Chinese Journal of Electronics
Vol.32, No.5, Sept. 2023

Scheduling Pattern of Time Triggered Ethernet Based on Reinforcement Learning

HE Feng[1,2], XIONG Li[2], ZHOU Xuan[2], LI Haoruo[2], and XIONG Huagang[2]

(1. *Shenzhen Institute of Beihang University, Shenzhen 518000, China*)

(2. *School of Electronics and Information Engineering, Beihang University, Beijing 100191, China*)

Abstract — Time-triggered Ethernet (TTEthernet or TTE for short) is a deterministic and congestion-free network based on the Ethernet standard. It supports mix-critical real-time applications by providing different message classes. The time-triggered (TT) messages have strict end-to-end delay and accurate jitter requirement, and the rate-constrained (RC) messages have less determinism than TT messages but with bounded end-to-end delay requirement. Traditionally, the scheduling of TT messages makes it free of conflicts for the transmission on physical links, but ignoring RC messages scheduling, so it cannot guarantee the transmission of RC messages with a bounded delay. Therefore, the design of TT schedule becomes the key to TTE network applications within avionics environment. In this paper, we propose an algorithm called RLTS based on reinforcement learning and tree search, to optimize the end-to-end delays of both TT and RC messages. Besides, its computation speed is dozens of times faster than satisfied modularity theory (SMT) with asynchronous method for the calculation of the optimal scheduling table. In the case of a large network with more than 1000 TT and 1000 RC messages, the RLTS method can find a scheduling timetable in 10 seconds, and reduce the worst-case delay of RC messages averagely by 20% compared to the genetic algorithm. Meanwhile, our algorithm has a good generalization performance, in another word, it can quickly adjust itself to satisfy the scheduling with the similar performance as before. By using our method, the scheduling pattern of TTEthernet is further discussed. According to the experimental results, the uniformly distributed slots scheduling pattern, namely the porosity scheduling model which is usually recommended for TTE application, is not always suitable for general situations.

Key words — Time-triggered Ethernet, Schedule pattern, Reinforcement learning, Network calculus.

I. Introduction

The standard Ethernet (IEEE 802.3 2012) is known as an unsuitable solution for real-time and safety-critical message transmission especially in the context of avionics environment [1], [2]. Defined by ARINC 664 P7 standard, Avionics Full DupleX Switched Ethernet (AFDX) adopts the concept of virtual link (VL) to enhance the determinacy of event-triggered (ET) message transmission and can provide a bounded end-to-end latency guarantee, but it still has inherent transmission uncertainty due to the asynchronous arrival of messages from different transmission sources [3], [4]. System-wide clock synchronization established by AS6802 provides the basis of time-triggered (TT) communication in TTEthernet, in which TT messages are sent and forwarded at the predefined time windows specified by the static schedule tables [5], [6]. As planning in advance for scheduling windows of TT messages, there is no conflict for TT messages transmission. But rate-constrained (RC) messages still would encounter interference not only from other RC messages but also from TT messages. How to generate a reasonable scheduling table in TTEthernet is really a hard job because different scheduling windows arrangements have different schedulability for TT messages and different impacts on RC messages.

In fact, only focusing on TT messages arrangement, it is difficult to find an analytic method to get a feasible schedulable result. So the dominant literature adopt the satisfied modularity theory (SMT) to solve this problem [7]. Actually, different TT scheduling tables have great impact on the distribution of RC message end-to-end delays. For example, the centralized scheduling pattern and porosity scheduling pattern are typically two different scheduling patterns which will lead to different scheduling tables and cause huge difference in RC message delays. Generally speaking, porosity scheduling

Manuscript Received Dec. 6, 2021; Accepted May 5, 2022. This work was supported by the Technology Development Fund of Shenzhen (2021Szvup082) and the National Natural Science Foundation of China (62071023).

might have more scheduling chance for RC messages in most cases, namely short worst-case end-to-end delays (WCD). But if the porosities are too even, some RC message might not find any appropriate window to be scheduled out as the intervals between two related conjoint TT windows are too small to fit the RC message frames. Thus, the above discussion leads to one question: how to reduce the WCD of RC messages when generating the static schedule table for TT messages at the same time, and this question is still on the way to be solved since both the scheduling method for TT messages and the WCD analysis method for RC messages are complicated enough and time consuming. For example, SMT method usually needs too much time to get a feasible result, so new method should be found to solve this problem. But up to now few references discussed this question. The main work of this paper is to find a way to minimize the delays of TT message and RC messages simultaneously to get a global better real-time transmission guarantee.

In recent years, deep learning and reinforcement learning have achieved a great success in many fields, such as robot control, video games like Atari, board games Go, and other situations [8]. Deep learning can easily describe high-dimensional complexity due to its multi-level associated networks, and has excellent capability of generalization. Reinforcement learning can solve the task without supervise information like labels. When the unresolved problem is particularly complex to describe, imitation learning is often used to improve the convergence speed [9]. For example, AlphaGo Master begins by training a supervised learning policy network directly according to human expert moves [10]. In the case which is difficult to evaluate rewards, count-based algorithm like Go-Explore policy uses exploration and recording to obtain high-quality action-series, then executes imitation learning on these samples to train the neural network with noise to explore to get better samples [11]. For combinatorial optimization problems such as traveling salesman problem, minimum vertex cover, pointer network, S2V DQN and many other algorithms have been adopted to solve them [12], [13]. Thus, deep learning and reinforcement learning potentially can be used to solve scheduling problems in TTE network.

In this paper, we propose a method based on reinforcement learning to generate the optimal scheduling tables with purpose to decrease the delays of TT messages and RC messages at the same time. We regard choosing offset for each message as the action, and the end-to-end delay of all messages as a reward. Under such an assumption, we can use asynchronous reinforcement learning implemented by imitation learning with exploration to quickly solve the scheduling problem [14]. According to experiments, our method can get an optimal scheduling table with low end-to-end delays of TT and RC messages in a short time, and it can suit the similar network configuration by transfer learning. Besides, some suggested scheduling patterns are found according to our experiments.

The rest of this paper is organized as follows. Section II gives a short survey about different generating methods for TTEthernet. Then, the TTEthernet system model is given in Section III. Next, Section IV gives the detailed philosophy and process about our method and Section V introduces the experiments based on an industrial network case. Finally, some conclusions are given in Section VI.

II. Related Work

The generating methods of scheduling table is a common problem for embedded real-time networks, such as 1553B, TTP and TTE. There are a lot of works on bus scheduling and schedulability analysis of end-to-end delays within embedded networking contexts, such as Dobrin and Fohler [15], Davis and Burns [16], Marau *et al.* [17]. The focus of this paper is the scheduling optimization for TTEthernet. SMT is a very powerful general-purpose constraint solver, a tool that combines decision-making procedures to solve problems in first-order logic related to its domain theory. But SMT also has obvious shortcomings that it will take a long time to solve the actual large-scale topology, and it can only find solutions that meet the requirements but difficult to find the optimal one in them. In most situations, we want to obtain an optimal scheduling result and it is clear that SMT cannot achieve it. Moreover, SMT has to completely recalculate even when there are few changes in configuration. Suethanuwong [18] proposes a scheduling approach of the TT traffic, ignoring RC traffic, which introduces equally distributed available time slots for BE traffic. Stochastic optimization algorithms like tabu search and genetic algorithms can optimize TT schedule tables and take into account RC end-to-end delays during the search space exploration [19], [20]. These methods often require a long calculation time to achieve algorithm convergence. When the network configuration is changed, the whole process needs to be executed from the beginning [21]. Moreover, these algorithms are very sensitive to parameters, the same algorithm with little different parameters may cause completely different convergence.

