Deep Reinforcement Learning aided No-wait Flow Scheduling in Time-Sensitive Networks

1st Xiaolong Wang
Information Department of
Beijing University of Technology
Beijing, China
wxlong124@163.com

2nd Haipeng Yao
State Key Laboratory of
Networking and Switching Technology
Beijing University of Posts
and Telecommunications
Beijing, China
yaohaipeng@bupt.edu.cn

3rd Tianle Mai
State Key Laboratory of
Networking and Switching Technology
Beijing University of Posts
and Telecommunications
Beijing, China
machealmai@gmail.com

4rd Tianzheng Nie
State Key Laboratory of
Networking and Switching Technology
Beijing University of Posts
and Telecommunications
Beijing, China
nietianzheng1998@163.com

5rd Lin Zhu
State Key Laboratory of
Networking and Switching Technology
Beijing University of Posts
and Telecommunications
Beijing, China
2020140241@bupt.cn

6th Yunjie Liu
State Key Laboratory of
Networking and Switching Technology
Beijing University of Posts
and Telecommunications
Beijing, China
liuyj@chinaunicom.cn

2022 IEEE Wireless Communications and Networking Conference (WCNC) | 978-1-6654-4266-4/22/$31.00 ©2022 IEEE | DOI: 10.1109/WCNC51071.2022.9771665

***Abstract*—Emerging latency-sensitive applications (e.g., industrial control, in-vehicle networks) require that the networks guaranteed data delivery with low, bounded latency. To meet this requirement, the IEEE 802.1 Working Group developed the time-sensitive networks (TSN) standard to enable deterministic communication on standard Ethernet. TSN technology is developed to enable deterministic communication using traffic scheduling and shaping technology. However, while the TSN standards define the mechanisms to handle scheduled traffic, it does not specify algorithms to compute fine-grained traffic scheduling policy. Current TSN flow scheduling schemes largely rely on a manual process, requiring knowledge of the traffic pattern and network topology features. Inspired by recent successes in applying reinforcement learning in online control, we propose a deep reinforcement learning aided no-waiting flow scheduling algorithm in TSN. Extensive simulations are performed to verify that our algorithm can find the optimal solution in an acceptable time.**

***Index Terms*—Time-sensitive networks, deep reinforcement learning, no-wait flow scheduling.**

I. Introduction

RECENTLY, emerging latency-sensitive applications (e.g., industrial control [1], in-vehicle networks [2]) pose stringent latency and reliability requirements for the network. For example, in the automotive environment, accurate timing and guaranteed data delivery are critical for driving safety. These emerging applications require the network to provide determinism data transmission (i.e., bounded low latency and zero jitters). To this end, the IEEE 802.1 Working Group has developed the time-sensitive networks (TSN) standard for providing timing accuracy in the sub-microsecond range. TSN technology is developed to enable deterministic communication using precisely synchronized clocks, traffic scheduling, and shaping technology.

