Mitigation of Scheduling Violations in Time-Sensitive Networking using Deep Deterministic Policy Gradient

Boyang Zhou
Lehigh University
Bethlehem, PA, USA
boz319@lehigh.edu

Liang Cheng
EECS, University of Toledo
Toledo, OH, USA
liang.cheng@utoledo.edu

ABSTRACT

Time-Sensitive Networking (TSN) is designed for real-time applications, usually pertaining to a set of Time-Triggered (TT) data flows. TT traffic generally requires low packet loss and guaranteed upper bounds on end-to-end delay. To guarantee the end-to-end delay bounds, TSN uses Time-Aware Shaper (TAS) to provide deterministic service to TT flows. Each frame of TT traffic is scheduled a specific time slot at each switch for its transmission. Several factors may influence frame transmissions, which then impact the scheduling in the whole network. These factors may cause frames sent in wrong time slots, namely misbehaviors. To mitigate the occurrence of misbehaviors, we need to find proper scheduling for the whole network. In our research, we use a reinforcement-learning model, which is called Deep Deterministic Policy Gradient (DDPG), to find the suitable scheduling. DDPG is used to model the uncertainty caused by the transmission-influencing factors such as time-synchronization errors. Compared with the state of the art, our approach using DDPG significantly decreases the number of misbehaviors in TSN scenarios studied and improves the delay performance of the network.
[1]

CCS CONCEPTS

• **Networks** → **Network algorithms**; **Network performance evaluation**;

KEYWORDS

Time-Sensitive Networking, Deep Deterministic Policy Gradient, Scheduling

ACM Reference Format:
Boyang Zhou and Liang Cheng. 2021. Mitigation of Scheduling Violations in Time-Sensitive Networking using Deep Deterministic Policy Gradient. In *Workshop on Flexible Networks Artificial Intelligence Supported Network Flexibility and Agility (SIGCOMM '21), August 27, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3472735.3473385

[1]The research was done while Dr. Liang Cheng was at Lehigh University. Institutional affiliation is provided for identification purposes.

SIGCOMM '21, August 27, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8634-0/21/08.
https://doi.org/10.1145/3472735.3473385

1 INTRODUCTION

Time-Sensitive Networking (TSN) is developed by the Time-Sensitive Networking Task Group, extending IEEE 802.1Q to provide a deterministic performance guarantee to Time-Triggered (TT) traffic, which has a hard delay requirement. TSN can be applied to many fields, such as industrial automation systems, telesurgery, and autonomous driving [9]. Reliable data exchange is the prerequisite for the real-time control in these systems [16]. Data flows between sensors, actuators, and controllers usually have hard delay requirements. TSN is a potential candidate to serve as the communication backbone in these real-time systems because it can guarantee deterministic delay for TT traffic.

In TSN, each switch schedules a specific time slot for the transmission of each TT frame to provide deterministic service. However, the determinism can be violated when TT frames are not sent in their scheduled time slots (defined as misbehaviors), which is further discussed in Section 2.1. Monitoring techniques are needed to detect misbehaviors, which can be treated as scheduling violations. Previously, several techniques have been developed to monitor data center networks or local area networks, such as NetFlow [3] and SNMP [12]. Most techniques are not designed for TSN, and thus they may not be able to discover the misbehaviors in TSN networks. Recently, a new monitoring system, TSN-insight [1], is proposed to record the TSN network status. However, monitoring can only discover the occurrence of misbehaviors (detailed classifications of misbehaviors are discussed in Section 3.1). Our work aims to reduce the number of misbehaviors (i.e. to mitigate the scheduling violations) by using reinforcement learning methods. The benefit of choosing reinforcement learning methods is that they do not require datasets with ground truth, which saves a lot of effort to collect data with labels.

Since we model the TSN network with continuous states and actions, we select Deep Deterministic Policy Gradient (DDPG), which is a reinforcement learning method for continuous action space and continuous state space. Our idea is to use DDPG to find suitable scheduling so that the number of misbehaviors is minimized. Meanwhile, the queuing time of frames and the network resource assigned to TT traffic should be optimized. In other words, each frame should not encounter a queuing delay leading to the violation of its delay requirement, and the bandwidth allocated to TT traffic should be minimized so that bandwidth left for the lower priority traffic can be maximized. Our DDPG model implements offline training to find suitable GCLs to reduce the number of misbehaviors. It establishes the foundation for adjusting GCLs online in the future.