For search problems, A* and its variants have a wide range of applications, but A* requires some appropriate heuristic functions, and different heuristic functions have different guidance and convergence, so they have different optimization effects in different environments, and the results are usually not ideal [22]. In addition, when the scheduling process is not completed, heuristic functions can only guarantee the delay of current TT messages, but the bound of latency of the RC messages is difficult to estimate with the heuristic function, so usually they are not suitable for the problem of scheduling. On the other hand, search structure such as A* conforms to Markov processes, so models like neural network can be used to replace these heuristic functions [23]. Model-based algorithms can be used to adapt to the distribution of data and quickly find out feasible solutions. At the same time, gradient-based optimization methods such as stochastic gradient descent can greatly improve the optimization speed.

III. System Model

The topology of the network is modeled as a directed graph, $G=(V,E)$, where V is the set of nodes representing the system terminals or switches in the network and E is the set of directed edges connecting two nodes. We denote the link between node v_m and node v_n by $(v_m,v_m)\in E$, where the first node in the pair description defines the sending node and the second node defines the receiving node. Information between the sender and receiver is communicated in the form of TT flow that are periodic TT frames according to AS6802. Since flows may have different periods, we consider an overall schedule cycle which is larger than (or equal to) any individual flow period. We denote the hyper-period of all flows as prd_s which is the least common multiple of the periods of all TT flows.

We denote the set of TT flows by F. And we use a quadruple to represent a flow f_i on a dataflow link $[v_k,v_l]$, i.e., $f[v_k,v_l]=\{f_i.period, f_i^{[v_k,v_l]}.offset, f_i.length, f_i.sequence\}$. The period and length of the TT flow are given based on the application specification. The flow sequence, namely $f_i.sequence$, identifies different TT frames of the same flow f_i in different periods. The departure time of TT flow f_i from v_k to v_l, namely $f_i^{[v_k,v_l]}.offset$, is scheduled by TT schedulers. For the scheduling problem of TT flow, there exist three constrain conditions as follows.

1) No-conflict constraints

When a message is transmitted between a transmitter and a receiver, it must occupy an exclusive port and link for sending and receiving. The time windows for sending different messages cannot overlap with each other. Otherwise, data conflicts may exist in the transmitting messages, resulting in the loss of message frames. The expression is

$$\begin{aligned}&\forall\,[v_k,v_l]\in E,\forall f_i\in F\\&(f_i.offset\geq f_j.offset+f_i.length)\\&\qquad\cup\,(f_k.offset\geq f_i.offset+f_i.length)\end{aligned}\tag{1}$$

2) Path-dependent constraint

Messages are sent step by step between ports. The minimum single-hop delay is determined by the fixed transmission delay of the physical link. Set *hopdelay* as the minimum single-hop delay. The expression is

$$\begin{aligned}&\forall\,[v_k,v_l],[v_i,v_r]\in E,\forall f_i\in F\\&hopdelay\leq f_i^{[v_i,v_r]}.offset-f_i^{[v_k,v_l]}.offset\end{aligned}\tag{2}$$

3) End-to-end delay constraints

The source nodes send flows at the specific time according to schedule tables. For example, the n-th departure time of flow f_i from v_k to v_l is specified by $n\times f_i^{[v_k,v_l]}.period+f_i^{[v_k,v_l]}.offset$. Assuming the first dataflow link and the last dataflow link of f_i are $[v_0,v_1]$ and $[v_n,v_{n+1}]$, respectively, the end-to-end latency of flow f_i is $f_i^{[v_n,v_{n+1}]}.offset-f_i^{[v_0,v_1]}.offset$. And the end-to-end latency should not exceed the maximum end-to-end latency. Set *latency* as the maximum end-to-end latency. The expression is

$$\begin{aligned}&\forall[v_0,v_1],[v_n,v_{n+1}]\in E,\forall f_i\in F\\&latency\geq f_i^{[v_n,v_{n+1}]}.offset-f_i^{[v_0,v_1]}.offset\end{aligned}\tag{3}$$

After finishing scheduling all messages, the scheduled results are transformed into schedule tables stored in devices. As a flow f_i transmits $prd_s/f_i.period$ frames in a hyper-period prd_s, it possesses $prd_s/f_i.period$ time slots in prd_s. In order to make the hyper-period simple, the period of TT flows are set to powers of 2, that is to say: $f_i.period\in\{4,8,16,32,64,128,256\}$.

After the above system modeling, the scheduling problem of TT flow is transformed into finding an effective scheduling scheme under the given network topology and the requirements of end-to-end delay of TT flow, so that all TT flows can meet the constraint requirements. We define this model based on assumptions that all network nodes (switches) have distributed synchronization capabilities and real-time storage and forwarding capabilities.

IV. Method

Our approach is based on reinforcement learning implemented by imitation learning with tree search [24], namely RLTS method. Firstly, in the process of search optimization scheduling table, three different methods of reinforcement learning which are pure reinforcement learning, imitation learning with exploration, and imitation learning with asynchronous methods are applied to speed up the search of solution space. For pure reinforcement learning, using deterministic policy gradient (DGP) algorithm, a preliminary and locally optimal message scheduling table can be obtained. After the DPG algorithm training, we will get a deterministic schedule table which will be optimized by using imitation learning with exploration. The advantage of imitation learning is that it has a quick converge speed, but the disadvantage is that imitation learning might experience catastrophic performance degradation if it encounters an untrained state. As for the asynchronous imitation learning, it actually uses the multi-process mechanism of the computer, and uses multiple CPUs to speed up the reinforcement learning and search, so as to make the calculation faster.

Some basic settings are introduced and defined by using the incremental description, which can change the scheduling problem into a Markov decision process (MDP). Thus we can treat the scheduling timetable before and after each generating operation as the current and the next state. After that, the detailed search and learning algorithm are proposed. The overall process of our method is shown in Fig.1.

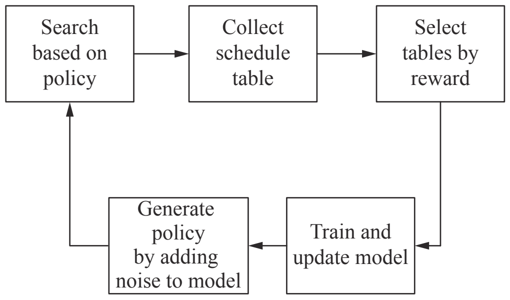

Fig. 1. The overall process to generate scheduling table.

1. Basic setting

From the aspect of scheduling, current messages distributed in scheduling table can be treated as a state, so the state just reflects all needed information without the need of the historical states and actions. Thus, this kind of scheduling process can be transformed into a Markov process and handled with tree search method [25]. Furthermore, we use neural network as the function approximator.

State State is defined as the configuration mode for the current physical link and message in TTE network. Every state must uniquely describe the current physical link and message information. But it cannot be too complex to affect the calculus speed. So we use the current scheduling table and some statistics information of the physical links like the bandwidth utilization of each physical link and the number of unprocessed messages and so on to represent the current state. The scheduling table after executing an action will be treated as the next state.

Action Action is defined as a scheduling offset of a message in the current physical links. Generally speaking, the scheduling offset of each message cannot be larger than its period. Otherwise, the considering message cannot achieve its deadline assurance. Besides, the offsets could be potentially assigned with any possible value as long as it is smaller than the corresponding deadline, which may lead to a continuous distribution for the offsets. This kind of selection will directly result in too large search space and too slow convergence speed for the generating algorithm. In order to balance the speed and accuracy, we use logarithm function to act on the output of neural network to generate offsets, here namely the actions, thereby bringing new operation for reinforcement learning.

Reward Reward is defined as a value related to the delays of TT and RC messages. Formula (4) as below shows an example that defines reward. TT message delays can be obtained from the scheduling table directly, and RC message delays are calculated by network calculus or some other methods. During the search process, we do not give any reward, but perform our constraints like contention free, bound switch memory and so on. After search process completed, we can give a reward calculated to TT and RC messages. In order to make sure the gradient of the training in reinforcement learning is unbiased, we adopt the whole state-action-reward triplet tuple series (Monte Carlo) to execute training process.

$$reward = -(ttdelay + rcdelay) \tag{4}$$

2. Search framework

In the search framework of our proposed method, we use the combination of depth-first and optimal-first search strategies and exclude the branches that obviously cannot find results to speed up the search efficiency. Algorithm 1 shows the basic search framework. In order to speed up the traversal search and obtain feasible solutions, we can traverse the search messages in a specific order like ascending order of the ratio of message length to period or descending order of the message period.

