DRLS: A Deep Reinforcement Learning Based Scheduler for Time-Triggered Ethernet

Chunmeng Zhong
School of Software
Tsinghua University
Beijing, China 100084
zcm18@mails.tsinghua.edu.cn

Hongyu Jia
School of Software
Tsinghua University
Beijing, China 100084
jiahy19@mails.tsinghua.edu.cn

Hai Wan
School of Software
Tsinghua University
Beijing, China 100084
wanhai@tsinghua.edu.cn

Xibin Zhao
School of Software
Tsinghua University
Beijing, China 100084
zxb@tsinghua.edu.cn

2021 International Conference on Computer Communications and Networks (ICCCN) | 978-1-6654-1278-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICCCN52240.2021.9522239

***Abstract*—Time-triggered (TT) communication has long been studied in various industrial domains. The most challenging task of TT communication is to find a feasible schedule table. Network changes are inevitable due to the topology dynamics, varying data transmission requirements, etc. Once changes occur, the schedule table needs to be re-calculated in a timely manner. Solver-based methods and heuristic-based methods were proposed to solve this problem. However, solver-based methods employ integer linear programming (ILP) or satisfiability modulo theories (SMT) which have high computational complexity. On the other hand, heuristic-based methods are fast, but they need to be handcrafted based on the application characteristics. Thus, these methods are not general enough to work in complex scenarios especially in large networks.**

In this paper we propose DRLS – Deep Reinforcement Learning based TT Scheduling method. DRLS first trains an application or network specific scheduling agent offline. Then, the agent can be used for online scheduling of TT flows. However, off-the-shelf reinforcement learning techniques cannot handle the TT scheduling problem with typical complexity and scale. DRLS provides novel solutions to this challenge, including three key innovations: new representations for TT network adapted to various topologies, proper deep neural network (DNN) structures to capture network characteristics, and scalable reinforcement learning (RL) models to handle online TT scheduling. Comprehensive experiments have been conducted to compare the performance of DRLS and other methods (heuristics-based methods such as HLS, LS, HLD+LD, LS+LD, and ILP-based method). The results show that DRLS can not only adapt to specific network topologies, but also have better performance: runs much faster than ILP solver-based methods, and schedules about 23.9% more flows than traditional handcrafted heuristic-based methods.

***Index Terms*—time-triggered Ethernet, TTEthernet scheduling, deep reinforcement learning**

I. Introduction

Time-triggered (TT) communication with the end-to-end real-time guarantees has long been studied in the domain of aerospace, industrial control, and automotive electronics [1]. TTEthernet [2] is a deterministic, synchronized, and congestion-free network protocol based on the traditional Ethernet standard. TTEthernet integrates three types of traffic: TT messages, Rate Constrained (RC) messages, and Best-Effort (BE) messages. Traffics in different types offer different QoS (e.g., delay and jitter upper bounds). Usually, TT messages have the highest QoS guarantee and are used to deliver critical control data. TT messages are transmitted according to a pre-calculated schedule table that contains the route information, sending and receiving point-in-times of each TT flow. The schedule table is well designed such that collisions between TT messages and other RC and BE messages are avoided. Finding a suitable schedule table is an important task for TT communication, and this problem has been proved to be NP-complete [3].

The configuration of a TT network is not always static: the network topology may change due to node/link failure [4]; the data transmission requirements may change due to the change of upper applications. Once change happens, we need to renew the TT schedule table accordingly. One possible solution to this situation is that we pre-compute a TT schedule table for each possible network configuration, and switch to the corresponding TT schedule table when change happens. If the number of possible configurations is very large, then this solution is infeasible. Hence, we need to compute the schedule table on-the-fly in a timely manner which we call the dynamic TT scheduling problem.

The methods for synthesis of TT schedule table can be mainly divided into two categories: solver-based methods and heuristic-based methods. Solver-based methods model the network topology, transmission requirements, application specific constraints, application transmission requirements, etc., as a set of constraints (or formulas). Then these constraints are fed into a SMT tool [5] [6] or an ILP tool [7] to get a feasible solution which is then translated into a TT schedule table. Since the problem is an NP problem [3], solver-based methods often have high computational complexity that makes them hard to compute a solution in a short time. On the other hand, heuristic-based methods are faster. But the heuristics are usually handcrafted and need to be designed with good domain knowledge and expertise. Since heuristics are constructed under certain assumptions (for example, tabu search-based methods [8]), they are not generalized enough to have good performance for different configurations. Furthermore, some heuristics (for example, list-base methods [9]) do not scale up well for large networks. Therefore, designing an efficient and high-quality scheduling heuristic is a critical and promising way to solve the dynamic TT scheduling problem.

The TT scheduling problem is essentially a combinational

978-1-6654-1278-0/21/$31.00 ©2021 IEEE

optimization problem. At present, deep reinforcement learning (DRL) has been used to solve scheduling problems [10] [11] [12]. These works use DRL to train a high-quality domain-specific agent and then the agent is used to solve the corresponding scheduling problem. In this paper, we present the DRL-based TTEthernet Scheduler (DRLS), which includes two phases: (1) an offline training phase: agents are trained using specific network topologies or random topologies; (2) an online inference phase: the trained agent is used to compute a TT schedule table for the specific network topology and the flow requirements incrementally.

The design of DRLS faces three challenges: (1) how to encode the time-triggered network status since status representation is a key ingredient to DRL? (2) how to design the deep neural network (DNN) structure of the agent? (3) how to enable incremental (or dynamic) scheduling, since the trained agent is used to schedule TT flows one by one?

The frame of a TT flow is transmitted from the source node to the destination node hop by hop. At each hop, the agent trained by DRLS takes the system state as input and generates an action, which tells the TT frame when and to which neighbor node it should route. DRLS uses directed graphs to represent the network topology and the information of network resources. Two kinds of features are attached to the edges of the graph and are extracted to express the state of the network. The first one is static features which are calculated only once at the beginning of scheduling. The static features contain a reachable matrix which represents the distance of each pair of edges (which is used for frame routing), hyper-period (which is the least common multiple of the periods of all flows), etc.; the other one is dynamic features which is updated every time the frame arrives at a new node. The static features consist of the location of the frame, the allocation of network resources, the remaining transmission time, etc..

The above raw state features are delivered to an embedding neural network to convert them to a uniform format and extract useful information to construct an embedded state feature. Then a policy neural network uses the embedded state feature to calculate the possibilities for each action. Not all actions can be converted to a valid schedule so we propose the concept of control gate to filter invalid actions. The one with the highest probability of the remaining actions is selected as the output. The scheduling of a TT flow stops when the frame arrives at the destination node and the schedule are constructed by the actions sequence generated by the agent. In order to reduce the action space and improve the speed of DRLS, we divide the hyper-period into equally sized time slots. The size of the time slots is long enough for the transmission of the longest frame on any link.

The contributions of the paper are as follows:

1) To the best of our knowledge, we are the first to use DRL to solve the dynamic TT scheduling problem. We propose a DRL-based modeling, training and application method specific to TT flow scheduling.
2) We propose several optimization methods to enhance the feasibility and scheduling ability of DRLS. We use multiple simple actions instead of one complex action to construct a route, which enables DRLS to dynamically schedule TT flows and adapt quickly when the network topology changes. The control gate mechanism is proposed to alleviate the uncertainty of DNN and improve the feasibility of the scheduler.
3) Comprehensive experiments of both industrial and artificial scenarios are conducted to evaluate DRLS. Compared with ILP solver-based methods, the running time is decreased and the average scheduling time per flow is less than 30 milliseconds. DRLS can schedule 23.9% more TT flows than heuristic-based methods on average. DRLS is also verified by simulation using the network simulator OMNeT++.

