org/10.1109/TVT.2019.2899627

[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention Is All You Need. arXiv:1706.03762 [cs.CL]

[38] Stefano Vitturi, Claudio Zunino, and Thilo Sauter. 2019. Industrial Communication Systems and Their Future Challenges: Next-Generation Ethernet, IIoT, and 5G. *Proc. IEEE* 107, 6 (2019), 944–961. https://doi.org/10.1109/JPROC.2019.2913443

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

[40] Chuanyu Xue, Tianyu Zhang, Yuanbin Zhou, Mark Nixon, Andrew Loveless, and Song Han. 2024. Real-Time Scheduling for 802.1Qbv Time-Sensitive Networking (TSN): A Systematic Review and Experimental Study. arXiv:2305.16772 [cs.NI]

[41] Jinli Yan, Wei Quan, Xuyan Jiang, and Zhigang Sun. 2020. Injection time planning: Making CQF practical in time-sensitive networking. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 616–625.

[42] Chunmeng Zhong, Hongyu Jia, Hai Wan, and Xibin Zhao. 2021. DRLS: A Deep Reinforcement Learning Based Scheduler for Time-Triggered Ethernet. In *2021 International Conference on Computer Communications and Networks (ICCCN)*. 1–11. https://doi.org/10.1109/ICCCN52240.2021.9522239

A APPENDICES

A.1 Ablation Experiments

A.1.1 Impact of Training Pattern:

Setup: In the training process of each episode, the data sampling method will impact the training performance and convergence speed. The difference between 2 training data sampling methods, i.e., 1. Sampling trajectories of unsuccessfully routed flows and 2. Random sampling is compared by route agent. We train the agent in the 20-node ring-topology network and the flow number is set as 16.

Results: Fig. 20 shows the comparison. Method 1 converges in the 51st episode while Method 2 is still not converged in the 220th episode and the ultimate routing success rate of Method 1 exceeds Method 2 by 75%. The reason is that the feedback of unsuccessfully routed flows contains the knowledge that the agent lacks. Thus, sampling the trajectories of these flows to update the agent parameter helps the agent learn immediately. Instead, Method 2 randomly samples trajectories in each episode, which share the same distribution in all episodes. It leads the agent to learn generally the same knowledge as the training progresses and tends to converge to local optimums.

(a) (b)

Fig. 20. The comparison of data sampling method in training. (a) The loss of route agent training exploiting two data sampling methods. (b) The reward of route agent training exploiting two data sampling methods.

A.1.2 Effect of Curriculum Learning:

Setup: During the training of the time agent, we leverage curriculum learning to improve generalization and speed up convergence. We display its performance and the comparison to the case without curriculum learning. We train the agent in the 20-node ring-topology network and the flow number is set as 100.

Results: Fig. 21 displays the performance comparison of agents with and without curriculum learning. As shown in Fig. 21b, Agent1 taking no curriculum is unstable and its performance keeps fluctuating. Agent 2 only taking curriculum 1 converges to local optimum in episode 1094 and the fluctuation intensity declines compared to Agent1. Agent 3 taking Curriculum 1 and 2 converges in episode 22 and the average reward it gained exceeds Agent 2 by 71.01%. It demonstrates that designing complete curriculums for agents can improve generalization and speed up convergence when training data is fed in a complexity-increasing manner.

A.1.3 Effect of Positional Encoding:

Setup: In the training process of the time agent, the positional encoding will contribute to the convergence speed and model stability. We verify this by comparing the training of two time agents, i.e., one utilizing the positional encoding while the other excludes it. The verification setting is 5-node ring topology and the flow number is 11.

(a) (b)

Fig. 21. The performance of time agent adopting curriculum learning. (a) The training process of the time agent taking Curriculum 1 and 2. The Agent converges in both curriculums. (b) The test of agents taking no curriculum, only curriculum 1, and both curriculum 1 and 2 respectively.

(a) (b)

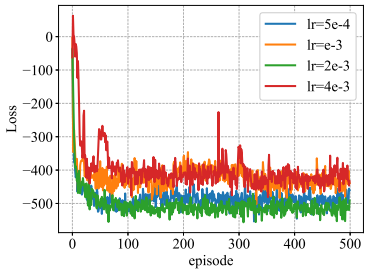
Fig. 22. The (a) loss comparison and (b) reward comparison of training between time agents with and without positional encoding.