Algorithm 1 Tree search framework of schedule

```
1: Input: the maximal attempts times: $t_{\max}$, the number
   of messages: $x$.
2: Output: a feasible schedule table.
3: Initialize:
4: Initial state: $state$;
5: Policy to choose the first hop offset: Choose($state$);
6: Policy to select each hop offset: Search($offset_1$);
7: Policy to backtrack: Backtracking($offsets$);
```

```
8:  Empty scheduling table: $Schedule$;
9:  for message in messages
10:     get state from environment;
11:       for $t$ in range($t_{\max}$)
12:         $offset_1$ = Choose ($state$);
13:         $offsets$ = Search ($offset_1$);
14:       if $t == t_{\max}$
15:         Backtracking($offsets$);
16:       else
17:       $Schedule = Schedule + offsets$;
18:       end for
19:   end for
```

Without support of reinforcement learning, the function **Choose**(*state*) uses a manually designed heuristic function or a completely Random **Search**(RS) method, i.e. A* to find the shortest path. When using the reinforcement learning, the **Choose**(*state*) function is a multi-layer perception (MLP) neural network, the input is current state and the output is an action added noise. The function **Backtracking**(*offsets*) is used to recursively trace back to the position of the last layer to search in other way according to the hyper-parameters. The function **Search**($offset_1$) is used to complete the search process for the entire set of messages in the following nodes according to their transmission paths. The function **Search**($offset_1$) is given in Algorithm 2.

Algorithm 2 Search($offset_1$)

```
1:  Input: first hop offset: $offset_1$.
2:  Output: offset series for this message: $offsets$.
3:  Initialize:
4:  The maximal attempts times $back_{\max}$;
5:  The basic technical delay for TT messages: lower;
6:  The deadline of TT messages: $upper$;
7:  The balancing parameter: $a$;
8:  RandomChoice($lower, upper$);
9:  Policy to backtrack:Backtracking ($offsets$);
10: Delay of this TT message: $delay$;
11: Delay of this message in this hop: $delay_{\text{tmp}}$;
12: Offset list: $offsets$=[ ];
13: while the current hop is not the last hop
14:     $back = 0, offset = offset_1$;
15:     for back in range($back_{\max}$)
16:       get $upper_i$ and $lower_i$ by using formula (5);
17:       $diff = a * (upper - lower) * back/back_{\max}$;
18:       $higher = lower + diff$;
19:       $delay$ =RandomChoice ($lower, higher$);
20:       if $delay < higher$
21:        break
22:       $offsets$.append($offset + delay_{\text{tmp}}$);
23:       current hop = the next hop;
24:     end for
25:   if $offsets$ is feasible
26:     Output $offsets$;
27:   else
28:     Backtracking($offsets$);
29: end while
```

Actually, **RandomChoice**(*lower*, *higher*) can randomly select values according to typical distribution methods like normal or uniform distribution. When the number of TT messages is small and the length of messages are short, we can quickly obtain a feasible offset within the scope between the upper bound and lower bound, namely a small delay in the current forwarding node. But if the number of TT messages is big enough, for example, more than 1000 messages, piecewise formula (5) is used to widen the search space to increase the likelihood to get a feasible solution. In (5), new $upper_i$ and $lower_i$ can be calculated according to the current back tracing times back, and it will be compared with the predesigned back tracing interval defined by $back_{n-1}$ and $back_n$. Parameters w_{nu}, w_{nl}, b_{nu} and b_{nl} are used to balance the selecting interval and need to be specified before search operation [26].

$$\begin{cases} upper_1 = w_{1u} \times offset + b_{1u} : back_0 \leq back \leq back_1 \\ lower_1 = w_{1l} \times offset + b_{1l} : back_0 \leq back \leq back_1 \\ \quad\quad \vdots \\ upper_n = w_{nu} \times offset + b_{nu} : back_{n-1} \leq back \leq back_n \\ upper_n = w_{nu} \times offset + b_{nu} : back_{n-1} \leq back \leq back_n \end{cases} \tag{5}$$

3. Pure reinforcement learning

Reinforcement learning adopts trial-and-error and iterative approach to achieve the best reward according to the interaction with the environment and finally obtains the optimal response strategy. It does not require the complete environment dynamic model and not need the manually designed heuristic function. So it is quite suitable to solve the decision problem especially in complex situation.

Reinforcement learning can use gradient-based optimization algorithms to make converge much faster, and use neural network as the function approximator to make the model generalizable. Q-learning is widely used in reinforcement learning, but it cannot solve problems with large-scale action space or continuous action space, while policy gradient does. So we can use stochastic policy gradient (SPG) and DPG to train neural network [27].

In the process of DPG, we define Q as the estimation of the cumulative reward. The optimization target of our method is to maximize Q. The loss function includes two parts. The first part represents the estimation accuracy of Q: $(Q-reward)^2$ in (6), and the second part stands for maximizing Q: $(-Q)$ in (6). In this way, we can train the neural network by minimizing the loss function. In the process of SPG, there is no need to estimate Q, and the only optimization target is to maximize cumulative reward by minimizing the inverse of the loss function as (7). $p(a;\theta)$ means the probability of choosing action a when the parameter of policy network is θ. The pseudo code for the both algorithms are shown in Algorithms 3 and 4. Because the action space in the problem is discrete, we use the DPG algorithm to obtain a trained neural network.

$$loss = (Q - reward)^2 + (-Q) \tag{6}$$

$$loss = -\sum p(a;\theta) \times reward \tag{7}$$

Algorithm 3 DPG Algorithm

```
1: Input: current state.
2: Output: trained neural network.
3: Initialize: policy neural network $p$;
4: while not converge
5:    run Algorithm 1 using $p$ with random noise;
6:    receive schedule table as new samples;
7:    update $p$ network parameters by minimizing the loss
      defined in formula (6).
```

Algorithm 4 SPG Algorithm

```
1: Input: current state.
2: Output: trained neural network.
3: Initialize: policy neural network $p$;
4: while not converge
5:    run Algorithm 1 using $p$;
6:    receive schedule table as new samples;
7:    update $p$ network parameters by minimizing the loss
      defined in formula (7).
```

4. Imitation learning with exploration

After the DPG algorithm training, we will get a deterministic schedule table which is supported to be the optimal schedule table (possibly suboptimal), and all the subsequent schedule table should always coincide with this optimal solution. But its convergence speed highly depends on the specific parameters and the corresponding search space. This may cause our algorithm unstable, sometime it will take a long time to complete training process. The advantage of imitation learning is that it has a quick converge speed, but the disadvantage is that imitation learning might experience catastrophic performance degradation if it encounters an untrained state. For a specified avionics networking case, we have all the information about the transmission requirements and the detailed messages parameters as all the characteristics of the messages are defined according to avionics function requirements, such as the deadline, frame length, period. Therefore, imitation learning is used to replace DPG in case of large-scale messages transmission scenarios. In these networking cases, random noise is added into the training algorithm to get better exploration (just like adding noise to DPG), and dataset aggregation (Dagger) is used to execute imitation learning. Since the optimization object is decreasing the delay of the schedule table, there is no need to artificially provide labels for the new generated schedule table samples. After a new sample is obtained, its delay can be used as the label directly [28].

These schedule tables are generated by our algorithm with exploration. In other words, they are produced by neighbor search, so the output of policy is also close to the adjacent samples. In this situation, the similar action (*offsets*) sequences have different but similar labels (*delays*). Thus, neighborhood search provides the generalization ability, and we can also find whether there exists a better schedule table at the same time. The experiments in Section IV also illustrate this philosophy, so we can figure out better schedule table around our sample schedule tables.