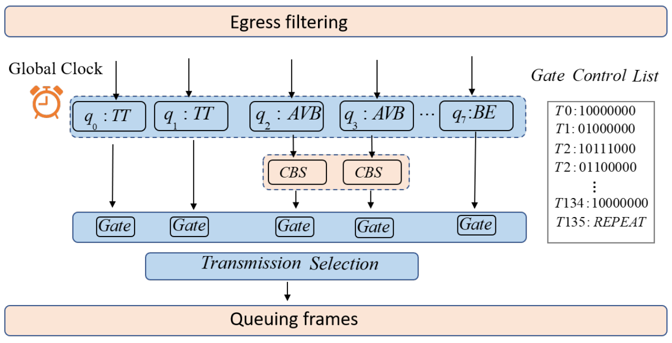


Fig. 1. A Time-aware shaper in Egress port of TSN switch

TSN allows Time-Triggered (TT) traffic with the highest priority which adheres to static schedules, Audio Video Bridging (AVB) traffic, and Best-Effort (BE) traffic with the lowest priority to be transmitted on the same communication network. The TT traffic has a low-latency transmission boundary (e.g., tens of microseconds) and transmission period [3]. The AVB traffic requires a Worst Case end-to-end Delay (WCD) boundary, and BE traffic has no time guarantee requirement. To guarantee the quality of services of different traffic types, a time-aware shaper (TAS) is implemented on each output port of the TSN switch. The TAS separates the traffic with different priorities into different queues based on the priority code points in the VLAN label. The open or closed state of the queue is controlled by the gate control list (GCL) which is calculated by the scheduling algorithm. A value of 1 denotes an open queue, whereas a value of 0 indicates a closed queue. Additionally, flows in the same queue at the output port are sent a in first-in-first-out (FIFO) order. GCL is $T1$: 01000000 at time $T1$, as shown in Fig. 1, implying that only the queue $q1$ is open at $T1$, while the other queues are closed. Only frames in the queue $q1$ can be transmitted at this time. While these traffic shaping algorithms allow multiple traffic classes with varied priority to coexist on the same network, how to schedule different TSN flows is another critical challenge [3]. The traffic scheduling is a well-known NP-hard problem [4]. Currently, the most prevalent TSN traffic scheduling mechanisms employ guard bands to prevent time slices from being consumed by critical traffic, and therefore guaranteed the high priority traffic service. While the guard bands safeguard the high-priority and essential traffic time slices, it suffers from the well-known performance problems (e.g., loss of bandwidth, the minimum procurable time slice length and cycle period). The network operators urgently need to develop effective TSN traffic scheduling schemes to enhance network utilization. Recently, the no-wait scheduling mechanism received quantities of attention from both academics and industrial. These mechanisms can reduce the number of guard bands, so that more bandwidth can be saved and more transmission resources for BE traffic.

Recently, a large and growing body of literature has investigated TSN traffic scheduling. In [5], Dürr *et al.* introduced the no-wait model for TSN scheduling with no queuing delay on switches. Also, they proposed a heuristics algorithm for computing schedules and a schedule compression technique to reduce bandwidth wastage. In [6], Ayman *et al.* proposed an iterated ILP-based scheduling (IIS) approach for the sake of higher scalability. In [7], Voica *et al.* adopted a GCL synthesis approach, which takes AVB traffic into consideration while scheduling TT traffic. Besides, a Time-sensitive Software-defined Network (TSSDN) is introduced to solve the combined problem of routing and scheduling TT traffic by using various ILP formulations in [8]. In addition, considering the dynamic scheduling problem in TSSDN, N. Nayak *et al.* proposed algorithms to tackle incrementally adding TT flows in [9]. The experiment results showed that the proposed algorithms compute schedules of TT flows in sub-seconds.

However, current TSN traffic scheduling schemes rely heavily on manual processing, requiring the design of sophisticated heuristics algorithms, and therefore exhibit poor scalability and robustness to network dynamics. Inspired by recent successes in applying reinforcement learning in many scenarios (e.g., autonomous driving [10], AlphaGo Zero [11]), in this paper, we propose a deep reinforcement learning-based flow scheduling scheme. Also, we introduce the Asynchronous Advantage Actor-Critic (A3C) algorithm in our scheme [12]. Extensive experiments are simulated to evaluate the proposed algorithm.

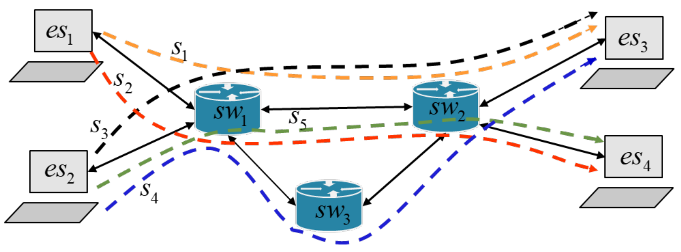


Fig. 2. An illustrative example of four $ES(es_1, es_2, es_3, es_4)$which exchange five streams $(s_1, s_2, s_3, s_4, s_5)$ via TSN network composed of three TSN switches (sw_1, sw_2, sw_3).