Results: Fig. 22 displays the effect of positional encoding. It is the comparison between the training loss and reward of 2 time agents, one with positional encoding while the other excludes it. As Fig. 22a shows, the agent without positional encoding converges to local optimal. In contrast, the agent with positional encoding converges to a smaller loss, indicating its capacity of learning from the environment to generate effective policy. Fig. 22b shows that the agent with positional encoding acquires a higher reward than the one without encoding. Meanwhile, the agent with encoding converges in the 720th episode, while the one without it is still unstable in the 1983rd episode. It suggests that positional encoding contributes to better performance and model stability.

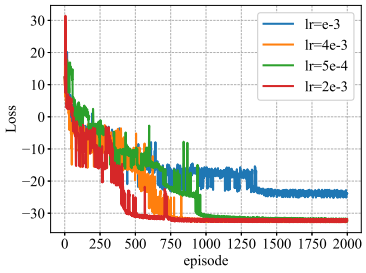
A.1.4 Selection of different model parameters:

Setup: In the training process of the route agent and time agent, the parameters will impact the training performance and convergence speed. We conduct ablation experiments on these parameters including learning rate, reward parameters of route agent ζ and η, and reward parameter of time agent ξ. We train bot route agent and time agent in the 20-node ring-topology network and the flow number is set as 16.

Results: Fig. 23 shows the comparison of training loss convergence of route agent and time agent with different learning rates. For the route agent with a learning rate of 0.002, its training loss is the minimum compared with the training loss with other learning rates, indicating the highest learning capacity at a learning rate of 0.002. For the time agent with a learning rate of 0.002, though the ultimate training loss is the same as the cases with learning rates of



(a) The training loss of the route agent with different learning rates.



(b) The training loss of the time agent with different learning rates.

Fig. 23. Ablation experiments of learning rate with (a) route agent and (b) time agent.

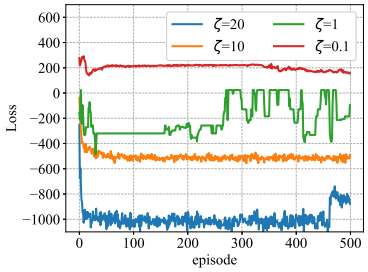


Fig. 24. The training loss of the route agent with different ζ values.

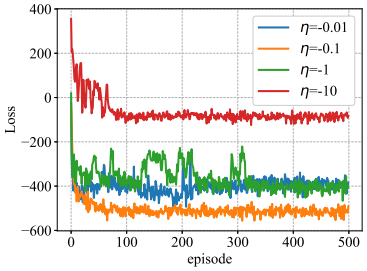


Fig. 25. The training loss of the route agent with different η values.

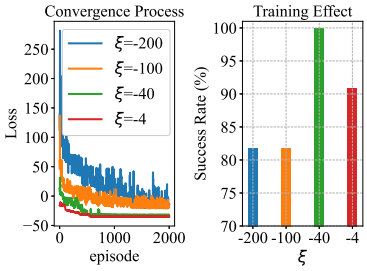


Fig. 26. The training loss and scheduling success rate of the time agent with different ξ values.

0.004 and 0.0005, the convergence speed with a learning rate of 0.002 is the fastest. The time agent with a learning rate of 0.002 is converged in the 713th episode, but with a learning rate of 0.004, it is converged in the 826th episode.

Fig. 24 and Fig. 25 show the comparison of training loss of route agent with different ζ and η values respectively. For all the tested ζ values, only the route agent with ζ=10 is converged. Note that the loss of route agent with ζ =20 reaches the lowest because the value of ζ increases the reward and then reduces the loss values, but the model does not converge. Thus it reflects no model generalization improvement. As a result, 10 is the most suitable value for ζ. For all the tested η values, the route agent with η=-0.1 shows the fastest convergence speed, and it converges at the 32nd episode, while the route agent with η=-10 converges at the 97th episode. Thus -0.1 is the most suitable value for η.

Fig. 26 shows the comparison of training loss convergence scheduling performance of time agent with different ξ values. The time agents all converge to the same value with ξ being -4 and -40. But the performance of the model with ξ=-40 reaches the highest scheduling success rate 100% while the performance of the model with ξ=-4 is 90.9%. Thus -40 is the most suitable value for ξ.

A.2 Scheduling Constraints

Frame Constraint: The offset of any frame has to be greater than or equal to 0. Additionally, the entire transmission window (offset plus frame duration) has to fit within the frame period. The constraint is as follows:

$$\forall f_i \in F,\ \forall [v_a, v_b] \in f_i.Rt,\ \forall f_{i,m}^{[v_a,v_b]} \in F_i^{[v_a,v_b]}:$$
$$(f_{i,m}^{[v_a,v_b]}.\phi \geq 0) \wedge (f_{i,m}^{[v_a,v_b]}.\phi \leq f_{i,m}^{[v_a,v_b]}.period - f_{i,m}^{[v_a,v_b]}.L), \tag{12}$$

where $F_i^{[v_a,v_b]}$ refers to the set of frames of flow f_i on link $[v_a, v_b]$, and $f_i.Rt$ refers to the route of flow f_i.