Since we adopt the logarithm function to calculate the action value, the contribution of different label to the loss function is also different. Without such logarithm operation the loss value will be too large to cause the oscillation of our model or the explosion of the gradient. On the other side, as the absolute value difference of action is too large, the error of the smaller value of the action might hardly have contribute to the loss function, which makes it difficult to train.

$$loss = (out - action)^2 \times action \tag{8}$$

In formula (8), *out* is the output value of the neural network, and *action* is the logarithmic value of the offset with the maximal reward until now. Noise is defined as a normal or uniform distribution. In imitation learning process, we use formula (8) to train the neural network. The implement of imitation learning with exploration is shown in Algorithm 5.

Algorithm 5 Imitation learning with Dagger

```
1: Input: empty scheduling table Schedule.
2: Output: trained neural network.
3: Initialize:
4: Count $T$, max count $T_{\max}$;
5: Policy network to choose offsets $p$;
6: Sample set of scheduling tables $D$;
7: Noise like normal and uniform distributions;
8: while $T < T_{\max}$
9:    Run Algorithm1 with $p$ with noise;
10:   Gain new sample set $D_{\text{new}}$;
11:   Aggregate $D = D \cup D_{\text{new}}$;
12:   Train $p$ from sampled data in $D$ using formula (8);
13:   $T = T + 1$.
```

We run imitation learning in sample set with the largest reward for the schedule tables by exploring within a neighborhood interval. The selection of the neighborhood obeys a manually designed distribution that the closer neighbor holds a larger probability and the farther neighbor holds a lower probability. Finally, remaining a small probability to perform a completely random search can ensure our algorithm jump out of the current neighbor to have chance to find a more feasible solution in a big scope. Under such conditions, our algorithm can find out the optimal schedule table if the iterations times are adequate.

During training process, we retain some deviation while fitting in each epoch. Considering all the samples are generated by the interaction between the algorithm and the environment, there must exist bias. We can exploit the deviation to expand the exploration interval. Complete fitting will lead to premature converge, and results in insufficient exploration. It is more likely to accumulate bias to cause worse results. When there are some deviation between the current result and the fitting target, we can roughly guarantee that the current solution is in the neighborhood of the target. In this way, we can search in a larger range, and the overall direction is always tending to the final target while exploring and optimizing.

If the training is adequate, theoretically DPG and imitation learning with random exploration have the same convergence accuracy. SPG can converge to multiple preferred paths and maintain the maximum probability of the optimal path, but it might easily fall into sub-optimal search region. Therefore, we use imitation learning here to accelerate convergence rather than DPG. In order to enhance the exploration efficiency, formula (5) is used in exploration process, and the output of the neural network can obey distributions such as uniform or normal methods. The range of the distribution will be adjusted according to the times of visit.

5. Imitation learning with asynchronous methods

As in Ape-X DQN, we decompose the standard deep reinforcement learning algorithm into two parts which run concurrently without high-level synchronization [29]. The first part consists of stepping through an environment, evaluating a policy implemented as a deep neural network, and storing the scheduling table. The second part consists of sampling batches of data to train and update the policy parameters. The corresponding pseudo code is shown in Algorithm 6.

Algorithm 6 Asynchronous imitation learning with Dagger

```
1:  **Input:** empty scheduling table Schedule.
2:  **Output:** trained neural network.
3:  **Initialize:**
4:  Max count: $T_{\max}$;
5:  Number of actors: $n$;
6:  $n$ policy networks to choose their $offsets$ called Actor: $p_w$;
7:  One policy network for training called Learner: $p_m$;
8:  Sample set of scheduling tables $D$;
9:  Normal and uniform distributions: $noise$;
10: One multiprocessing queue: $queue$;
11: Asynchronous apply **for** $i$ **in** $n$
12:    **for** $T$ **in** range($T_{\max}$)
13:       Actor runs Algorithm 1($p_w$) with $noise$;
14:       Push schedule tables in queue;
15:    **end for**
16:  **end for**
17:  **while not all** $actors$ complete
18:    Asynchronously get new samples from $queue$;
19:    Put new samples in sample set $D_{\text{new}}$;
20:    Aggregate $D = D \cup D_{\text{new}}$;
21:    Select data from $D$;
22:    Learner trains $p_m$ according to data in $D$ using formula (8);
23:    Set $p_w$ parameters as $p_m$ parameters;
24:  **end while**
```

The updated network parameters are periodically communicated to the actors from the learner. Different from Ape-X DQN, all the selected training samples are complete schedule tables. As for the schedule problem, we do not know whether the action has long-term reward unless the schedule process is complete. So we use a whole sample to guarantee its rewards unbiased, and use rewards to sort the schedule tables and choose the best sample to train our model. Since actors and learner are parallel, large number of samples for training has no obvious effect to the speed of searching for the schedule tables. Furthermore, if more actors perform the searching process, the probability to find better results would be larger.

V. Experiment

In this part, an industrial networking case is used to illuminate our proposed generating method for scheduling table. The topology is shown in Fig.2 with the similar access switches, backbone switches and connected physical links in A380. The link rates for all physical links are the same as 100 Mbit/s and the frame lengths are set within the scope of [64 bytes, 1518 bytes]. According to link rates and frame lengths, the basic transmission time of each frame is from 6.72 μs to 123.04 μs, here we use the range from 6 μs to 125 μs for simplicity. Followed by ARINC 664 P7 protocol in which the bandwidth allocation gap for virtual link is set from [1 ms, 128 ms], the periods of messages are assigned from the same time interval set. Without loss of generality, the periods and lengths of messages are all considered as integers and the periods obey the power rule of 2 from 1 ms to 128 ms. On the whole, 1050 TT messages and 1932 RC messages are tested in most of our experiments, and the transmission route is specified by load-balancing routing strategy mixed up with shortest path strategy. For the convenience of calculation and optimization, we use the scheduling wait time of TT messages after subtracting the technical delay from the end-to-end delay as the goal of TT messages, and RC delay is the mean of the upper bound of end-to-end latency computed by network calculus in preemption integration policy for all virtual links [30].

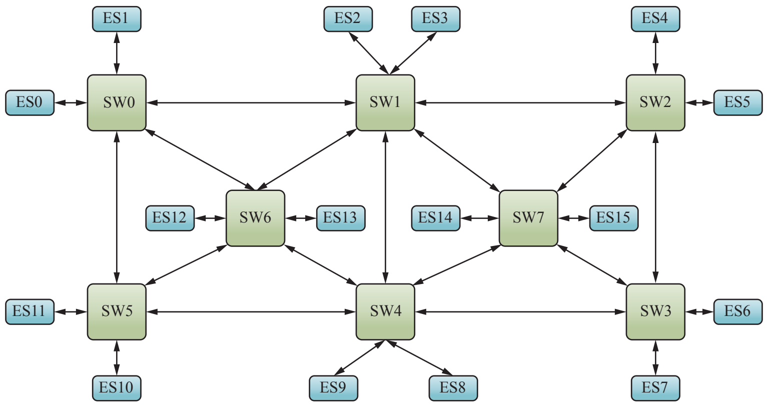

Fig. 2. Networking topology for the four experiments.

We design four experiments to show the availability of our algorithm with different number of messages, different length of messages. Also we will show the generalization of our model.

Hyper-parameters

In order to implement the imitation learning, we use 3-layer fully-connected network which is shown in Fig.3 and each layer contains a hidden layer with size as 256, also ELU is used as the activation function both for the actor and the learner in each layer. The training process adopts the Adam optimizer with learning rate as 0.0005 [30]–[33].