We start by introducing related work in section II. The system model and overview are discussed in section III and IV, respectively. Section V shows the design of DRLS. We illustrate the evaluate and simulation in section VI and VII. And finally, we draw the conclusion with some discussion in section VIII.

II. Related work

At present, there are a variety of methods related to TT flow scheduling. Generally speaking, scheduling methods can be divided into solver-based and heuristic-based methods. Moreover, solver-based methods can be divided into static and dynamic methods.

Solver-based solution constructs the objective function according to system constraints related to the period and end-to-end delay. Steiner W [5] et al. proposed a static scheduling method based on SMT solver, which leverages a time-triggered communication paradigm to schedule the time window of TT flows offline. There are many other methods based on SMT solvers such as [6] [13] [14]. Generally speaking, these methods solve the scheduling scheme by setting constraints and optimization goals. They can achieve a locally optimal solution by solving the objective function. These static methods usually calculate the TT schedule offline and then distribute the schedule table to each switch or host. Obviously, they cannot adapt to changes in the network topology, and they cannot schedule newly come TT flows either.

Traditional heuristic methods use lots of heuristic rules base on professional experiences to schedule TT flows. On the premise of meeting the deadline of both TT flows and RC flows, Tabu search-based approach [8] can generate an offline schedule of TT flows while minimizing the end-to-end delay of the RC flows. To take Audio-Video-Bridging traffic into account, [15] constructed an initial solution in a greedy manner and iteratively search for a better solution using a modified List Scheduling heuristic. Based on period allocation, Ripoll [16] proposed a schedule generation algorithm that reduced the size of the hyper-period and the size of the schedule table, thus reduced the storage resource requirements for embedded devices. Nasri M [17] [18] proposed a method based on the reconciliation period, which reduces the demand for resources equipment while scheduling TT flows.

Dynamic methods can schedule changing TT flows. [19] proposes a method that exploits the global view of the control plane in SDN to incrementally schedule flows. This method has a relatively low run time because it schedules TT flows on the host devices (terminal devices). Based on a genetic algorithm wrapper feature selection approach, [20] ensembles multiple classifiers and proposes a scheduling method based on machine learning for real-time scheduling scenarios. Although these dynamic scheduling solutions can schedule TT flows incrementally, they make use of some features of specific domain scenarios (such as the control plane of SDN), which is very difficult to get from other TT flow scheduling scenarios. Wang N [7] proposes a dynamic scheduling method based on ILP solver, which can generate an adaptive schedule for the system quickly, but can only be used in the train control network.

As mentioned before, run time is a critical criterion of a scheduler especially in scenarios where TT flows change frequently. Some offline equivalence strategies [21] [22] calculate and store the offline schedule table and then update online which significantly reduces the storage space and scheduling calculation time. However, it is only applicable to scenarios where the initial information is known and contains only a few updates. Nasri [23] proposed a predictable linear-time online preemptive scheduling algorithm, which can solve the problem of sampling delay. However, such algorithms cannot schedule in complex networks with a large number of TT flows.

DRL is a hot technology which has been widely used in network scheduling. In 1994, Boyan J A [24] et al. began to use the Q-Learning method for network scheduling . In recent years, [11] uses graph embedding to extract the feature of nodes in the graph and exploit Q-Learning to train neural networks. [10] uses a graph neural network (GNN) and a policy network to schedule data processing jobs on the distributed computing clusters. [25] tries to use DRL algorithms to solve cellular network traffic scheduling problems. [26] use DRL to solve TDMA link scheduling problem in wireless sensor network considering the energy of devices. It can be seen that many scholars have tried to use neural networks to cope with scheduling problems in recent years. However, to the best of our knowledge, no one has yet proposed a solution using DRL to solve TT flow scheduling problems.

In summary, we can see that the solver-based static method takes a long time to schedule and cannot adapt to changes in the network topology. Traditional heuristic-based dynamic methods have low run time but require the professional knowledge of TT scheduling and are not universal. Therefore, we choose to use DRL technology to solve the TT flow scheduling problem.

III. System Model and Problem Definition

The network topology is modeled as a directed graph $G = (V, E)$, where V is the set of nodes representing the switches in the network. E is the set of directed edges connecting two nodes. If there exists a physical link between v_m and v_n then $(v_m, v_n), (v_n, v_m) \in E$, where the first node in the pair description defines the source node and the second node defines the destination node.

A TT flow is a periodic unicast message from the source node to the destination node. We denote the set of flows in the network as S. A TT flow $s_k \in S$ is defined as a tuple $(src_k, dst_k, len_k, prd_k, delay_k)$, where src_k, dst_k, len_k, prd_k, $delay_k$ are the source node, the destination node, the size in bytes of the message transmitted, the sending period and maximum allowed end-to-end delay of the flow s_k, respectively. Since flows may have different periods, we consider an overall schedule cycle (hyper-period) which is larger than (or equal to) any individual flow period $prd_k \in P$. We denote the hyper-period of all flows as prd_s which is the $least\ common\ multiple$ of the periods of all TT flows.

Similar to [27], the l-th instance of a flow $s_k \in S$ routed through $e_i \in E$ is defined by the frames $f_{k,l}^{e_i} \in F_k^{e_i}$, where $F_k^{e_i} \subset F^{e_i}$ is the set of all frames of flow s_k that are to be scheduled on edge e_i. To schedule a TT flow we need to determine when $f_{k,l}^{e_i}$ travel through e_i. To reduce the complexity of scheduling, one of the most used techniques is the division of the hyper-period into equally sized time slots [28]. We divide 1 millisecond into α time slots donating by T, each of which can transmit the longest instance of Ethernet frame (i.e., an MTU frame). There are $prd_s \times \alpha$ time slots in one hyper-period. $f_{k,l}^{e_i}$ choose one of the time slots to travel through e_i. We denote the status of j-th time slot t_j of e_i in the hyper-period as $Q_{e_i}^{t_j} \in Q_{e_i}$, where $Q_{e_i} \subset Q$ is the status of all time slots of e_i. $Q_{e_i}^{t_j} \in Q_{e_i}$ is an integer representing the number of flows which travel through e_i using the time slots t_j. A time slot can only be occupied by one frame so we have the constraint:

$$\forall e_i \in E, t_j \in T, Q_{e_i}^{t_j} \in \{0, 1\} \tag{1}$$

We denote the schedule of a TT flow from source node src_k to destination node dst_k as $\{(e_0, t_0), (e_1, t_1), \cdots, (e_n, t_n)\}$ where e_0 starts from src_k and e_n ends at dst_k. e_i and e_{i+1} are adjacent edges. The sorts $e_i.src$ and $e_i.dst$ are used for the source node and destination node of the edge e_i. We have the constraint:

$$\begin{aligned} e_0.src &= src_k \\ e_n.dst &= dst_k \\ \forall i \in \{0, 1, 2, \cdots, n-1\}, e_i.dst &= e_{i+1}.src \end{aligned} \tag{2}$$

The end-to-end delay should not exceed the maximum end-to-end delay $delay_k$.

$$t_n - t_0 \leq delay_k \tag{3}$$

As a flow s_k transmits prd_s/prd_k frames in a hyper-period prd_s, it possesses prd_s/prd_k time slots in prd_s.