In this paper, Section 2 introduces the background knowledge of TSN and DDPG. Section 3 describes the type of training we use,

the way we apply the DDPG model to TSN, and the selection of states, rewards, and actions of the model. Section 4 evaluates the performance of our DDPG model using OMNeT++ simulator. The worst-case upper bounds for TT frames when our DDPG model is used are provided in Section 4 as well. Section 5 concludes the performance and discusses the future work.

2 BACKGROUND KNOWLEDGE

In this section, we discuss the basic concepts of TSN and DDPG. TSN is one of the networking technologies designed for data transmissions in real-time systems. DDPG is a widely used reinforcement learning algorithm for continuous control [6].

2.1 TSN

TSN is designed for real-time applications. It takes advantage of IEEE 802.1Qbv Time-Aware Shaper (TAS) to provide deterministic end-to-end delay for TT flows. TAS separates the communication into repeating time cycles, which have the length of a hyperperiod. Hyperperiod is defined as the least common multiple of periods of all TT flows. All TT flows have their periodical patterns within a hyperperiod.

There are eight queues corresponding to eight priorities in TSN switches. Each queue has its own gate. Frames in the queue can be transmitted only when the gate of the queue is open. When several gates are opening at the same time, TSN switches usually choose the frame with the highest priority for the transmission. TAS utilizes Gate Control Lists (GCLs), which contain information about gate statuses of all eight queues in each hyperperiod, to control the transmission of frames of different priorities. A format of GCLs is discussed in Section 3.2.2. Note that TAS works in the output ports of TSN switches. All frames passing TSN switches are enqueued into the eight queues of the output ports based on their priorities. And GCLs control gate statuses of output ports, through which flows pass.

The objective of scheduling in TSN is finding GCLs that enable all TT frames to meet their delay and jitter requirements. Previous work on the scheduling in TSN [4, 10] formulates it as an optimization problem. Constraints of the optimization problem are derived based on factors such as delay and jitter requirements, and traffic patterns of the network. A feasible solution to the problem means that all requirements are fulfilled theoretically. The solution contains the offsets of flows, the number of windows in each hyperperiod, and the start and end time of each window. The offsets of flows define the sending time of each frame in the source.

However, a feasible solution does not guarantee that the delay requirement of TT traffic can be met in real hardware-software operation environments. Each frame is assigned a specific window in each switch on its path by the TSN scheduling. A window is defined as the time period when the gate of the queue is open. Once the frame is not sent during its assigned window, which means that a misbehavior occurs, the scheduling along the path might be violated. There are several possible reasons that can cause the violation/misbehaviors: i) time-synchronization errors between switches and hosts; ii) frame transmission dynamics at hosts; iii) scheduling dynamics in the switch fabric. Time-synchronization errors cause the drift of the assigned window of each TT frame in different switches. Frame transmission dynamics at hosts induces the inaccurate sending time of frames from their sources, which causes unknown jitter and changes the order of frames. Scheduling dynamics in the switch fabric can lead to that the order of frames in the same queue differs during runtime if the frames come from different input ports [7]. Our work aims to mitigate the impact of the uncertainty induced by the three factors on the scheduling in TSN.

2.2 DDPG

Reinforcement learning methods have already been used in the network research areas, such as traffic scheduling [2] [11] and routing optimization [13]. DDPG is a reinforcement learning method, which combines deep neural networks and Q-learning. This algorithm is developed by Google Deepmind [6]. The traditional Q-learning method is a model-free off-policy algorithm for agents to learn the optimal actions in different states. The convergence of Q-learning has been proved [15]. The Q-learning algorithm maintains a Q-table to store Q-values of all pairs of (s, a), where s is the state and a is the action. The Q-value is a measurement of the overall expected reward assuming that action a is taken upon state s. Although Q-learning is a powerful method, it only works for environments with discrete states and discrete actions.

Deep Q-learning can deal with the environments with continuous states [8]. Instead of using a Q-table to memorize Q-values, deep Q-learning uses neural networks to map states to (a, Q-value) pairs. Neural networks in deep Q-learning are used to implement the function approximation so that they can evaluate Q-values of actions in any state. Deep Q-learning performs experience replay, which randomly samples the previous data for training. The experience replay helps smooth the training distribution over past behaviors. However, deep Q-learning cannot be applied to the continuous action domain.