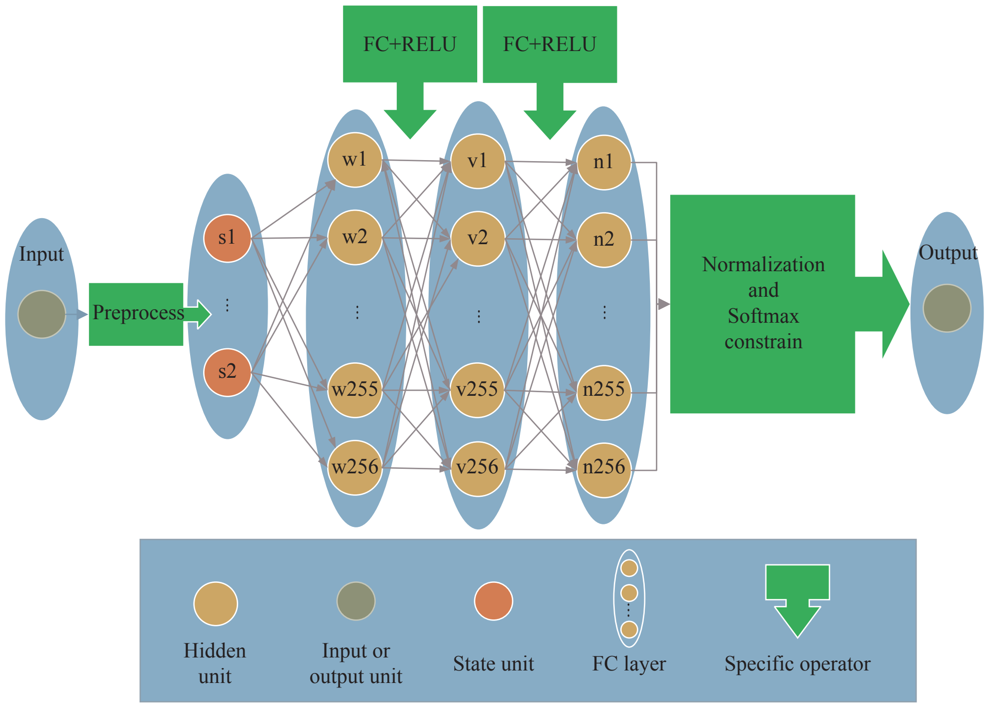

Fig. 3. Neutral network with 3 fully-connected layers.

The final optimization target is defined as follow:

$$reward = \alpha * TT_{waitTime} + (1-\alpha) * RC_{delay} \qquad (9)$$

In all experiments, we use genetic algorithm (GA) as the benchmark algorithm to show the optimization process. The population batch size is set as 160 in each epoch, and asynchronous method is applied for GA in a multi-core process system with 16 cores. For our imitation learning method, we only use asynchronous method in Experiment 4. For the others three experiments, we only use one CPU core to be the actor and one GPU core for the learner.

In formula (9), $TT_{waitTime}$ is the scheduling wait time of TT messages in the outputting ports, RC_{dealy} is the end-to-end delay of RC messages, and α is a balance parameter between two optimization goals. We can get different optimized policy network by adjusting the parameter α.

1. Experiment 1

Experiment 1 is used to show the basic performance of our RLTS method. Typically, the calculation speed of genetic algorithm is related to the size of the population. Under the acceptable level of optimization, we try to make the calculate speed fast.

The results are shown in Tables 1–6, which give the detailed results according to different network configurations. From the results, we can see that our RLTS algorithm performs better than the other algorithms, such as SMT, GA and random search (RS), under different scales of the networking scenarios with different numbers of messages. Both the delays of TT and RC messages can be optimized according to our RLTS algorithm. The performance of the RS algorithm is taken as the benchmark to calculate the optimization degree of our method, which is shown in formula (10). Thus, the detailed optimization degrees compared with RS according to the results shown in Tables 1–6 can be calculated, which are 12.5%, 21.2%, 12.0%, 20.1%, 23.1% and 27.4% for the RC messages respectively.

$$rate\% = (result_{RS} - result_{RLTS})/result_{RS} \times 100\% \tag{10}$$

Table 1. 1050 TT messages and 1932 RC messages

Method	Speed (s/epoch)	TT wait time (μs)	TT Delay (μs)	RC delay (ms)
SMT	128.8	1355.3	1436.60	55.24
RS	2.17	7.82	89.12	46.21
GA	171.4	5.80	87.9	45.03
RLST	4.58	4.76	86.06	40.42

Table 2. 1050 TT messages and 1404 RC messages

Method	Speed (s/epoch)	TT wait time (μs)	TT delay (μs)	RC delay (ms)
SMT	128.8	1248.6	1329.90	35.09
RS	2.17	12.20	93.50	30.29
GA	171.4	5.85	87.15	28.08
RLST	4.58	5.61	86.91	23.87

If SMT algorithm is taken as the benchmark, we can find our RLTS algorithm has more advantages for both small delays and fast responding speed. Generally speaking, the TT message delays of our RLTS algorithm are nearly one percent of the delays of SMT algorithm, and the RC message delays of our RLTS algorithm are also close to the half of the delays of SMT algorithm. When considering the computation speed, the speed of our RLTS algorithm is dozens of times faster than SMT. For most networking scenarios, our RLTS algorithm can obtain an optimal solution within 10 seconds.

Table 3. 570 TT messages and 1932 RC messages

Method	Speed (s/epoch)	TT wait time (μs)	TT delay (μs)	RC delay (ms)
SMT	28.9	1278.6	1381.43	46.23
RS	1.44	6.54	109.37	44.24
GA	31.5	4.92	107.78	41.89
RLST	2.52	2.97	105.80	38.93

Table 4. 570 TT messages and 966 RC messages

Method	Speed (s/epoch)	TT wait time (μs)	TT delay (μs)	RC delay (ms)
SMT	28.9	1278.6	1381.43	20.23
RS	1.44	6.09	108.92	18.87
GA	37.3	5.59	108.41	18.29
RLST	2.52	5.27	108.10	15.07

Table 5. 1050 TT messages and 966 RC messages

Method	Speed (s/epoch)	TT wait time (μs)	TT delay (μs)	RC delay (ms)
SMT	128.8	1355.3	1436.60	25.52
RS	2.17	7.81	89.11	20.03
GA	160.2	5.80	87.90	19.60
RLST	4.58	4.65	85.95	15.42

Table 6. 1581 TT messages and 966 RC messages

Method	Speed (s/epoch)	TT wait time (μs)	TT delay (μs)	RC delay (ms)
SMT	492.7	1201.1	1267.43	33.59
RS	6.63	13.68	80.01	21.66
GA	298.6	7.63	73.96	20.57
RLST	12.64	7.03	73.36	15.73

Besides, we can find out that our RLTS algorithm can obtain more advantage for the scheduling optimization with a large radio of the number of TT messages to the number of RC messages. It is intuitive that, if there are more TT messages, there would be more slots among TT scheduling windows, which finally results in more complexity for solving the scheduling problem, so that our method will have more choices to figure out the optimal scheduling solution.

2. Experiment 2

In this experiment, we will test our RLTS algorithm in a scenario including 1050 TT messages and 1932 RC messages but with different message lengths by randomly multiplying a coefficient between 0.5 and 2. The results are shown in Figs.4–6 including the delays of TT messages and RC messages and the standard deviation of slots among TT scheduling windows respectively. The horizontal arises in Figs.4–6 are the epoch numbers.