Similar to [29], the period of TT flows are set to powers of 2, that is to say:

$$\forall prd_k \in P, prd_k \in \{4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048\} \tag{4}$$

The problem of TT scheduling can be formulated as: given the network topology and TT flow requirements, finding valid schedules (routes and corresponding time slots) for all TT flows satisfying all constraints above. We define this model based on assumptions that all network nodes (switches) have distributed synchronization capabilities and real-time storage and forwarding capabilities.

IV. Overview of DRLS

Reinforcement learning (RL) is a general framework which learns to make decisions according to the observation of previous transition data. Typically, an RL consists of four parts, the state space ST, the action space A, the reward R, and policy π. At each timestamp, according to the current state st, the RL $agent$ selects the action a according to π and then a reward $r \in R$ is given by the $Environment$. This process is usually modeled as Markov Decision Process. Deep reinforcement learning is an RL using DNN to implement the policy π which can cope with more complex problems than traditional RL.

As shown in Figure 1, $Agent$ of DRLS is the decision maker deciding what action to take, and $Environment$ modifies the status of the network after a decision is made and gives the rewards to the decision. When the scheduling starts, a state $st \in ST$ is observed containing the information of the network configuration and TT flows. As mentioned above, the schedule of a flow consists of two parts, route and sending time. Then $Agent$ makes a route decision using the DNN and a time slot assignment decision using Low Degree (LD) method and these two decisions constitute the action a. $Environment$ calculates the reward r for $Agent$ regarding to the scheduling target and modifies the network status by performing the action a. This reward is related to the scheduling delay and the completion status of scheduling. The reward is used to continuously improve the performance of the DNN. All processes above are specifically illustrated in section V.

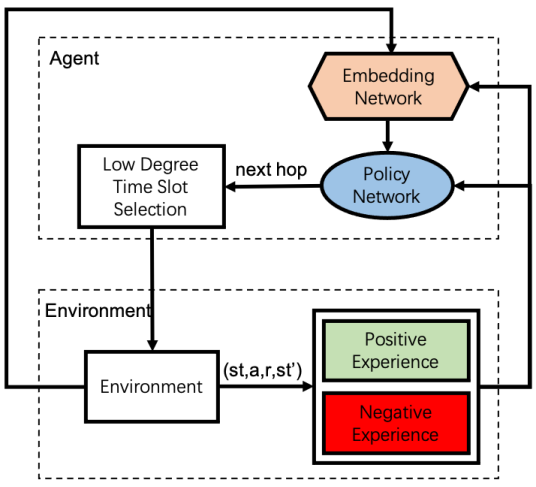


Fig. 1. The structure of DRLS.

We use two neural networks (i.e., embedding neural network and policy neural network) to implement the policy π of DRLS. The embedding network extracts the global topology information of G and the embedding vector of each edge. The policy network is used to determine the next action. The DNN is specifically discussed in section V-D

There are three major challenges in the training and inference of DRLS:

1) **Network Status Representation** The modeling of reinforcement learning needs to fully describe the status of the network and the status of each TT flow scheduled in the network. At the same time, the modeling needs to consider changes of network topology due to network failure. It is a challenge to represent the dynamically changing network resources and topology. This challenge is discussed in section V-A
2) **Dynamic Scheduling** A scheduler cannot know all the TT flows in advance in dynamic scheduling scenarios. The scalability of a scheduler is essential when the network topology becomes large. Many ILP solver-based methods and heuristic methods have poor performance when scheduling for a network with a large number of nodes. How to schedule TT flows dynamically in a large network is a big challenge which is discussed in section V-B
3) **DNN Modeling and Training** DNN is an important part of DRLS which computes route for a flow s_k. How to construct a DNN that is capable of making the appropriate route decision and can finish running in a reasonable time is a challenging problem. The DNN in DRLS is specifically discussed in section V-D and section V-E.

V. Detailed Design of DRLS

In this section, we first specifically describe the components of DRLS mentioned in Section IV then introduce a novel time slot assignment method, the LD method. The solution to the three major challenges is also illustrated in this section. The source of DRLS is available online https://github.com/MengMeng96/DRLS.

A. State Modeling

A system state st represents all the information the $Agent$ needs to generate an action a. We implement st using a vector of real numbers which representing the information of each edge (physical link) in the network. st consists of 6 parts: 1) The distance between this edge and destination node dst_k. A distance of 0 means that the message has reached the destination node dst_k. Generally, the closer the better. 2) Whether this edge is adjacent to the edge of the last action or this edge starts with src_k. The message of the TT flow is currently stored in the destination node of the edge of the last action. Therefore, the next edge needs to be adjacent to the last edge. 3) Whether this edge composes a loop with the edges of the actions before. The schedule of the TT flow should avoid network link loops which waste network resources and leads to a longer delay. We record the nodes been visited and an edge composes a loop if its destination node has been visited.

4) The degree of congestion of this edge, which is expressed as the ratio of the number of free time slots to the total number of time slots. Generally, choose an uncrowded edge can maintain load balancing and avoid bottleneck links 5) This edge has at least one available time slot or not. An edge is invalid if it has no available time slot for the current TT flow. 6)The lowest degree (related to LD method in section V-C) of the time slots of this edge.

Besides st, we also calculate a reachable matrix M which is of size $|E| \times |E| \times D$ where $|E|$ means the number of edges and D is the longest distance between any two edges in the network topology. The distance of two edges is defined as the number of nodes in the shortest route between these two edges. We use $M_{i,j,d}$ represent the (i, j, d)-th value in M and we have:

$$M_{i,j,1} = \begin{cases} 1, if\ e_i.dst = e_j.src \\ 0, if\ e_i.dst \neq e_j.src \end{cases} \tag{5}$$

$$\forall\ d\ \in\ \{2, 3, \cdots, D\},$$
$$M_{i,j,d} = \begin{cases} 1, if \sum_{k=1}^{k=|E|} M_{i,k,1} \times M_{k,j,d-1} > 0 \\ 0, if \sum_{k=1}^{k=|E|} M_{i,k,1} \times M_{k,j,d-1} = 0 \end{cases} \tag{6}$$

$M_{i,j,d} = 1$ means there is a route between e_i and e_j which contains d nodes. st is recalculated every time before making an action decision a while M is only calculated once. st represents the information of each edge and M is related to the network topology.

B. Action Modeling

When using DRL to solve scheduling problems, the action space is huge due to large numbers of choices of routes and sending point-in-time of frames on network nodes [30]. DRLS reduces the size of the action space by dividing the route of a TT flow into a set of adjacent edges. Each action only determines one of the edges instead of the whole route. That is to say, the schedule $\{(e_0, t_0), (e_1, t_1), \cdots, (e_n, t_n)\}$ of a TT flow is derived from a sequence of actions. In other words, each (e_i, t_i) is related to an action. This kind of scheduling paradigm gives DRLS good scalability and solves the dynamic scheduling problem. An action is defined as a pair (e_i, t_j) that represents the edge e_i (the flow need to forward to) and the time slot t_j (assigned to the flow). The edge e_i is chosen by a DNN which is illustrated in section V-D. The time slot is chosen by the LD time slot assignment method in the next subsection.