DDPG adopts the idea of deep Q-learning and extends the working domain to environments with continuous states and continuous actions. There is a main network and a target network in DDPG. Both networks have the same structure. As DDPG uses an actor-critic approach, each network contains an actor network and a critic network. The actor network, whose input is the state, is used to predict the action. The critic network is used to evaluate the Q-value of (s, a) pairs. Similar to deep Q-learning, DDPG utilizes the experience replay to train the model. Unlike traditional neural networks, DDPG does not require datasets with ground truth for training. Instead, DDPG uses the target network to approximate Q-values gradually. Figure 1 [17] shows the structure and the training process of DDPG. The loss functions of the actor network and the critic network are shown in the figure. Parameters of both the main network and the target network are updated in each episode based on the gradient.

3 DDPG FOR GCL CONFIGURATION

In our research, we use DDPG to find suitable GCLs to decrease the number of misbehaviors in TSN. Our DDPG model utilizes the knowledge of misbehaviors to model the uncertainty in the network. Thus, in Section 3.1, we provide the classification and

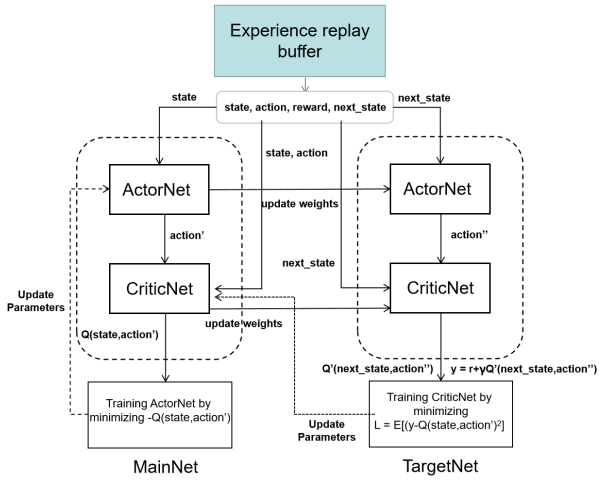


Figure 1: Structure of DDPG

example of misbehaviors. In Section 3.2, we discuss how we apply DDPG to find suitable GCLs.

3.1 The classification of misbehaviors

Feasible GCLs, which enables meeting the delay requirement of TT traffic, can be derived by solving the optimization problem. However, as discussed in Section 2.1, frames may not be sent in their assigned windows, which means that misbehaviors occur due to the jitter and the disorder.

In order to distinguish misbehaviors in the network, we classify misbehaviors into two types. i) If the frame is sent ahead of its assigned window, we treat this type of misbehaviors as lead misbehaviors. ii) If the frame is sent after its assigned window, a lag misbehavior happens. Figure 2 shows an example of a lead misbehavior and a lag misbehavior.

Figure 2 (a) depicts the topology and the traffic pattern used in this example. There are three hosts (H_1, H_2, and H_3) and two TSN switches (S_1 and S_2) in the network. Flows f_1 and f_2 are sent by H_1 and the destination of the two flows is H_3. Flow f_3 is sent from H_2 to H_3. Suppose they have the same period and the same frame size. The hyperperiod is the period of these flows, and each flow sends one frame in each hyperperiod in this case.

Figure 2 (b) shows the correct processing of all frames on different links. In this scenario, the offsets of the frames and the windows assigned to the frames are derived by the scheduling in TSN. As shown in the figure, Frame 1 and Frame 2 should be sent in window w_1 of the output port of S_1, and in window w_3 of the output port of S_2. Frame 3 should be sent in window w_2 of the output port of S_1, and in window w_4 of the output port of S_2. However, if there is jitter, which causes Frame 2 arriving later than Frame 3 at switch S_1, the scheduling of Frame 2 and Frame 3 is violated.

Figure 2 (c) shows the anormal transmission caused by the jitter. Frame 3 is sent ahead of its assigned windows, which is a lead misbehavior. Frame 2 then encounters a lag misbehavior because it is sent in w_2 and w_4 instead of w_1 and w_3.