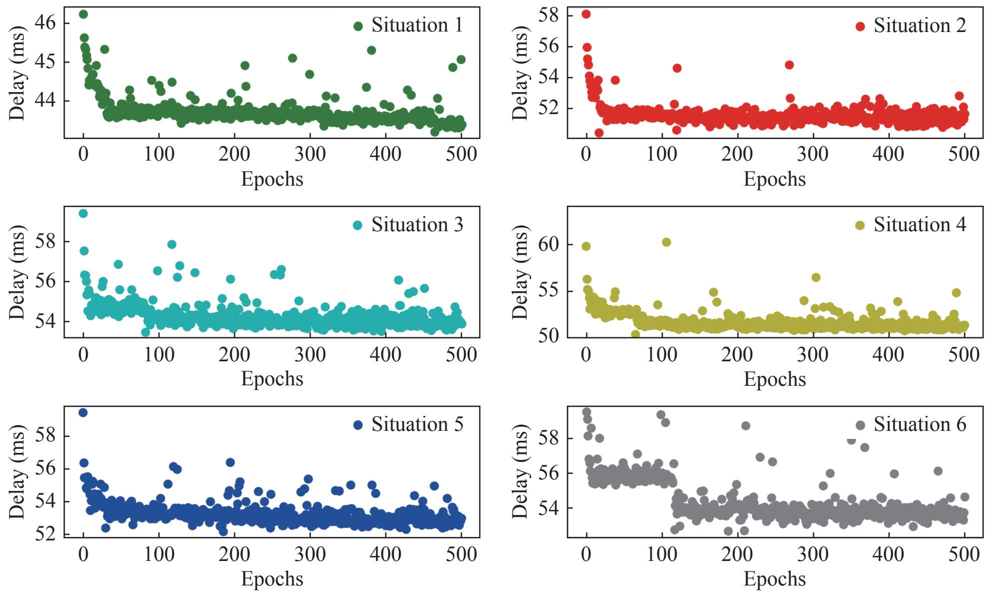

Fig. 4. RC message delay of RLTS algorithm in different situations.

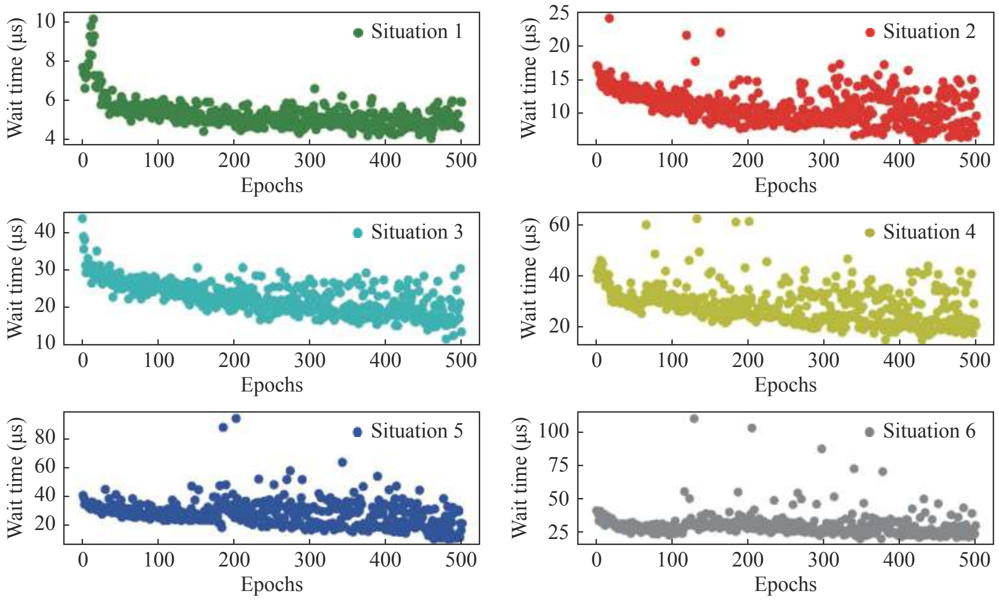

Fig. 5. TT message wait time of RLTS algorithm in different situations.

We can find that our RLTS algorithm still can optimize the delays for both TT and RC messages at the same time with different message lengths. The test is performed six times, namely situation 1 to situation 6. In each test situation, the lengths of messages are randomly assigned. In Fig.6, it shows that, the standard deviation for slots between the adjacent TT scheduling windows is not stably decreasing with the process for optimization, but converges to different levels which not always gather into a same degree. For better comparison, we also draw the baseline of RS algorithm in Fig.6. We can find the convergence trend of our RLTS algorithm is not always consistent with the RS algorithm, which just means it is not always better for scheduling problem with a more uniform distribution of scheduling slots.

3. Experiment 3

Next, we design Experiment 3 to illustrate the gen-

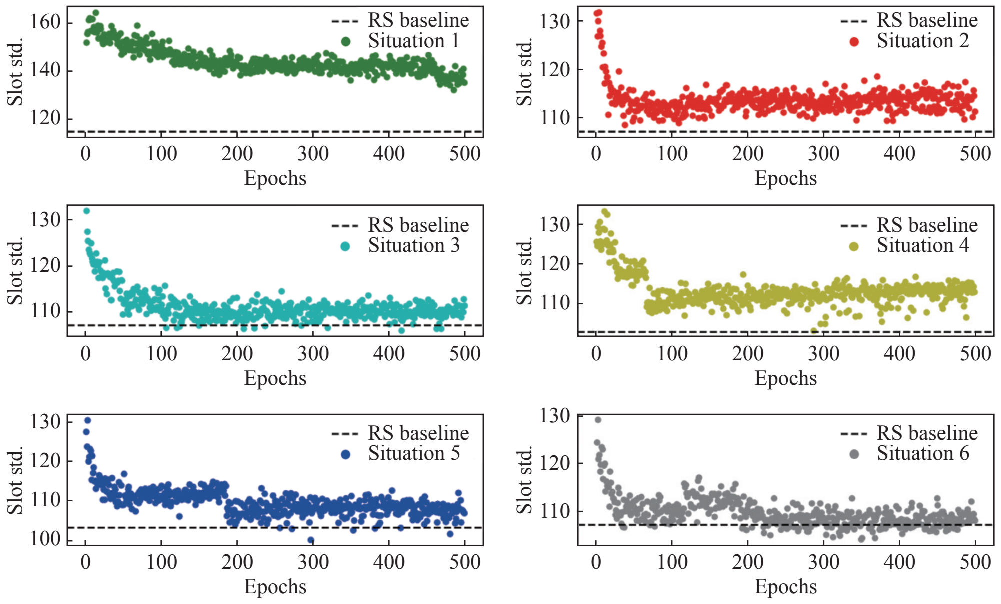

Fig. 6. Standard deviation of slots in different situations.

eralization ability of our trained model. A series of message length changes are applied to check whether our RLTS algorithm could quickly adjust its inherent mechanism to suit the new network situation. In Experiment 3, 1050 TT messages and 1932 RC messages are used to train our neural network with RLTS, and the same number of messages with different frame length are used to test the generalization. Different from Experiment 2, this experiment uses a trained model for fine-tune, while Experiment 2 is trained from the beginning, so the number of epochs in Experiment 3 is much less than Experiment 2.

The basic message setting in Experiment 3 is similar to the case with 1050 TT messages and 1932 RC messages in Experiment 1. Then we change the message lengths randomly according to the corresponding message IDs which also are selected randomly. In this way, we can guarantee the randomness of the generating for messages. Totally, over half the messages are randomly selected out to change their message lengths by multiplying a random coefficient to the original lengths. We will investigate the generalization ability of our RLTS algorithm. Results after 50 epoch times are shown in the follow context, which mainly illustrate the fitting curves for message delays and epoch times.

Operation 1 (small change) Operation 1 shows the influence of a relatively small change for message length. The range of the multiplying coefficient is limited within the scope of 0.9 to 1.2.

Operation 2 (medium change) Operation 2 adopts a larger change for message length than Operation 1, where the range of the multiplying coefficient is between 0.8 and 1.4.

Operation 3 (large change) Operation 3 will have the maximal change for message length, where the range of the multiplying coefficient is between 0.8 and 1.5.