C. Low Degree Time Slot Assignment Method

Choosing an appropriate time slot t_j will enable edge e_i to carry more TT flows. TT flows with a small period would occupy more time slots than those with large periods and all of these time slots must be available. A time slot t_j is called low degree time slot if it can only carry a TT flow with a large period because a subset of needed time slots has been occupied by other TT flows. We use $G(e_i, t_j, prd_k)$ to represent if the time slot t_j of edge e_i can carry a flow which has a period prd_k, then we have:

$$G(e_i, t_j, prd_k) = \prod_{l=0}^{\frac{prd_s}{prd_k}-1} (1 - Q_{e_i}^{t_{j+l\times prd_k \times \alpha}}) \tag{7}$$

Remember prd_k is the period of flow s_k and prd_s is the hyper-period of all flows. We then define the degree $\rho(e_i, t_j)$ of a time slot as:

$$\rho(e_i, t_j) = \sum_{prd_k \in P} \begin{cases} prd_s/prd_k, if\ G(e_i, t_j, prd_k) = 1 \\ 0, if\ G(e_i, t_j, prd_k) = 0 \end{cases} \tag{8}$$

High degree time slots can carry TT flows with large or small periods but low degree time slots can only carry those with large periods. LD method prefers to choose a valid time slot t_j with a low degree $\rho(e_i, t_j)$ which can enable the network to carry more TT flows. As shown in Figure 2, for a network in which each edge has 16 time slots numbered from 0 to 15. Time slots t_2, t_5, t_6, t_{12}, t_{14} are unavailable. Time slot t_3 can carry a TT flow whose period is 4 as all the time slots it needs, t_3, t_7, t_{11}, t_{15}, are available. Time slot t_0 cannot carry it because time slot t_{12} is unavailable. If we use time slot t_3 to carry a TT flow with period 8, then there will be no valid time slot for TT flows with period 4 in the future. As a result, the LD time slot assignment method prefers to select time slot t_0 to carry a TT flow with period 8 rather than time slot t_3.

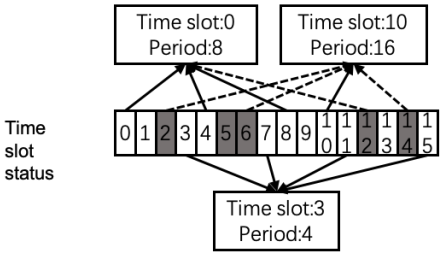


Fig. 2. Example of the LD time slot assignment.

Meanwhile, the scheduler needs to minimize the end-to-end delay of a TT flow s_k. Choosing an early time slot can reduce the delay so the LD method chooses the earliest one (the closest one to the previews action) among the valid and low degree time slots.

D. Deep Neural Network Representation

The DNN uses the information of all the edges as input and outputs the possibility of selecting each edge. The edge e_i of the action (e_i, t_j) is derived from the outcome of the DNN. The DNN, which is learnt from [10], has two parts, the embedding neural network, and the policy neural network.

The embedding network builds an embedding V_b using edge information st and global information M.

$$\begin{aligned} V_b^0 &= st \\ \forall d \in \{1, 2, \cdots, D-1\}, V_b^{d+1} &= \Psi_\theta(V_b^d + V_b^d \times M_d) \end{aligned} \tag{9}$$

Ψ is the embedding network which consists of 4 fully connected layers of size 8, 8, 16 and 8. M_d is a $|E| \times |E|$ matrix representing the reachable information of length d. The policy network χ consists of 4 fully connected layer of size 8, 32, 16 and 8. χ takes V_b^D as input and outputs the possibility V_p of selecting each edge (e.g. in $[0, 1]$).

$$V_p = \chi_\theta(V_b^D) \tag{10}$$

The edge with the highest possibility is chosen. With the reward R (described in the next subsection) of selecting the action and the parameters θ of the DNN, the loss function of the neural network is defined as:

$$Loss_\theta = -\log \chi_\theta \times R \tag{11}$$

The following formula is used to update the parameters θ of Ψ and χ. λ represents the update step size and $\nabla_\theta Loss$ is the gradient of the loss function.

$$\theta = \theta + \lambda \times \nabla_\theta Loss \tag{12}$$

E. Environment Modeling

A rewards R is calculated by the $Environment$ of DRLS when an action decision a is made. Reward R is used to train the DNN of DRLS to update its parameters θ. H_k is used to represent if the flow s_k is successfully scheduled. Successfully scheduling a flow means finding a route from src_k to dst_k and the corresponding time slots whose end-to-end delay is less than the maximum end-to-end delay $delay_k$.

$$H_k = \begin{cases} 1, & if\ s_k\ is\ successfully\ scheduled \\ -1, & if\ s_k\ isn't\ successfully\ scheduled \end{cases} \tag{13}$$

We define the link utilization rate (link usage) as the number of time slots occupied by TT flows divided by the number of all time slots. Remember $Q_{e_i}^{t_j} \in \{0, 1\}$ represents whether the time slot t_j is occupied. The link usage of all links (global usage) U_s and the link usage U_{e_i} of an individual edge e_i is:

$$U_s = \frac{\sum_{Q_{e_i} \subset Q} \sum_{Q_{e_i}^{t_j} \in Q_{e_i}} Q_{e_i}^{t_j}}{\sum_{Q_{e_i} \subset Q} \sum_{Q_{e_i}^{t_j} \in Q_{e_i}} 1} \tag{14}$$

$$U_{e_i} = \frac{\sum_{Q_{e_i}^{t_j} \in Q_{e_i}} Q_{e_i}^{t_j}}{\sum_{Q_{e_i}^{t_j} \in Q_{e_i}} 1} \tag{15}$$

It's worth noting that individual usage and global usage are changing over time and U_s and U_{e_i} are the usage when a flow is successfully scheduled. The link usage reflects the degree of congestion of the link. If the usage of an edge U_{e_i} is larger than the global usage U_s then e_i is likely to be a bottleneck link. If the schedule of s_k is $\{(e_0, t_0), (e_1, t_1), \cdots, (e_n, t_n)\}$, with α, β, γ representing the weight of each part, the rewards $r \in R$ of all actions are defined as:

$$r_n = \alpha H_k + \beta(U_s - U_{e_n}) + \gamma\rho(e_n, t_n) \tag{16}$$

$$\forall\ i \in \{1, \cdots, n-1\}, r_i = r_n \times \frac{i}{n} \tag{17}$$

We cannot tell if a flow is successfully scheduled (H_k) until the last action is chosen. As a result, only the reward of the last action r_n can be calculated using Equation 16 and the rewards of other actions are the decayed value of r_n. Intuitively, earlier action has less influence on the final result.

F. Optimization Methods

Besides the above methods, three important techniques are used in DRLS:

1) Double Channel Experience Replay: When training DRLS, TT flows are randomly generated and DRLS computes schedules for these TT flows one by one. When the scheduling for a flow completes (successfully or incorrectly), rewards are given for all actions taken during the scheduling process. We use $Experience$ to represent the action and its reward and we call an $Experience$ is a positive one if the flow s_k it belongs to is successfully scheduled. As shown in Figure 1, positive and negative experiences are saved in the positive experience pool and negative experience poll, respectively [31] [32] [33]. When the current network scheduling stops, 800 (at most) positive experiences and 200 (at most) negative experiences are randomly selected from the experience pool to train the DNN.