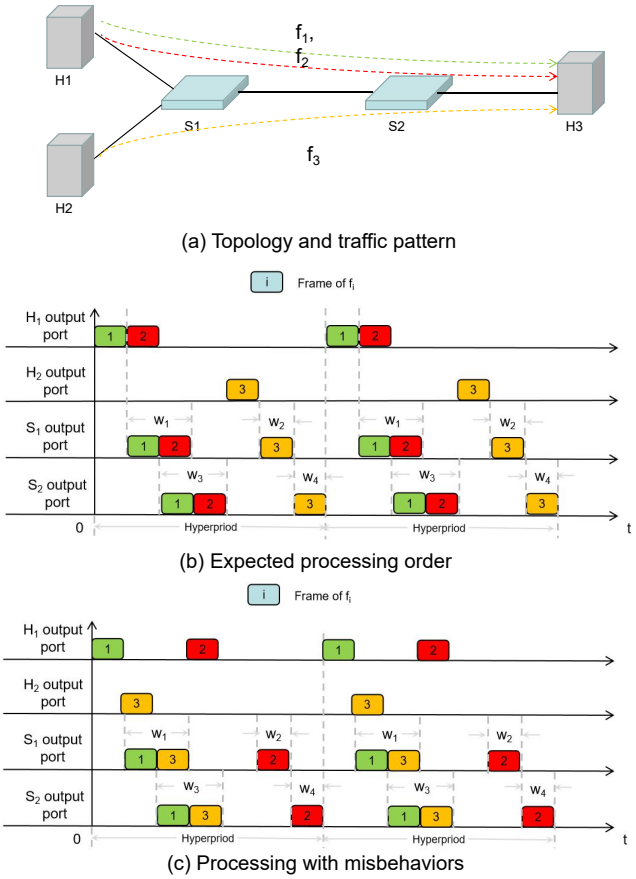


Figure 2: An example of misbehaviors

The aim of using DDPG is to find suitable GCLs to reduce the number of lead and lag misbehaviors. It is impossible to eliminate all misbehaviors in some scenarios, e.g. what is shown in Figure 2 (c). Note that the order of Frame 2 and Frame 3 is reversed. Frame 3 should be sent in w_2, and Frame 2 should be sent in w_1 of the output port of S_1. Since Frame 3 arrives earlier than Frame 2 at S_1, and w_2 is scheduled later than w_1 in this case, Frame 2 and Frame 3 cannot be sent in their assigned windows simultaneously. Even if we increase the size of window w_1 to accommodate Frame 2, there is still a lead misbehavior because Frame 3 must be sent before Frame 2. Thus, there is at least one lead misbehavior or one lag misbehavior of the output port of S_1 in this scenario.

3.2 Settings of the DDPG model

Before discussing the application of DDPG, we should first decide the type of training used in this work. There are two types of training that can be implemented in TSN networks: global training and local training. The global training takes in the information of the whole network, and the model can find suitable GCLs of all output ports in TSN networks. However, this type of training may

need a longer time in collecting information than the one using local information only. Thus, we choose the local training method in our research.

The local training method is a per-port training, which means that each output port is trained separately. There are two advantages of using the local training. i) The local training is simpler and easier to converge compared with global training. ii) Since all information can be obtained locally, local training does not cause extra bandwidth usage and delay induced by collecting the information globally.

Since we use local training, the selection of states, actions, and rewards of the model is based on the local information of the port. We will discuss how to select suitable states, actions, and rewards as follows.

3.2.1 States. In reinforcement learning, the state is used to describe the status of the environment. Agents take in the state of the environment to make the decision on the next step action. Thus, a proper state contributes to the decision of the action with a high Q-value. In our model, the number of misbehaviors in the output port is the most important information, which should be included in the state vector.

In order to detect misbehaviors, TSN switches should have TT traffic tables, which map the frames of TT traffic to their assigned windows in the local memory. This mapping is determined by the scheduling based on [4], which is defined as the optimization method in this paper. TSN switches can discover lead and lag misbehaviors in the network by comparing the sending windows of frames with the assigned windows to the frames in the mapping.

Besides the number of misbehaviors, the total length of windows and total waiting time of windows (a.k.a. the total wait) should be included in the state vector as well because they influence the reward calculation in Section 3.2.3. The total length is the sum of sizes of all windows in each hyperperiod, and the total wait is the sum of the starting time of all windows. For example, if there are two windows in a hyperperiod, which have a length of 40 ms and 60 ms, then the total length is 100 ms. If the first window opens at 30 ms and the second window opens at 100 ms in a hyperperiod, the total wait is 130 ms. In our model, the state vector contains the number of misbehaviors, the total length, and the total wait.