The results are shown in Fig.7, Fig.8 and Fig.9. For better comparison, we also take RS algorithm results as the baseline in Figs.7–9. It can be seen that our RLTS model can attain much better results than the RS algorithm in a short period of time, which just shows our model can quickly adapt itself to the similar configuration states. Considering our model is a self-exploration method, it is not strange that our RLTS model can achieve better and fast solution than RS algorithm based on pre-train. According to the change degree of message length from Operation 1 to Operation 3, the randomness also raises with an increasing scope for the coefficient. When the message length distribution is completely different, the goal of our algorithm is changed to search for a feasible solution first, and then optimize it. In all the test cases in Experi-

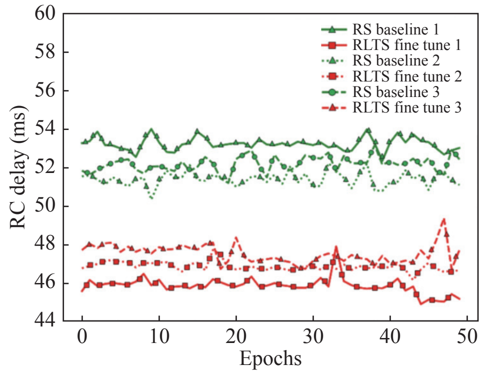

Fig. 7. Delay of RC messages.

ments 3, results show that our RLTS model has a good generalization ability and can handle with the similar configuration fast. Besides, the results also show that the optimal scheduling solution does not always prefer to the uniform distribution for the TT scheduling windows.

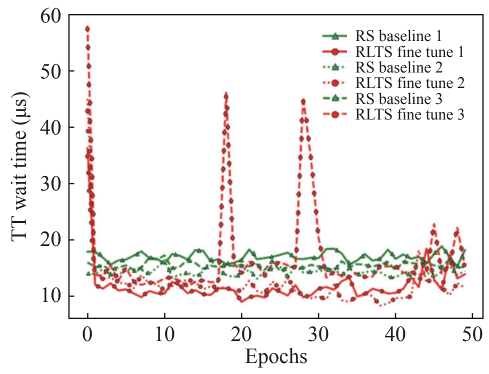

Fig. 8. Wait Time of TT messages.

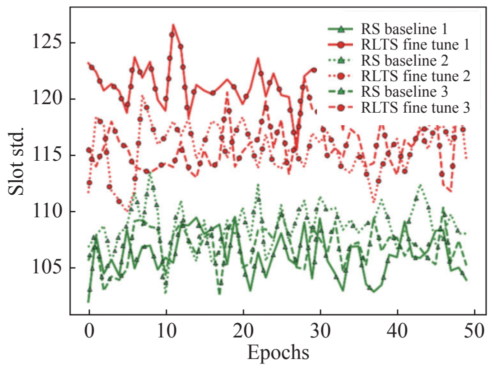

Fig. 9. Standard deviation of slots among TT scheduling windows.

4. Experiment 4

Finally, we apply the asynchronous method for fast and better exploration. Since our test platform is a multi-core process system with up to 16 CPU cores, we choose several cores to explore and collect schedule tables to speed up the training process. Results are shown in Table 7.

Table 7. Multi-process with 1050 TT and 1932 RC

Number of processors	TT wait time (μs)	TT delay (μs)	RC delay (ms)
1(RS)	7.82	89.12	42.21
1	6.63	86.06	40.42
2	298.6	85.70	39.92
4	12.64	85.78	39.74
8	492.7	85.09	39.63
16	492.7	85.64	39.61

Also the detailed information is shown in Figs.10–12. We can find the asynchronous method based on multi-core processing greatly accelerates the convergence speed for RC message delays. If more processors take part in the exploration and calculation process, our RLTS algorithm can achieve a much fast convergence speed, also attain better scheduling solution with smaller RC delays, though the optimization degree is not so obvious. For TT message delay, different number of participating processors has a much smaller effect to the convergence speed than for RC message. Since we mainly focus on RC delay, there is no obvious influence for the convergence speed or the final TT wait time if more processors participate in the whole process as shown in Fig.11.

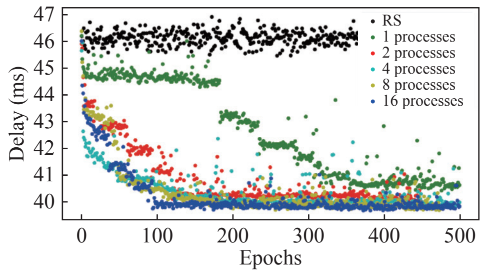

Fig. 10. Delay of RC messages for multi-cores test.

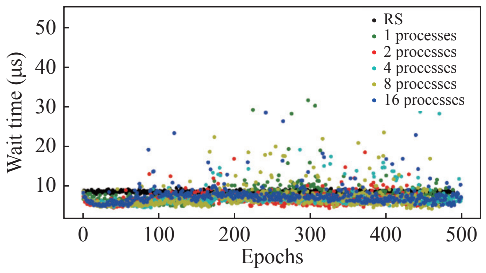

Fig. 11. TT messages wait time for multi-cores test.

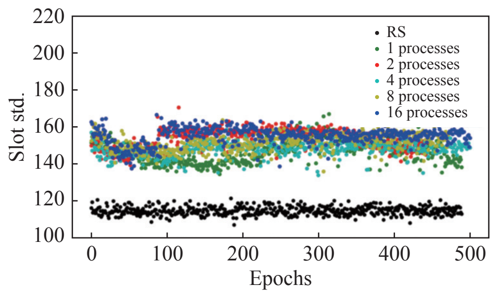

Fig. 12. Standard deviation of slots among TT scheduling windows for multi-cores test.

At last, Fig.12 shows the standard deviation of slots among TT scheduling windows. We can find the standard deviation of RS algorithm is much smaller than our RLTS algorithm which just means the diversity of scheduling gaps generated by RS are more uniform than the results by our RLTS.

So the scheduling gaps by RS seem as more expected as the porosity scheduling strategy does. It can be explained since RS algorithm adopts the completely random method to make the exploration and this kind of exploration method potentially would result in normal distribution for the solutions. As shown in Experiment 1, our RLTS algorithm has more advantage in delays both for TT and RC messages than RS algorithm, and it can reduce the worst-case delay of RC averagely by 20% and the wait time of TT averagely by 44%. So a more uniform TT scheduling window distribution does not always mean a better scheduling result both for TT and RC messages as our RLTS algorithm could achieve better scheduling results but with larger standard deviation of slots than RS algorithm.

VI. Analysis and Discussion

When considering the application of TTEthernet in real time networking environment especially in avionics context, we propose to use reinforcement learning to solve and optimize schedule problem, which can achieve excellent performance beyond SMT and other heuristic methods like GA.

Due to the periodicity of the TT message, its own scheduling gap is relatively uniform. In particular, when all TT messages have the same frame length and the same period, a scheduling table with a uniform distribution for the slots among TT scheduling windows could be the optimal solution with the lowest delays for both TT and RC messages. However, in the practical networking scenario, the distribution of TT messages could not be so even, so it is really hard to assure that the end-to-end delays for all messages are always small. In fact, due to the diversity of TT and RC messages, it is almost impossible to encounter an example with a uniform distribution for the slots to solve the scheduling problem [34]. In fact, it needs to adjust the offsets of messages repeatedly to ensure that the scheduling windows could be available and the delays are relatively small. And this kind of work is quite complicated and depressing.

To solve the problem, we adopt the reinforcement learning method, namely the RLTS algorithm, to minimize the delays both for TT and RC messages and can achieve an optimal scheduling table. By using neural network, it can obtain satisfactory scheduling results in a short time with a good generalization ability. Besides, our RLTS algorithm with tree search strategy can obtain a much fast calculation speed. When combining the asynchronous method to our model, it can further accelerate several times which just lays a foundation for the further optimization.

According to the experimental results based on an industrial networking case with more than 1000 TT messages and 1900 RC messages, the traditional view that the uniform porosity for TT scheduling windows could contribute to better performance for RC messages is not always the most suitable strategy for general situations. The best scheduling pattern of TTEthernet might lay in the middle between the centralized scheduling pattern and porosity scheduling pattern.

References

[1] IEEE Std 802.3-2022: 1968, IEEE Standard for Ethernet, Available at: *https://ieeexplore.ieee.org/document/9844436*.

[2] Y. H. Lee, *Safety and Certification Approaches for Ethernet Based Aviation Databuses*, Technical Report, DOT/FAA/AR-05/52, Federal Aviation Administration, 2005.