2) Control Gate: DNN would not always generate feasible actions (i.e., actions that can be translated into a valid scheduling table). Hence, in order to generate feasible action decisions, the control gate is proposed to omit edges which is obviously invalid. There are three kinds of edges that are treated as invalid. 1) Non-adjacent edges. As mentioned above, the chosen edge should next to the last edge. 2) Edges that compose an edge loop. A loop wastes the network resource and leads to a large delay. 3) Edges that don't have any available time slot for the current TT flow.

The DNN outputs the possibility of selecting each edge. Then the control gate marks invalid edges. Finally, the agent chooses an edge from valid edges which has the highest possibility. By using the control gate technology, the error caused by the lack of training can be compensated, and the agent is more likely to make valid actions.

3) Failure Recovery: Network nodes (switches) and links may fail, resulting in changes in network topology and network resources that affect subsequent TT flows running in this network. The failed links or nodes should never be used to carry TT flow anymore and the affected flows need to be rescheduled.

Failure recovery is done in four steps. First, information about the affected TT flows are collected, i.e. $(src_k, dst_k, len_k, prd_k, delay_k)$. Then the network resources occupied by the affected flows, such as the time slots of an edge, are released. Third, recalculate the global information

M as the distance between the edges also changes when the network topology changes. The fourth step is to reschedule the affected TT flows using the new global information M.

VI. Evaluation

In this section, we first describe the evaluations settings and baseline methods in section VI-A and VI-B. DRLS is fully evaluated in a variety of scenarios using different types of tests in order to answer the following questions:

1) Compared to other handcrafted heuristics, whether DRLS has better performance in various topologies with a large number of TT flows? (Section VI-C)
2) In link failure scenario, can DRLS schedule more TT flows than other heuristics? (Section VI-D)
3) Does DRLS continuously schedule TT flows at higher link usage than other heuristics? (Section VI-E)
4) As a heuristic method, does DRLS run reasonably faster? (Section VI-F)

A. Evaluation Settings

All experiments were run on an Intel(R) Core(TM) i7-8700 64bit CPU @ 3.20GHz with 16 GB of RAM. It is worth noting that the GPU is not used in the evaluation process.

We evaluate the performance of DRLS on 3 kinds of network topologies: simplified AFDX (Avionics Full-Duplex switched Ethernet) network used on the Airbus A380, ladder network topology used on train communication network, and randomly generated network topologies.

1) **Simplified AFDX Topology** AFDX is a typical real-time avionics network used on the Airbus A380 [34]. We use a simplified version as shown in Figure 3. The nodes in figure represent switches and lines represent full-duplex links.

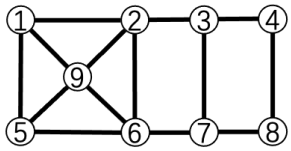


Fig. 3. Simplified AFDX network used in the Airbus A380

2) **Ladder Topology** Ladder topology is a typical topology introduced in IEC 61375-3-4 (Ethernet Consist Network) which is an international standard of train communication network. Figure 4 shows a ladder topology of size 8, in which nodes denote switches and lines denote full-duplex links. The size of the ladder topology can vary such that a small ladder topology could have only 6 switches while a large ladder topology could have 10 switches.
3) **Random Topology** We use an algorithm to generate random topologies. The number of nodes of each topology is randomly selected between $(5, 15)$ with uniform distribution. The possibility of edge connecting any two nodes v_m and v_n is 0.35. In other words, for any two nodes v_m and v_n, a random number between $(0, 1)$ is generated. If the value is larger than 0.35, then an edge connecting v_m and v_n is added.

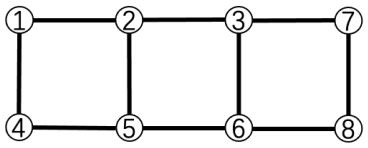


Fig. 4. Ladder network topology

Besides the network topology, TT steams are the other input of the scheduler. The flows used in the training phase and evaluation phase are different and both randomly generated. As mentioned before, a TT flow s_k is defined by a tuple $(src_k, dst_k, len_k, prd_k, delay_k)$. For each random flow, the source node src_k and the destination node dst_k are randomly selected from all nodes in the network. The frame length len_k is an integer randomly selected from $(64, 1518)$. The period prd_k (in millisecond) is selected from set $\{4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048\}$. The maximum delay $delay_k$ is randomly selected between 4 and 256 milliseconds. For convenience, we choose a uniform communication speed for all physical links of 1Gbit/s and we divide a millisecond into 4 time slots which means a frame of maximum size can be transmitted in one time slot.

We trained three agents, one for simplified AFDX topology, one for ladder topologies with different sizes, and one for random topologies. Each agent is trained with the related topology and randomly generated flows which are mentioned above for 20 hours. These agents are evaluated using the same topology as the training phrase but with different flows.

B. Baseline Methods Compared

To evaluate the effectiveness of our model, we compare our model with the following baseline methods. All the methods are well-tuned and we report their best performance.

1) **HLS** Heuristic list scheduler (HLS) [9] finds all the routing possibilities between the source node and destination node of a flow. For each of the possible routes, HLS finds the time slot which leads to the minimum end-to-end delay. The route which has the minimum end-to-end delay among all the routing possibilities is chosen. If the minimum delay exceeds the maximum end-to-end delay $delay_k$, the scheduler fails to schedule this flow.
2) **LS** List scheduler (LS) is almost the same as HLS. The only difference is LS considers the shortest route which containing the minimum number of edges instead of examining all possible routes.
3) **HLS+LD** Heuristic list scheduler with LD time slot assignment method (HLS+LD) is almost the same with HLS but it uses the LD time slot assignment method to select a time slot for each edge. HLS+LD chooses the route with the minimum total LD score among all the routing possibilities. If the end-to-end delay of the

chosen route exceeds the maximum end-to-end delay $delay_k$, then the next best route is chosen. If all routes cannot meet the maximum end-to-end delay $delay_k$, the scheduler fails to schedule this flow.

4) **LS+LD** List scheduler with LD time slot assignment method (LS+LD) is almost the same with HLS+LD. The only difference is LS+LD considers the shortest route instead of examining all possible routes.
5) **ILP** ILP-solver based scheduler (ILP) constructs the objective function according to system constraints related to the period and end-to-end delay. An ILP solver is used to find a feasible solution [7]

C. Incremental Scheduling Scenario

The purpose of this scenario is to verify whether DRLS can schedule more flows than other baseline methods.

In this scenario, all 3 kinds of topologies are used. During one test, the same randomly generated TT flows are fed to all schedulers. If a scheduler fails to schedule a flow then it stops. Finally, we get the maximum number of successfully scheduled flows and the maximum link usage of each scheduler. Note that every scheduler runs at most 3600 seconds.

As shown in Figure 5(a), DRLS schedules more flows than LS+LD (32.7% on average) and LS (183.9% on average) in the simplified AFDX network topology. DRLS, LS+LD, and LS finish running within 60 seconds while HLS and HLS+LD schedule few flows in 3600 seconds. It can be seen that the number of flows which scheduled by DRLS has a larger variance than LS+LD and LS. This is because, on one hand, LS+LD and LS are more stable than DRLS. On the other hand, as shown in Figure 6(a), DRLS has a larger link usage as it schedules more flows which means the network becomes more crowded. A crowded network is more likely to encounter an accident.