3.2.2 Actions. In our work, we aim to reduce the number of misbehaviors by modifying GCLs used in the network. Thus, actions in our model should be able to find suitable GCLs of output ports. In this paper, we use the NeSTiNg project [5], which is based on OMNeT++ [14] to simulate TSN networks. In NeSTiNg, a GCL can be represented by a time array and a bit-vector array. A bit-vector in the bit-vector array indicates the status of eight gates. If the ith bit of the bit-vector is 1, it means that the gate of the ith queue is open. Otherwise, it is closed. For example, if the time array is [20 ms, 30 ms, 50 ms], and the bit-vector array is [00000000, 11111111, 00000000], it means that all the gates of eight queues are closed in the first 20 ms and the last 50 ms, and all the gates are open in the middle 30 ms in the hyperperiod of 100 ms. In our model, we do not want to change the bit-vector array because the modification of the bit-vector array can ruin the mapping between frames and windows, which is stored in TSN switches. The mapping between frames and windows is given by the results from the optimization method in [4]. Thus, we only need to modify the time array of the GCLs, which means that the action should correspond to the time array.

The actor network in the main network is used to predict the action. We use the softmax activation function in the output layer of the actor network. The sum of all elements is 1 in the output array of the actor network. Since the requirement of a feasible time array is that the sum of all elements equals to the hyperperiod, the new time array can be obtained through multiplying each element of the output array by the hyperperiod. Using this way, the model can produce continuous GCLs with a fixed hyperperiod.

3.2.3 Rewards. The selection of rewards significantly influences the performance of the DDPG model because it guides the model how the parameters should be updated. We consider three requirements for designing rewards. The basic requirement of the model, which has the highest priority, is to minimize the number of misbehaviors. The second requirement is to minimize the bandwidth reserved for the TT traffic so that flows with lower priorities have the opportunity to be sent. Thus, the total length should be minimized. The last requirement is to reduce the queuing time of TT frames because larger queuing delay may lead to higher probability of violating delay requirements. We expect that the window opens immediately once the first frame sent in this window arrives. Thus, we want to minimize the total wait.

In order to achieve these goals, we design the reward as shown in Equation 1, where num_mis is the number of misbehaviors and $transTime$ is the transmission time of all frames in one hyperperiod. We introduce α and β to the reward calculation to avoid the occurrence of zero in the multipliers. If one multiplier is zero, we cannot optimize the variables associated with another multiplier via DDPG. The subtraction of $w_3 * \max(0, (transTime - total\_length))$ is used to punish the situations when the total length is less than the transmission time of all frames because there must be misbehaviors in this type of situations. Since minimizing the total length may increase the total wait, we need to find a balance between the total weight and the total length. Thus, weights w_1 and w_2 are used to decide the significance of total wait and total length.

$$\begin{aligned} reward =& (-num\_mis - \alpha) * (w_1 * total\_length + \\ & w_2 * total\_wait + \beta) \\ & - w_3 * \max(0, (transTime - total\_length)) \end{aligned} \tag{1}$$

After the selection of states, actions, and rewards, we then need to decide the architecture of neural networks in DDPG. The actor network has two hidden layers with the relu activation function, and the critic network has four hidden layers, which have relu, relu, tanh, and linear activation functions, respectively.

4 EVALUATION OF THE DDPG MODEL

In this section, we describe the simulation results of the performance of our DDPG model. The model is integrated with the NeSTiNg project, which is used to simulate TSN networks based on OMNeT++. Then, whether the results from the model can reduce the number of misbehaviors and improve the delay performance of the network can be evaluated. We have modified the NeSTiNg project so that it can detect misbehaviors and provide necessary information needed by our DDPG model. We set $\alpha = 0.1$, $\beta = 7$,

$w_1 = 6$, $w_2 = 3$, and $w_3 = 1$ for the calculation of rewards in the training.

We compare the results of networks using three types of GCLs. The first type of GCLs is obtained from the optimization method, which is described in [4]. The second type of GCLs comes from our DDPG model. The last type of GCLs is assumed to be ideal. All frames can be sent with the minimum queuing time and without misbehaviors when using ideal GCLs.

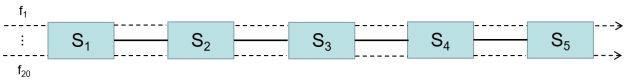


Figure 3: The network topology and the traffic pattern used in the simulation

In the simulation, we use a tandem topology with five switches and twenty TT flows. The topology and the traffic pattern are shown in Figure 3. In this case, there are twenty TT flows transmitted through the same path. Thus, all these twenty flows have to compete for the resources of the same output ports of switches.