II. System Model and Problem Formulation

A. Network Model

Consider a network topology $G \equiv (V, E)$ with a set of network nodes V and a set of links between any two network nodes. The full-duplex link between node $v_a \in V$ and $v_b \in V$ is represented by $(v_a, v_b) \in E$. Moreover, the ordered pair $[v_a, v_b]$ and $[v_b, v_a]$ represent the two directional llinks of the full-duplex link (v_a, v_b) , where the first and second elements represent the start node and end node respectively. Assuming all network nodes have the same transmission rate, eg.,$100M/s$. The network nodes V are composed of a set of TSN switches $Sw = \{sw_1, sw_2, ..., sw_m\}$ and a set of end systems $Es = \{es_1, es_2, .., es_k\}$.

The TSN switches employ IEEE 1588 Precision Time Protocol (PTP) [13] to achieved precisely synchronized clocks. And the DPDK packet processing architecture is employed by the end systems to send and receive data packets. In this way, the host stack is bypassed to avoid uncertain processing delays on the host side, that is, they can follow the calculated global TSN transmission schedule.

In our model, each time-triggered flows can be represented by a six-tuple

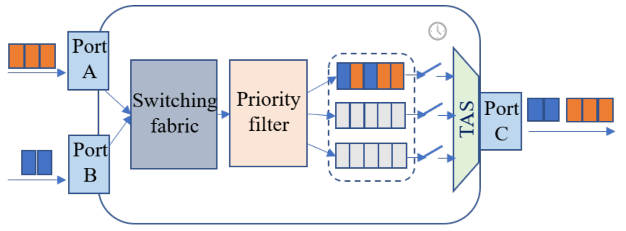
$$f_i = (src_i, dst_i, ddl_i, size_i, cycle_i, path_i) \\ \forall f_i \in F, i \in [0, n-1]. \tag{1}$$

Here F is the set of time-triggered flows, and n is the number of flows. The feature of each flow consists of source address src_i, destination address dst_i, deadline ddl_i, data size $size_i$ which is limited to a maximum frame size of 1500 bytes by Ethernet frame, period $cycle_i$, and path $path_i$ which is calculated by the shortest path algorithm.

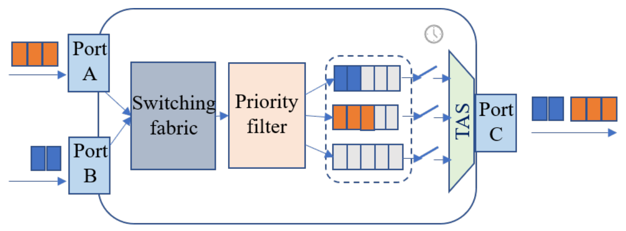
The flows can have different periods, thus the least common multiple of all flow periods denotes the super cycle of F, which can be expressed as:

$$sched_{cycle} = LCM(F \cdot cycle_i) \tag{2}$$

A path $path$ is an ordered sequence, which means starting from the source sending node $v_s \in Es$ and passing through several intermediate nodes $v_1, v_2, ..., v_i \in Sw$ to the destination node $v_d \in Es$. An example is shown in Fig. 2. There are three switches $Sw = \{sw_1, sw_2, sw_3\}$ and four end systems $Es = \{es_1, es_2, es_3, es_4\}$ in the network. The path of the



(a) interleaving flows.



(b) isoloation of flows.

Fig. 3. Flow interleaving and isolation within an egress port of a TSN switch.

stream s_1 from the source es_1 to the destination es_3 can be expressed as:

$$path_1 = \{[es_1, sw_1, sw_2, es_3]\} \tag{3}$$

B. No-wait Scheduling Model

In the TSN network, end-to-end latency consists of the propagation delay on network links, the processing delay of the switches, the queuing delay due to congestion, and the transmission delay. Here, we assume that all switches have the same processing delay d^{proc} and we denote that the propagation delay of all links is d^{prop}. Let $d_{i,j}^{trans}$, $d_{i,j}^{queu}$ represent the transmission delay and the queuing delay of the flow f_i on the switch sw_j.