Figure 5(b) shows that HLS+LD schedules more flows than DRLS in the ladder topology with 6 nodes. The result is not surprising because HLS+LD considers all possible routes, hence it gets almost the optimal solution. DRLS schedules more flows (20% on average) than LS+LD in the ladder topology with 6 nodes and schedules slightly more flows (0.5% on average) than LS+LD in ladder topology with 8 and 10 nodes. When the size of ladder topology becomes larger (12 and 14 nodes) LS+LD schedules more (2.5% on average) steams than DRLS. This is because the ladder topology is so simple that the shortest route is the best route. As we can see in Figure 6(b), DRLS has larger link usage than LS+LD in all ladder topologies which shows that DRLS prefers to choose a longer route. This policy can avoid bottleneck link in complex network topology but seems do not work for ladder topology.

As shown in Figure 7, DRLS schedules more flows than other schedulers (23.9% than LS+LD, 248.4% than LS on average) in all random topologies. HLS+LD is almost as good as DRLS but it can schedule few flows when the network becomes large (has more than 8 nodes). The result in Figure 8 shows that DRLS, LS+LD, and LS schedule more flows when the network topology becomes large while the link usages have barely budged. This is because a bottleneck link is likely to raise when the network becomes crowded.

D. Link Failure Scenario

The purpose of this scenario is to test whether DRLS and other baseline methods can well handle topology change due to link failure. The test consists of two parts.

1) In the first part, the same randomly generated TT flows are fed into all schedulers. When the number of flows reaches 1000, a random link failure is triggered. The flows routing pass through the link are affected and need to be re-scheduled. We test whether the scheduler can perform the re-scheduling successfully.
2) Then randomly generated TT flows are fed to all schedulers to get the maximum number of successfully scheduled flows after the failure occurs (the same as section VI-C).

DRLS, LS+LD, and LS can all recover from a link failure and Figure 9(a) shows that DRLS can schedule more flows in a changed network topology than LS+LD and LS. Compared with the original simplified AFDX topology (Figure 5(a)), DRLS schedules 5% (on average) fewer flows when a link failure occurs.

E. Saturation Scheduling Scenario

The purpose of this test is to verify whether DRLS can schedule more flows with a relatively high link usage. The following steps are used to maintain the link usage at a high level (i.e., 50% [1]):

1) Randomly TT flows are generated and scheduled, which increase the link usage;
2) When the link usage reaches 50%, 100 randomly selected flows are deleted to lower the link usage;
3) Goto step 1.

The method mentioned in section VI-C is used to get the maximum number of successfully scheduled flows for each scheduler.

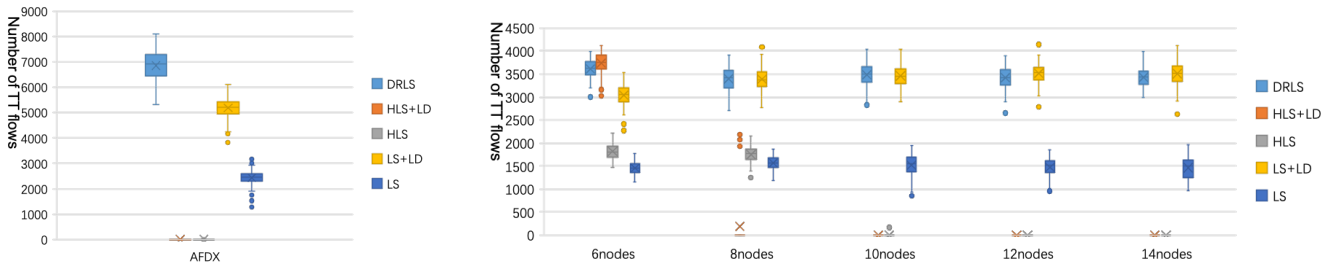
The result in Figure 9(b) shows that DRLS schedules far more flows than LS+LD. Other schedulers are not taken into account as they cannot reach a link usage of 50% (Figure 6(a)). Compared with the result in Figure 5(a), DRLS schedules 204% (on average) more flows in this scenario than in the original simplified AFDX topology.

F. Scheduling Time Comparison

In this test, we compare the running time of different schedulers. During one test, a certain number of flows are randomly generated. The time (in seconds) used to schedule these flows in simplified AFDX topology of all schedulers are compared in Table I. Please note that the ILP scheduling is only executed once and the running time of DRLS, LS+LD, and LS is the average running time of 100 tests.

The result shows that DRLS runs faster than ILP while LS+LD and LS are faster than DRLS. LS+LD and LS are

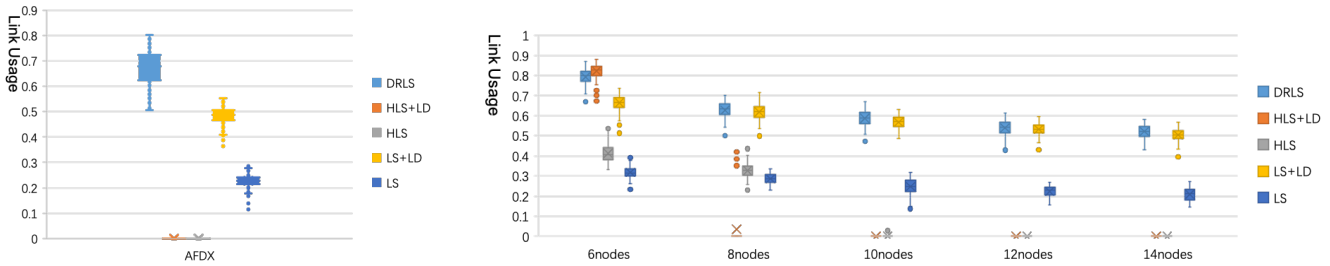
[1]Usually, the link usage of all TT flows is below 30%.



(a) Test results of the simplified AFDX network topology

(b) Test results of ladder network topologies

Fig. 5. The comparison of the number of flows scheduled by different schedulers in simplified AFDX network topology and ladder network topologies of different scale



(a) Test results of the simplified AFDX network topology

(b) Test results of ladder network topologies

Fig. 6. The comparison of the max link usage of scheduling results of different schedulers in simplified AFDX network topology and ladder network topologies of different scale

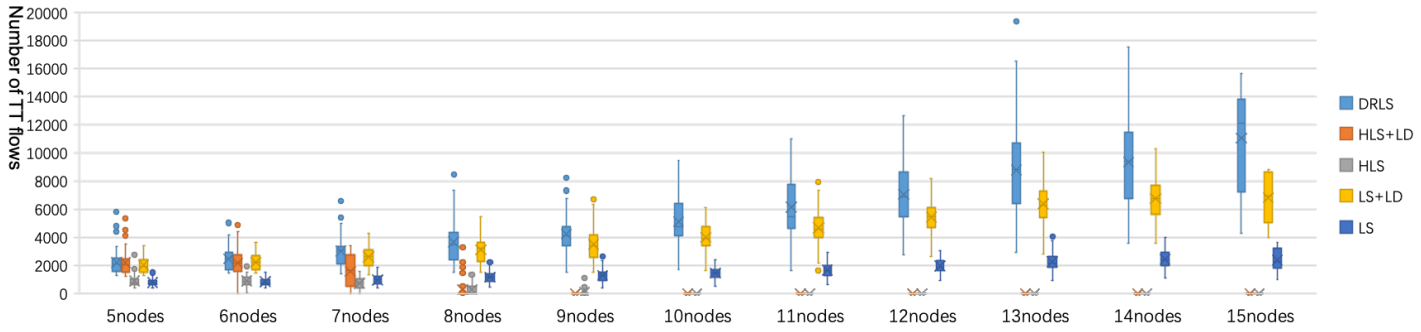


Fig. 7. The comparison of the number of flows scheduled by different schedulers in random network topology