[3] AEE Committee, *Arinc specification 664 p7, ARINC 664 aircraft data network*, Avionics full duplex switched Ethernet (AFDX) network, Technical report, Annapolis, MD, USA: Aeronautical Radio Inc., 2005.

[4] R. I. Davis, A. Burns, R. J. Bril, *et al.*, "Controller area network (CAN) schedulability analysis: refuted, revisited and revised," *Real-Time Systems*, vol.35, no.3, pp.239–272, 2005.

[5] SAE. SAE AS6802 Time-triggered Ethernet, Warrendale: SAE International, 2011.

[6] J. D. Decotignie, "Ethernet-based real-time and industrial communications," *Proceedings of the IEEE*, vol.93, no.6, pp.1102–1117, 2005.

[7] W. Steiner, "An evaluation of SMT-based schedule synthesis for time-triggered multi-hop networks," in *2010 31st IEEE Real-Time Systems Symposium*, San Diego, CA, USA, pp.375–384, 2010.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol.518, no.7540, pp.529–533, 2015.

[9] S. Daftry, J. A. Bagnell, and M. Hebert, "Learning transferable policies for monocular reactive MAV control," in *15th International Symposium on Experimental Robotics*, Nagasaki, Japan, pp.3–11, 2016.

[10] D. Silver, J. Schrittwieser, K. Simonyan, *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol.550, no.7676, pp.354–359, 2017.

[11] A. Ecoffet, J. Huizinga, J. Lehman, *et al.*, Go-explore: a new approach for hard-exploration problems, arXiv preprint arXiv: 1901.10995, 2019, doi: 10.48550/arXiv.1901.10995.

[12] H. J. Dai, E. B. Khalil, Y. Y. Zhang, *et al.*, "Learning combinatorial optimization algorithms over graphs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, Long Beach CA, USA, 2017.

[13] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Proceedings of the 28th International Conference on Neural Information Processing Systems*, Montreal, Canada, pp.2692–2700, 2015.

[14] V. Mnih, A. P. Badia, M. Mirza, *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning*, New York, NY, USA, pp.1928–1937, 2016.

[15] R. Dobrin and G. Fohler, "Implementing off-line message scheduling on controller area network (CAN)," in *ETFA 2001. 8th International Conference on Emerging Technologies and Factory Automation*, Antibes-Juan les Pins, France, pp.241–245, 2001.

[16] R. I. Davis and A. Burns, "Robust priority assignment for messages on controller area network (CAN)," *Real-Time Systems*, vol.41, no.2, pp.152–180, 2009.

[17] R. Marau, L. Almeida, P. Pedreiras, *et al.*, "Utilization-based schedulability analysis for switched Ethernet aiming dynamic QoS management," in *2010 IEEE 15th Conference on Emerging Technologies & Factory Automation*,

Bilbao, Spain, pp.1–10, 2010.

[18] E. Suethanuwong, "Scheduling time-triggered traffic in TTEthernet systems," in *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation*, Krakow, Poland, pp.1–4, 2012.

[19] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Computers & Operations Research*, vol.13, no.5, pp.533–549, 1986.

[20] Y. J. Zhang, F. He, G. S. Lu, *et al.*, "An imporosity message scheduling based on modified genetic algorithm for time-triggered Ethernet," *Science China Information Sciences*, vol.61, article no.019102, 2018.

[21] D. Tămaş-Selicean, P. Pop, and W. Steiner, "Design optimization of TTEthernet-based distributed real-time systems," *Real-Time Systems*, vol.51, no.1, pp.1–35, 2015.

[22] R. E. Korf, "Depth-first iterative-deepening: An optimal admissible tree search," *Artificial Intelligence*, vol.27, no.1, pp.97–109, 1985.

[23] R. S. Sutton, D. A. McAllester, S. P. Singh, *et al.*, "Policy gradient methods for reinforcement learning with function approximation," in *Proceedings of the 12th International Conference on Neural Information Processing Systems*, Denver, CO, USA, pp.1057–1063, 1999.

[24] H. R. Li, F. He, Z. Zheng, *et al.*, "Time-triggered communication scheduling method based on reinforcement learning," *Journal of Beijing University of Aeronautics and Astronautics*, vol.45, no.9, pp.1894–1901, 2019. (in Chinese)

[25] D. Silver, A. Huang, C. J. Maddison, *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol.529, no.7587, pp.484–489, 2016.

[26] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *17th European Conference on Machine Learning*, Berlin, Germany, pp.282–293, 2006.

[27] D. Silver, G. Lever, N. M. O. Heess, *et al.*, "Deterministic policy gradient algorithms," in *Proceedings of the 31st International Conference on International Conference on Machine Learning*, Beijing, China, pp.I-387–I-395, 2014.

[28] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the Twenty-First International Conference on Machine Learning*, Banff, Canada, pp.1–8, 2004.

[29] D. Horgan, J. Quan, D. Budden, *et al.*, "Distributed prioritized experience replay," in *6th International Conference on Learning Representations*, Vancouver, Canada, 2018.

[30] L. X. Zhao, P. Pop, Q. Li, *et al.*, "Timing analysis of rate-constrained traffic in TTEthernet using network calculus," *Real-Time Systems*, vol.53, no.2, pp.254–287, 2017.

[31] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, Fort Lauderdale, FL, USA, pp.315–323, 2011.

[32] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations*, San Diego, CA, USA, 2015.

[33] D. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (ELUs)," in *4th International Conference on Learning Representations*, San Juan, Puerto Rico, 2016.

[34] Y. J. Zhang, F. He, G. S. Lu, *et al.*, "Scheduling rate-constrained flows with dynamic programming priority in time-triggered ethernet," *Chinese Journal of Electronics*, vol.26, no.4, pp.849–855, 2017.

HE Feng received the Ph.D. degree in communication and information systems from the School of Electronic Information Engineering, Beihang University, China, in 2008. He is an Associate Professor with the School of Electronic Information Engineering, Beihang University, China. In this area, he has published over 76 peer-reviewed papers and 2 books. He has presided more than ten major projects in total, such as National Natural Science Foundation of China, National 863 Program and Civil Aircraft Research. His research interests include digital communication technology, communication network theory and technology, avionics integration, software defined network, embedded system, and real-time network.

XIONG Li was born in Nanchang, China. He received the B.S. degree in electronics and information engineering, Beihang University in 2020. He is currently pursuing the M.E. degree in communication and information system at the School of Electronic Information Engineering, Beihang University, China.

ZHOU Xuan (corresponding author) received the Ph.D. degree in communication and information systems from Beihang University in 2021. She is currently a Postdoc at the School of Electronic Information Engineering, Beihang University. The main research direction is real-time communication system, scheduling design and performance evaluation. She has published more than ten papers in related fields. (Email: lomoo@buaa.edu.cn)

LI Haoruo was born in Chengdu, China. He received the B.S. degree in electronics and information engineering, Beihang University in 2018. Then, he received the M.S. degree in communication and information system at the School of Electronic Information Engineering, Beihang University, China, in 2020.

XIONG Huagang received the Ph.D. degree in communication and information system from the School of Electronic Information Engineering, Beihang University, China, in 1998. He is currently a Full Professor with Beihang University, China. He has published over 305 peer-reviewed SCI/EI papers and 3 books. He has presided more than twenty major projects in total, such as National Natural Science Foundation of China, National 863 Program and Civil Aircraft Research. His research is focused on communication network theory and technology, avionics information integration, airborne network, and standards. He is the chief of BUAA-TTTech Time-Triggered Technology Joint Laboratory (TTTJL) at Beihang University. He is also the head of the Avionics and Bus Communications Research Team (ABC) at School of Electronic Information Engineering, Beihang University. Furthermore, he is a Member of China Aviation Electronics Standardization Committee, the director of Beijing Electronic Circuit Research Association, a Member of Avionics and Air Traffic Control Branch of China Society of Aeronautics and Astronautics, and an Expert of Civil Aircraft Scientific Research Group.
(Email: hgxiong@buaa.edu.cn)