We assign the highest priority to all TT flows. Suppose all flows have the same period, which is 100 μs, and the same frame size. The hyperperiod is 100 μs in this case. The bandwidth of each link used in the simulation is 1 Gbps. We use three different frame sizes, which are 1000 bits, 2000 bits, and 3000 bits, in the simulations. However, since headers have different sizes in OMNeT++, the sizes of the frames are not exactly the same, leading to different transmission delays. Because there are five output ports in use and we implement the per-port training (one output port at each switch), we need to train five DDPG models. The order of the training is the same as the order of output ports on the path of flows.

We first compare the network performance in terms of the number of misbehaviors. In order to test the performance of three types of GCLs with the same traffic pattern, we give all frames a constant jitter, which is 5 μs. Figure 4 demonstrates the accumulative number of misbehaviors in five switches using the GCLs from the optimization method varying over time. We can see that the curve is linear because we set a constant jitter of 5 μs. However, using the other two types of GCLs, there is no misbehavior in the network. It means that our DDPG model works well for eliminating misbehaviors in the tandem scenario.

Then, we compared the end-to-end delay performance of networks using the three types of GCLs. Figure 5 illustrates the histograms of delays and the average delay in the networks using the three types of GCLs. In this figure, the x-axis shows the range of delays, and the y-axis represents the density. The network using GCLs derived from the optimization method experiences bipolar delays. The reason for the bipolar delay pattern is that there are misbehaviors, which lead to a large end-to-end delay for misbehaved frames.

Since there are no misbehaviors occurring in networks using the other two types of GCLs, all frames have a similar delay. We can see that our DDPG model can decrease the average delay when the frame size is small. If the frame size is 3000 bits, our DDPG model has a larger average delay than the optimization method.

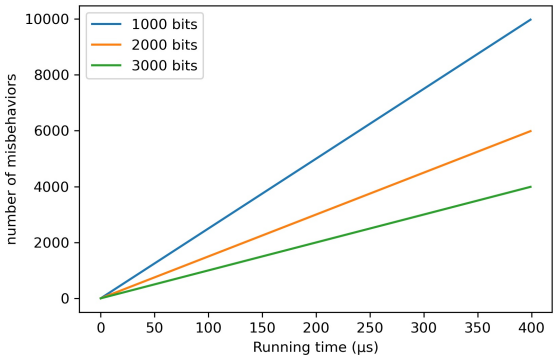


Figure 4: The accumulative number of misbehaviors in five switches using GCLs obtained from the optimization method

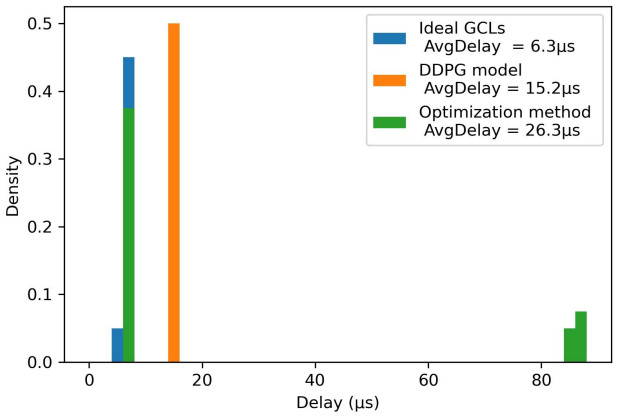
However, larger average delay does not mean that our DDPG model cannot improve the delay performance in this case. As shown in Figure 5 (c), the optimization method causes some frames to encounter an approximately 60 μs delay, which might exceed the delay requirements of TT flows. Meanwhile, all frames encounter an approximately 24 μs delay using our DDPG model. Compared with the results from ideal GCLs, our model always produces a larger end-to-end delay. We can conclude that our DDPG model improves the performance of the network in terms of misbehaviors and delays compared with GCLs derived from the optimization method. Using our DDPG model, we can eliminate all misbehaviors in the tandem scenario.

Besides the simulation results, we can provide an upper bound of the end-to-end delay using our DDPG model. We know that GCLs derived from the optimization method can guarantee deterministic delay if there is no misbehavior. GCLs from our DDPG model can provide a worst-case delay bound for the frame according to Equation 2 if there is no misbehavior or only lead misbehaviors in the last switch on the path of the frame. In this equation, D_{DDPG} is the delay of the frame when using our DDPG model. D_{OP} is the deterministic delay of the frame using GCLs from the optimization method. w^{end}_{DDPG} is the end time of the window, which is assigned to the frame in the last switch on its path, using the GCL obtained by our DDPG model. w^{start}_{OP} is the start time of the window, which is assigned to the frame in the last switch on its path, using the GCL from the optimization method.