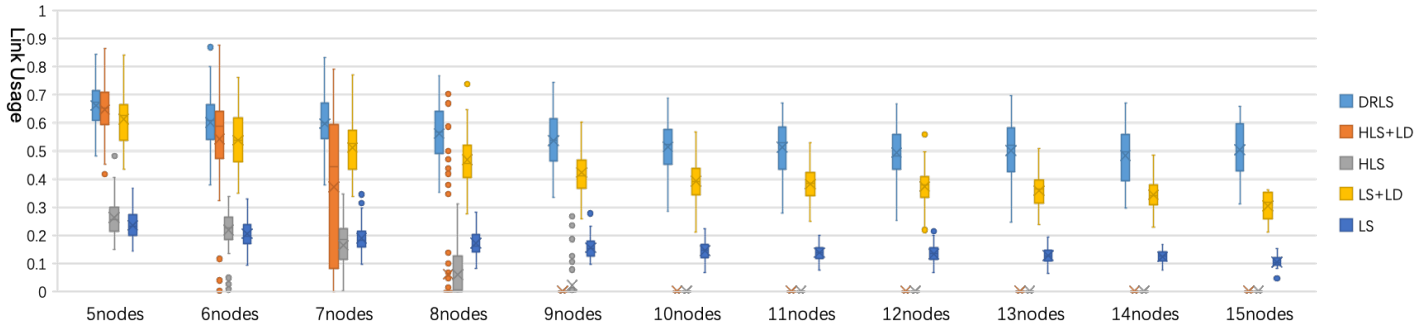
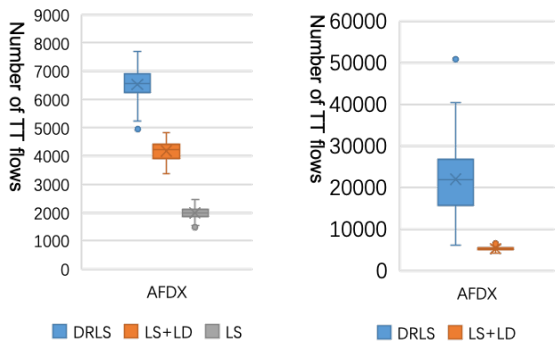


Fig. 8. The comparison of the max link usage of scheduling results of different schedulers in random network topology



(a) Test results of link failure scenario

(b) Test results of saturation scheduling scenario

Fig. 9. The comparison of the number of flows scheduled by different schedulers in simplified AFDX network topology

faster than DRLS because the former finds the shortest route using a Breadth First Search and the latter uses a DNN to find the route for a TT flow.

TABLE I
THE TOTAL RUN TIME (IN SECONDS) OF SCHEDULING A CERTAIN NUMBER OF TT FLOWS

Number of Flows	ILP	DRLS	LS+LD	LS
300	790.674	8.162057	0.175024	0.124243
500	2288.662	13.60343	0.291707	0.207072
700	4095.956	19.0448	0.40839	0.289901
900	7176.896	24.48617	0.525073	0.37273

Table II shows the average time (in seconds) used to schedule one flow. HLS+LD and HLS are much slower than other schedulers. The average time of DRLS, LS, and LS+LD is almost unchanged when the number of flows increases while the average time of ILP is increasing which shows that DRLS, LS, and LS+LD have better scalability than ILP. Since HLS and HLS+LD cannot finish in 3 hours, the scheduling times of 500 and 500+ flows of HLS+LD and HLS are left empty.

TABLE II
THE AVERAGE RUN TIME (IN SECONDS) OF SCHEDULING A CERTAIN NUMBER OF TT FLOWS

Number of Flows	HLS+LD	HLS	ILP	DRLS	LS+LD	LS
300	335.90	296.35	2.635	0.02671	0.00061	0.00040
500	-	-	4.577	0.02713	0.00065	0.00038
700	-	-	5.851	0.02681	0.00056	0.00041
900	-	-	7.974	0.02823	0.00057	0.00043

VII. INTEGRATION WITH NETWORK SIMULATOR OMNET++

In order to evaluate whether DRLS can be used in a real TTEthernet and to configure switches to schedule TT flows, we use the network simulator OMNeT++ simulator version 5.6.2 to verify the schedule calculated by DRLS.

A. Network Settings

We deploy the simplified AFDX network topology which mentioned above. To simulate the bandwidth and frame size mentioned in VI, the delay of each link is set to 250 microseconds. The TT flows transmitted in the network are 10 randomly generated flows and are showed in Table III.

TABLE III
TT FLOWS

No.	source node	destination node	period	max delay	frame length
1	4	2	256	97	380
2	5	3	64	56	115
3	3	7	128	66	171
4	9	4	16	10	945
5	7	8	64	54	451
6	6	1	256	142	553
7	2	4	256	75	259
8	4	6	512	148	1086
9	7	2	32	12	152
10	2	7	128	113	850

B. Translate Scheduling Result to Schedule Table

The scheduling result of DRLS needs to translate to the per-switch schedule table which tells each switch where and when to deliver a frame. The schedule of a flow s_k can be illustrated as $\{(e_0, t_0), (e_1, t_1), \cdots, (e_n, t_n)\}$ which e_i is the edge decision and t_i is the corresponding time slot. The frame of a TT flow is initially located at the source node of e_0 (src_k according to Equation 2) and the destination node of e_n is responsible to deliver it to the related upper application. When the source node of e_i receives the frame, the port to transport it is the one which connected to the destination node of e_i. As the length of time slots is set to 250 microseconds, the sending time of a frame at the source node of e_i is $t_i \times 250us$.

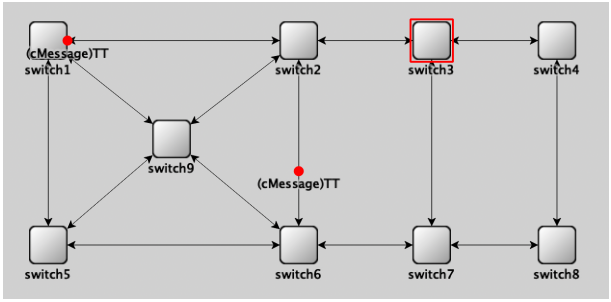


Fig. 10. Simulation of the simplified AFDX network with 10 randomly generated flows

C. Simulation Results and Analysis

The simulation results show that the schedule table calculated by DRLS can be translated to the configuration of a switch and applied to a real world application. Meanwhile,

all flows arrived at their destination on time which shows that DLRS is capable to schedule all TT flows without any collision.