Flow Transmission Constraint: The propagation of frames of a flow must follow the sequential order along the routed path of the flow.

$$\begin{aligned}
&\forall f_i \in F,\ \forall [v_a, v_x],\ [v_x, v_b] \in f_i.Rt,\\
&\forall f_{i,j}^{[v_a,v_x]} \in F_i^{[v_a,v_x]},\ \forall f_{i,j}^{[v_x,v_b]} \in F_i^{[v_x,v_b]}:\\
&f_{i,j}^{[v_x,v_b]}.\phi \geq f_{i,j}^{[v_a,v_x]}.\phi + f_{i,j}^{[v_a,v_x]}.L + \sigma,
\end{aligned} \tag{13}$$

where σ refers to the time synchronization error.

Deadline Constraint: The deadline constraint specifies that the end-to-end latency cannot exceed the deadline.

$$\begin{aligned}
&\forall f_i \in F,\ \forall j \in [0, \frac{T_{sched}}{f_i.period} - 1]\\
&f_{i,j}^{Last(f_i.Rt)}.\phi + f_{i,j}^{Last(f_i.Rt)}.L - f_{i,j}^{First(f_i.Rt)}.\phi \leq f_i.ddl,
\end{aligned} \tag{14}$$

where $Last(f_i.Rt)$ and $First(f_i.Rt)$ refer to the last link and the first link in the route sequence of flow f_i, respectively.

Link Constraint: Two frames routed through the same physical link in the network can not overlap in the time domain. The constraint is as follows:

$$\begin{aligned}
&\forall [v_a, v_b] \in E,\ \forall F_i^{[v_a,v_b]},\ F_j^{[v_a,v_b]},\ i \neq j,\\
&\forall f_{i,k}^{[v_a,v_b]} \in F_i^{[v_a,v_b]},\ \forall f_{j,l}^{[v_a,v_b]} \in F_j^{[v_a,v_b]},\\
&\forall \alpha \in [0, \frac{T_{sched}}{f_i.period} - 1],\ \forall \beta \in [0, \frac{T_{sched}}{f_j.period} - 1]:\\
&(f_{i,k}^{[v_a,v_b]}.\phi + \alpha \times f_{i,k}^{[v_a,v_b]}.period \geq\\
&f_{j,l}^{[v_a,v_b]}.\phi + \beta \times f_{j,l}^{[v_a,v_b]}.period + f_{j,l}^{[v_a,v_b]}.L) \vee\\
&(f_{j,l}^{[v_a,v_b]}.\phi + \beta \times f_{j,l}^{[v_a,v_b]}.period \geq\\
&f_{i,k}^{[v_a,v_b]}.\phi + \alpha \times f_{i,k}^{[v_a,v_b]}.period + f_{i,k}^{[v_a,v_b]}.L).
\end{aligned} \tag{15}$$

Deterministic Queue Constraint: To guarantee the deterministic arrival order, frames of queue-sharing flows must be scheduled so their arrival times are far apart to avoid interleaving. Thus, the constraint isolates two different frames such that one frame can be transmitted to a shared queue only after the other frame is dispatched from the queue.

$$\begin{aligned}
&\forall [v_a, v_b] \in E,\ \forall F_i^{[v_a,v_b]},\ F_j^{[v_a,v_b]},\ i \neq j,\\
&\forall f_{i,k}^{[v_a,v_b]} \in F_i^{[v_a,v_b]},\ \forall f_{j,l}^{[v_a,v_b]} \in F_j^{[v_a,v_b]},\\
&\forall \alpha \in [0, \frac{T_{sched}}{f_i.period} - 1],\ \forall \beta \in [0, \frac{T_{sched}}{f_j.period} - 1]:\\
&(f_{i,k}^{[v_a,v_b]}.\phi + \alpha \times f_{i,k}^{[v_a,v_b]}.period + \sigma \leq\\
&f_{j,l}^{[v_y,v_a]}.\phi + \beta \times f_{j,l}^{[v_y,v_a]}.period) \vee\\
&(f_{j,l}^{[v_a,v_b]}.\phi + \beta \times f_{j,l}^{[v_a,v_b]}.period + \sigma \leq\\
&f_{i,k}^{[v_x,v_a]}.\phi + \alpha \times f_{i,k}^{[v_x,v_a]}.period).
\end{aligned} \tag{16}$$