To minimize end-to-end delay, we eliminate the queuing delay in switches and enable TSN flows to be transmitted without waiting. By adjusting the injection time of the packet at the source, it can avoid conflicts with other flows in the network, thereby eliminating queuing delay. As shown in Fig. 2, flow f_1 transmits packets from end system es_1 to end system es_3 through the switches sw_1 and sw_2. We assume the initial time when the flow f_1 inject packets at es_1 is t_0. If flow f_1 is queued on switches sw_1, to ensure the flow no queuing on the network, we delay the time to inject the packets at source end system es_1, and the new time of injected packets is $t_0 + d_{1,1}^{queu}$.

Due to there is no queuing delay in the network, and the flow path has been determined, thus, the flow transmission schedule on each switch that the flow passes through can be calculated according to the time of injecting packet on the source end system, that is, the gate opening time of the corresponding queue. For example, the flow f_i injects packet at time t and transmits along the path $path_i = \{[es_1, sw_1, sw_2, es_3]\}$ to the destination without queuing delay in network. The transmission time of f_i in switches sw_1 and sw_2 can be calculated by:

$$\begin{aligned} T_{sw_1}^i &= t + d_{i,es_1}^{trans} + d^{prop} + d^{proc} \\ T_{sw_2}^i &= t + d_{i,es_1}^{trans} + d_{i,sw_1}^{trans} + 2 \times (d^{prop} + d^{proc}) \end{aligned} \tag{4}$$

Due to no queuing delay, the total delay of the flow f_i from the source node es_{src} to the destination es_{dst} through n switches can be expressed as:

$$delay_i = (n+1)(d^{prop} + d^{proc} + d^{trans}), \tag{5}$$

The duration $FlowSpan$ from the time the first bit of the first stream is sent at the source end system to the time the last bit of the last stream is processed at the destination end system can be expressed as follows:

$$FlowSpan = T_j^{fin} - T_i^{start}, \tag{6}$$

where T_i^{start} is the time the first stream f_i sent the first bit at the source end system, and T_j^{fin} is the time the last stream f_j completed the last bit at its destination end system.

C. Core Constraints in no-wait scheduling

There are some key constraints that can ensure the effective operation of our no-wait scheduling model.

1) Time Slot Constraint: Symbol $T = \{t_0, t_1, t_2, ..., t_{n-1}\}$ is represent the set of time slot. Consider n time slots in a cycle, the length of each time slot is the sum of the longest transmission delay, propagation delay, and switch delay of the flow, which can be expressed as follows:

$$slot\_len = max(d_i^{trans}) + d^{prop} + d^{proc}. \tag{7}$$

2) Injection Time Constraint: All TT flows must meet their end-to-end maximum delay requirements. The definition of deadline is that all frames of time-triggered flow in each cycle must arrive at the destination host at a specified time. To satisfy this constraint, each flow has the latest injection time at its source host, and must transmit the packet before the latest time, otherwise, it will occur catastrophic consequences. Thus, the latest injection time of a flow can be expressed as:

$$T_{latest}^i = ddl_i - delay_i. \tag{8}$$

In addition, data packets are allowed to be injected only at the beginning of the time slot.

3) FlowSpan Constraint: If the flow of the previous cycle is still in transmission at the beginning of the next cycle, the flow of the next cycle will be queued. To avoid this situation, the constraint is as follows:

$$FlowSpan \leq sched_{cycle}, \tag{9}$$

4) Flow Isolation Constraint: Since there will always be synchronization errors between devices, the order in which a single frame is placed in the scheduling queue during operation may be uncertain [14]. Consider such a situation, as shown in Fig. 3(a), two streams arrive at the switch from distinct sources at the same moment, which has the same egress port and are queued in the same scheduled queue. However, due to the unpredictable arrangement of two streams in the same queue at the egress port, each periodic instance will strengthen jitter during the overall end-to-end transmission. The uncertainty caused by the interleaving of a single frame weakens the time isolation of the stream, because changes in one stream may affect other streams by increasing interference delay and jitter.

To eliminate the frame interleaving issue caused by different streams in the same queue, in this paper, we separate the flows into different queues so that each frame is scheduled deterministically according to the schedule of the relevant timing gate, as shown in Fig. 3(b).