VIII. Conclusion

TTEthernet is essential for industrial control networks with TT communication requirements. One of the most challenging tasks in TTEthernet is the scheduling of TT flows. Most of the existing methods have obvious defects and cannot meet the scheduling requirements of TTEthernet, either perform static scheduling or have poor scalability. A TT flow scheduler with good scalability and dynamic scheduling ability is urgently needed. We propose a DRL based TT flow scheduling method, DRLS, which can dynamically schedule TT flows in a relatively short time. The policy of DRLS agent is implemented as a deep neural network, which can be understood as a well-designed heuristic. When the network topology changes, DRLS can recover in time and reschedule the affected TT flows. Compared with the heuristic rule-based and ILP solver-based scheduling methods, DRLS improves scheduling ability by 23.9% and reduces the computation time to 30 milliseconds per TT flow.

Acknowledgment

This work was supported in part by Key-Area Research and Development Program of Guangdong Province under Grant 2020B010164001, and in part by the NSFC Program under Grant U19A2062 and Grant U1801263.

References

[1] Kopetz H, Grunsteidl G. TTP-A time-triggered protocol for fault-tolerant real-time systems[C]//FTCS-23 The Twenty-Third International Symposium on Fault-Tolerant Computing. IEEE, 1993: 524-533.

[2] Gavriluţ V, Pop P. Traffic class assignment for mixed-criticality frames in TTEthernet[J]. ACM Sigbed Review, 2016, 13(4): 31-36.

[3] Ullman J D. NP-complete scheduling problems[J]. Journal of Computer and System sciences, 1975, 10(3): 384-393.

[4] Pozo F , Rodriguez-Navas G , Hansson H . Schedule Reparability: Enhancing Time-Triggered Network Recovery Upon Link Failures[C]// 2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA). IEEE, 2018.

[5] Steiner W. An evaluation of SMT-based schedule synthesis for time-triggered multi-hop networks[C]//2010 31st IEEE Real-Time Systems Symposium. IEEE, 2010: 375-384.

[6] Craciunas S S, Oliver R S, AG T C. An overview of scheduling mechanisms for time-sensitive networks[J]. Proceedings of the Real-time summer school LÉcole dÉté Temps Réel (ETR), 2017: 1551-3203.

[7] Wang N, Yu Q, Wan H, et al. Adaptive scheduling for multicluster time-triggered train communication networks[J]. IEEE Transactions on Industrial Informatics, 2018, 15(2): 1120-1130.

[8] Tamas-Selicean D, Pop P, Steiner W. Synthesis of communication schedules for TTEthernet-based mixed-criticality systems[C]//Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis. 2012: 473-482.

[9] Pahlevan M , Tabassam N , Obermaisser R . Heuristic list scheduler for time triggered traffic in time sensitive networks[J]. ACM SIGBED Review, 2019, 16(1):15-20.

[10] Mao H, Schwarzkopf M, Venkatakrishnan S B, et al. Learning scheduling algorithms for data processing clusters[M]//Proceedings of the ACM Special Interest Group on Data Communication. 2019: 270-288.

[11] Khalil E, Dai H, Zhang Y, et al. Learning combinatorial optimization algorithms over graphs[C]//Advances in Neural Information Processing Systems. 2017: 6348-6358.

[12] Obara M, Kashiyama T, Sekimoto Y. Deep Reinforcement Learning Approach for Train Rescheduling Utilizing Graph Theory[C]//2018 IEEE International Conference on Big Data (Big Data). IEEE, 2018: 4525-4533.

[13] Craciunas S S, Oliver R S, Chmelík M, et al. Scheduling real-time communication in IEEE 802.1 Qbv time sensitive networks[C]//Proceedings of the 24th International Conference on Real-Time Networks and Systems. 2016: 183-192.

[14] Steiner W, Craciunas S S, Oliver R S. Traffic planning for time-sensitive communication[J]. IEEE Communications Standards Magazine, 2018, 2(2): 42-47.

[15] Gavriluţ V, Zhao L, Raagaard M L, et al. AVB-aware routing and scheduling of time-triggered traffic for TSN[J]. Ieee Access, 2018, 6: 75229-75243.

[16] Ripoll, et al., Period Selection for Minimal Hyperperiod in Periodic Task Systems. Computers IEEE Transactions on. 62(9): p. 1813-1822.

[17] Nasri, M. and G. Fohler. An efficient method for assigning harmonic periods to hard real-time tasks with period ranges. in 2015 27th Euromicro Conference on Real-Time Systems. 2015. IEEE.

[18] Nasri, M., G. Fohler, and M. Kargahi. A framework to construct customized harmonic periods for real-time systems. in 2014 26th Euromicro Conference on Real-Time Systems. 2014. IEEE.

[19] Nayak N G, Dürr F, Rothermel K. Incremental flow scheduling and routing in time-sensitive software-defined networks[J]. IEEE Transactions on Industrial Informatics, 2017, 14(5): 2066-2075.

[20] Shiue Y R, Guh R S, Lee K C. Development of machine learning-based real time scheduling systems: using ensemble based on wrapper feature selection approach[J]. International journal of production research, 2012, 50(20): 5887-5905.

[21] Zhang, S., et al. An offline equivalence scheduling technique for time-triggered ethernet. in Proceedings of the 6th International Conference on Communications and Broadband Networking. 2018. ACM.

[22] Nasri M, Brandenburg B B. Offline equivalence: A non-preemptive scheduling technique for resource-constrained embedded real-time systems (outstanding paper)[C]//2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS). IEEE, 2017: 75-86.

[23] Nasri, M. and M. Kargahi, Precautious-RM: a predictable non-preemptive scheduling algorithm for harmonic tasks. Real-Time Systems, 2014. 50(4): p. 548-584.

[24] Boyan J A, Littman M L. Packet routing in dynamically changing networks: A reinforcement learning approach[C]//Advances in neural information processing systems. 1994: 671-678.

[25] Chinchali S, Hu P, Chu T, et al. Cellular network traffic scheduling with deep reinforcement learning[C]//Thirty-Second AAAI Conference on Artificial Intelligence. 2018.

[26] Huang H, Li J. Distributed Energy-Aware Reliable Routing and TDMA Link Scheduling in Wireless Sensor Networks[C]//2020 29th International Conference on Computer Communications and Networks (ICCCN). IEEE, 2020: 1-10.

[27] R S, Craciunas S S, Steiner W. IEEE 802.1 Qbv Gate Control List Synthesis Using Array Theory Encoding. In 2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)(Porto, Portugal)[J]. 2018.

[28] Mok A K, Wang W. Window-constrained real-time periodic task scheduling[C]//Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001)(Cat. No. 01PR1420). IEEE, 2001: 15-24.

[29] Yu Q, Gu M. Adaptive Group Routing and Scheduling in Multicast Time-Sensitive Networks[J]. IEEE Access, 2020, 8: 37855-37865.

[30] Khadilkar H . A Scalable Reinforcement Learning Algorithm for Scheduling Railway Lines[J]. IEEE Transactions on Intelligent Transportation Systems, 2019, 20(2):727-736.

[31] Mnih V, Kavukcuoglu K, Silver D, et al. Playing atari with deep reinforcement learning[J]. arXiv preprint arXiv:1312.5602, 2013.

[32] Liu R, Zou J. The effects of memory replay in reinforcement learning[C]//2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton). IEEE, 2018: 478-485.

[33] Schaul T , Quan J , Antonoglou I , et al. Prioritized Experience Replay[C]// ICLR. 2016.

[34] oulanger F, Marcadet D, Rayrole M, et al. A time synchronization protocol for A664-P7[C]//2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC). IEEE, 2018: 1-9.