D. Problem Formulation

The guard band is derived from gate driver events which control the opening and closing of the queue gates of the scheduled flow. Reasonable scheduling of time-triggered flows can reduce these events, thereby reducing the number of guard bands, so that more network throughput can be saved and more transmission resources for BE traffic.

Minimizing $FlowSpan$ means that the transmission of time-triggered flows is compressed at the beginning of the schedule, thereby reducing the number of the guard band.

According to our no-wait scheduling model, as long as the transmitting time interval between flows is large enough, the flow will not be queued in the network, but the deadline of the flows can not be satisfied. To avoid the above issue, the injection time of time-triggered flow in its source as early as possible.

Here, the optimization objective can be formulated as follows:

$$min(FlowSpan + \sum T_i), \tag{10}$$

where T_i denotes the injection time of flow f_i in its source.

III. DRL-based scheduling algorithm for TSN

The scheduling problem of TT traffic can be modeled as an Markov Decision Process (MDP). Formally, a MDP can be formalized as a 3-tuple $< A_t, S_t, R >$, where the S_t is the states space, the A_t is the action space, and the R is the immediate rewards.

Specifically, the state's space can be expressed as follows:

$$S = \{s_f \times s_{net}\}, \tag{11}$$

where $s_f = \{s_f^1, s_f^2, ..., s_f^n\}$ is the set of scheduled flow states, and $s_{net} = \{s_{link}^1, s_{link}^2, ..., s_{link}^n\}$ is the set of network link states. The action can be described as follows:

$$A = \{a_t\}, \tag{12}$$

where a_t is the $t-th$ stream, which is scheduled along the path of the s_t. As we want to inject each stream into the network and the completion time as early as possible, the reward can be expressed as:

$$R = \begin{cases} -\infty, \text{if} \quad FlowSpan > schedule\_size \\ -(\delta T_i + \gamma FlowSpan), \text{other} \end{cases} \tag{13}$$

Algorithm 1: The A3C-based scheduling algorithm for TSN

```
Initialize $\theta$, $\theta_v$, $\theta'$, $\theta'_v$ and global counter $T = 0$
repeat
    Reset gradients: $d\theta \leftarrow 0$, $d\theta_v \leftarrow 0$
    Synchronize parameters $\theta' \leftarrow \theta$, $\theta'_v \leftarrow \theta_v$
    Sampling the trajectory $\{s_0, a_0, r_0, ...\}$
    $T \leftarrow T + n$
    Calculate the sampled value:
        $R = \begin{cases} 0, \text{terminal state} \\ v(s_t; \theta'_v), \text{non-terminal state} \end{cases}$
    for $t \in \{n-1, 0\}$ do
        $R \leftarrow r_t + \gamma R$
        Accumulate gradients
        $\theta' : d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_t|s_t; \theta')(R - V(s_t; \theta'_v))$
        Accumulate gradients
        $\theta'_v : d\theta_v \leftarrow d\theta_v + \nabla_{\theta'_v} (R - V(s_t; \theta'_v))^2$
    end for
    Update $\theta$ and $\theta_v$ :
    $\theta \leftarrow d\theta$, $\theta_v \leftarrow d\theta_v$
until $T > T_{max}$
```

,where $0 \leq \delta, \gamma \leq 1$ are the weights, respectively. In this paper, $FlowSpan$ is the main optimization.

At each step, the agent takes an action a_t according to the current state s_t and current policy $\pi_i(a_t|s_t)$. Then, the underlying environment will generate an immediate reward R and transfer the current state s_t to the next state s_{t+1}. If the flow corresponding to the action a_t is correctly scheduled and completed, remove the action a_t from set A. The agent aims to learn a strategy that maximizes long-term future rewards $J_i(\pi_i) = \Sigma r(s, a_1, a_2, ..., a_n)$.

A3C algorithm is an asynchronous reinforcement learning algorithm with excellent performance [12]. Thus, we use $\pi(a|s;\theta)$ to denote the scheduling strategy, which adopts the parameter θ for parameterizations and uses a gradient ascent method for the actor process. The strategy goal gradient can be expressed as:

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta log\pi_\theta(s_t, a_t) A_{\pi_\theta}(s_t, a_t)], \tag{14}$$

where the advantage function $A(s_t, a_t)$ is introduced to evaluate the advantage of choosing action a_t in the state s_t. When it is lower than the average value, the item is negative, otherwise is positive. The advantage function can be expressed as:

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t). \tag{15}$$

We use the approximate value function $V_{\theta_v}(s_t)$ with the parameter θ_v to approximate the TD error that can calculate the policy gradient:

$$\delta_{\theta_v} \approx \sum_{i=0}^{k-1} \gamma_{t+1} V_{\theta_v}(s_{t+k}) - V_{\theta_v}(s_t), \tag{16}$$

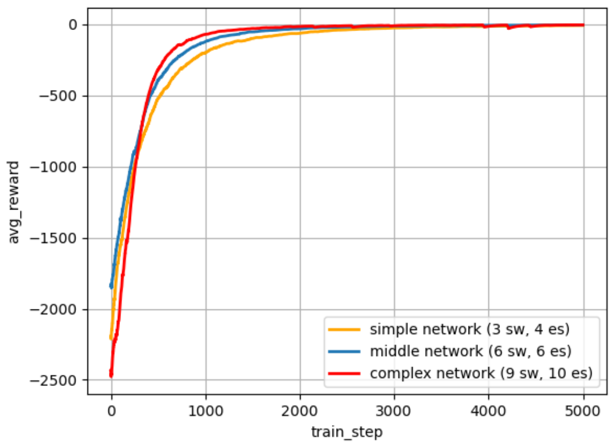


Fig. 4. The convergence of the proposed algorithm.

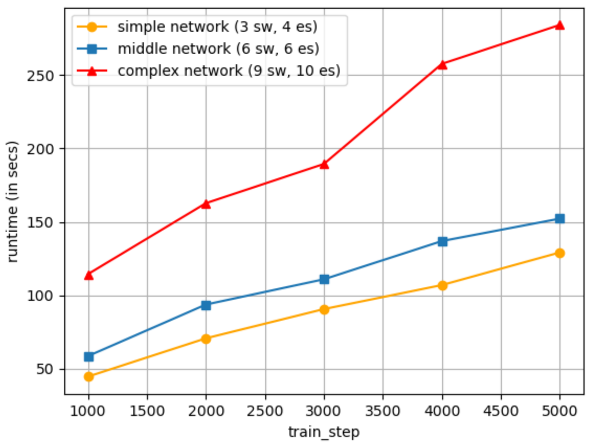


Fig. 6. The running time of training 60 streams with different training steps.

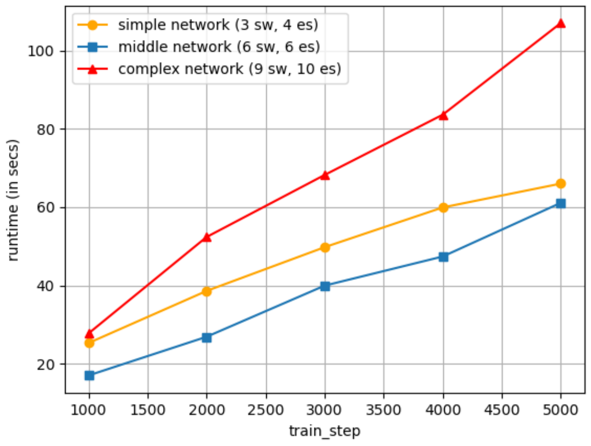


Fig. 5. The running time of training 30 streams with different training steps.

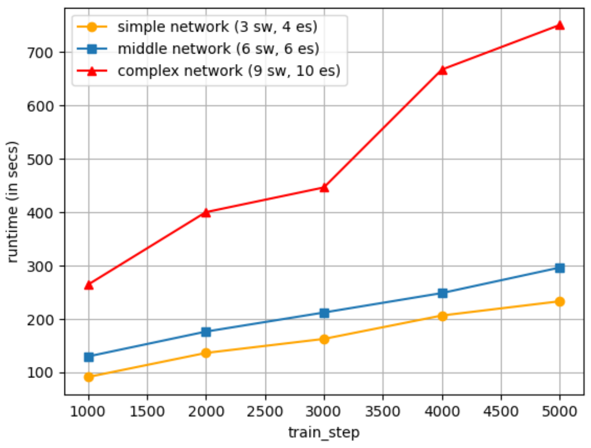


Fig. 7. The running time of training 100 streams with different training steps.

where k can change from one state to another and has an upper bound t_{max}. Furthermore, adding the entropy of the strategy π to the objective function can reduce the premature convergence of the suboptimal deterministic strategy, thereby improving the exploration. The parameter θ can be updated as follows:

$$\theta{=}\theta{+}\alpha\left[\nabla_{\theta'}log\pi\left(s_t,a_t;\theta'\right)\delta_{\theta_v}+\beta\nabla_{\theta'}H\left(\pi\left(s_t;\theta'\right)\right)\right], \quad (17)$$

where H is entropy, α is the learning rate, and the hyperparameter β controls the strength of the entropy regularization term.

The critic-network is described as a value function with a parameter θ_v , and Its focus is to enhance the accuracy of the value function estimation. Then, we use this value function to estimate the strategy from the actor-network. The critic-network receives the state s as input and outputs the value function estimate $V_{\theta_v}(s_t)$. The gradient descent method is employed to update the value function parameters θ_v :

$$\theta_v=\theta_v-\alpha_v\delta_{\theta_v}, \quad (18)$$

where α_v is the learning rate.

The proposed A3C-based TSN flow scheduling algorithm is shown in Algorithm 1.

IV. Evaluation and Results

We evaluated our algorithm's performance in this section. The evaluation mainly focuses on the running time and convergence performance of the TSN scheduling algorithm. In our simulation, the software environment is PyCharm and python3.6, and the hardware environment is Intel Core i5-4590 processor running at 3.30 GHz and 12 GB RAM.

In our experiment, we train the proposed algorithm with 1000-5000 steps in three different complexity network topologies, and set the parameters $\delta=0.7$, $\gamma=0.3$, and the learning rate as $1e-4$.

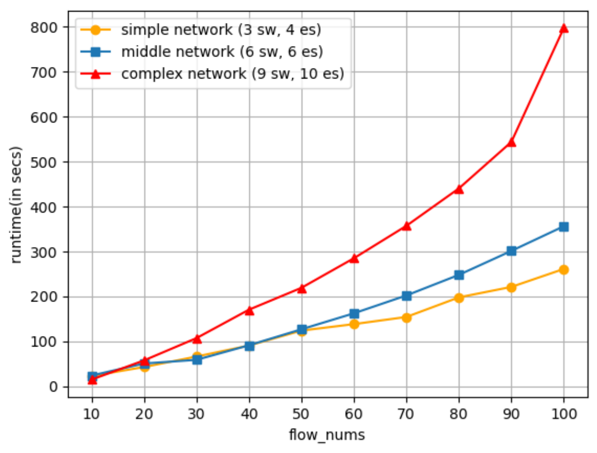


Fig. 8. The running time with different number of flows.

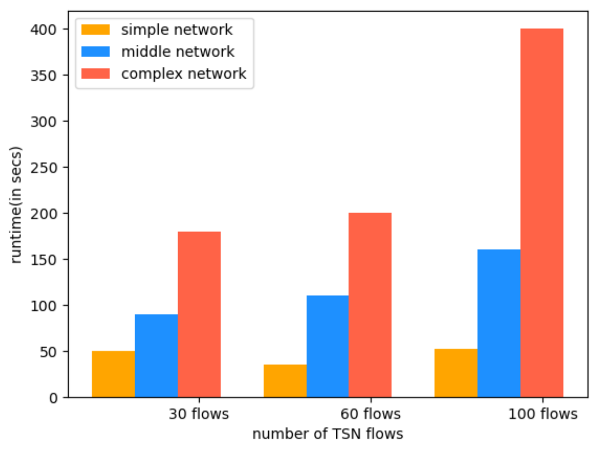


Fig. 9. The running time comparison in different scenarios.

A. Convergence analysis

First, the convergence of our proposed algorithm is analyzed. As shown in Fig. 4, we notice that the convergence of three different network topologies for scheduling 100 TT flows can converge at about 2500 train steps. The higher the complexity of the network, the faster our scheduling algorithm converges. Experimental results show that our proposed algorithm converges and can find the optimal solution.

B. Execution time

Topology complexity and the number of scheduling flows are the two main factors that affect the execution time of the scheduling algorithm. Fig. 5, Fig. 6, and Fig. 7 show the relationship between the number of training steps and running time of our scheduling algorithm in different scenarios. In the case of the same number of scheduled flows, the more complex the network topology, the longer the execution time with the same training steps. Moreover, the execution time of our scheduling algorithms increases as the number of scheduled flows increases with 5000 training steps, as shown in Fig. 8.

From the previous analysis of convergence, we can see that our algorithm can reach convergence when it is trained for about 3000 steps. Fig. 9 presents the evaluation results that the execution time of scheduling different numbers of flows with different topology complexity in the case of convergence. Furthermore, in three different network topologies, the execution time of scheduling 60 time-triggered flows is 35 seconds, 110 seconds, and 200 seconds respectively.

V. Conclusions

In this paper, we use deep reinforcement learning to compute the no-wait schedule for TSN. Simulation results show our proposed algorithm can find the optimal solution and the execution time is within an acceptable range. We currently consider routing in a static environment as a priori knowledge. In future work, we will jointly consider the routing and scheduling of time-sensitive flows in a dynamic environment.

References

[1] M. Wollschlaeger, T. Sauter, and J. Jasperneite, “The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0,” *IEEE industrial electronics magazine*, vol. 11, no. 1, pp. 17–27, 2017.
[2] S. Samii and H. Zinner, “Level 5 by layer 2: Time-sensitive networking for autonomous vehicles,” *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 62–68, 2018.
[3] W. Steiner, S. S. Craciunas, and R. S. Oliver, “Traffic planning for time-sensitive communication,” *IEEE Communications Standards Magazine*, vol. 2, pp. 42–47, 2018.
[4] K. W. Tindell, A. Burns, and A. J. Wellings, “Allocating hard real-time tasks: an np-hard problem made easy,” *Real-Time Systems*, vol. 4, no. 2, pp. 145–165, 1992.
[5] F. Dürr and N. Nayak, “No-wait packet scheduling for ieee time-sensitive networks (tsn),” in *RTNS ’16*, 2016.
[6] A. A. Atallah, G. B. Hamad, and O. A. Mohamed, “Routing and scheduling of time-triggered traffic in time-sensitive networks,” *IEEE Transactions on Industrial Informatics*, vol. 16, pp. 4525–4534, 2020.
[7] V. Gavrilut and P. Pop, “Scheduling in time sensitive networks (tsn) for mixed-criticality industrial applications,” *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*, pp. 1–4, 2018.
[8] N. Nayak, F. Dürr, and K. Rothermel, “Time-sensitive software-defined network (tssdn) for real-time applications,” in *RTNS ’16*, 2016.
[9] N. G. Nayak, F. Dürr, and K. Rothermel, “Incremental flow scheduling and routing in time-sensitive software-defined networks,” *IEEE Transactions on Industrial Informatics*, vol. 14, no. 5, pp. 2066–2075, 2018.
[10] S. Wang, D. Jia, and X. Weng, “Deep reinforcement learning for autonomous driving,” *arXiv preprint arXiv:1811.11329*, 2018.
[11] S. Singh, A. Okun, and A. Jackson, “Artificial intelligence: Learning to play go from scratch,” *Nature*, vol. 550, no. 7676, pp. 336–337, 2017.
[12] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” in *ICML*, 2016.
[13] F. Girela-López, J. López-Jiménez, M. Jiménez-López, R. Rodríguez, E. Ros, and J. Díaz, “Ieee 1588 high accuracy default profile: Applications and challenges,” *IEEE Access*, vol. 8, pp. 45 211–45 220, 2020.
[14] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, “Scheduling Real-Time Communication in IEEE 802.1Qbv Time Sensitive Networks,” pp. 183–192, 2016.