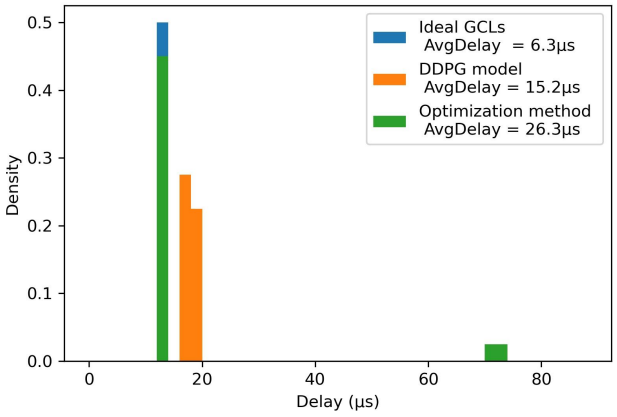
$$D_{DDPG} \leq D_{OP} + w^{end}_{DDPG} - w^{start}_{OP} \tag{2}$$

5 CONCLUSION AND FUTURE WORK

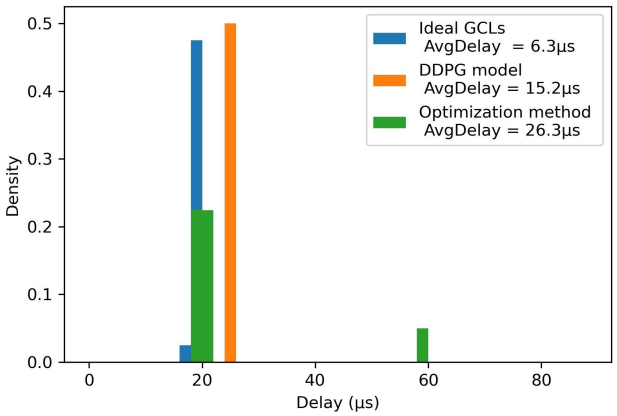
In this paper, we describe a DDPG model to find suitable GCLs in TSN networks to improve network performance. DDPG can model the uncertainty induced by time-synchronization errors, frame transmission dynamics, and scheduling dynamics in the switch fabric. Thus, our DDPG model can be used to provide suitable, although not optimal, GCLs to improve network performance. Compared with the optimization method, our DDPG model can eliminate all misbehaviors and produce better delay performance in the tandem



(a) Delay performance when the frame size is 1000 bits



(b) Delay performance when the frame size is 2000 bits



(c) Delay performance when the frame size is 3000 bits

Figure 5: Delay performance using three types of GCLs

scenario. Using GCLs derived from the DDPG model, no frame encounters a large latency caused by the wrong sending window(s). Therefore, it is more likely that frames will not violate their delay requirements using our DDPG model compared with using the optimization method. Furthermore, we provide a guaranteed delay bound for each frame when using our DDPG model if there is no misbehavior or only lead misbehaviors in the last switch on the path of the frame.

Our DDPG model currently implements offline training in the network to model the uncertainty. It establishes the foundation of using online training in our future work. Online training should dynamically adjust GCLs to reduce the occurrence of misbehaviors. The goal is to update GCLs periodically based on the information from the previous hyperperiod(s) and to offer better adaptability than the current offline training approach.

6 ACKNOWLEDGEMENT

This work is supported by NSF Award No. 1646458. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the author(s) and do not necessarily reflect the views of the sponsors of the research.

REFERENCES

[1] Tianyu Bu, Yi Yang, Xiangrui Yang, Wei Quan, and Zhigang Sun. 2019. TSN-Insight: An Efficient Network Monitor for TSN Networks. *Apnet* (2019).
[2] Sandeep Chinchali, Pan Hu, Tianshu Chu, Manu Sharma, Manu Bansal, Rakesh Misra, Marco Pavone, and Sachin Katti. 2018. Cellular network traffic scheduling with deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.
[3] Benoit Claise, Ganesh Sadasivan, Vamsi Valluri, and Martin Djernaes. 2004. Cisco systems netflow services export version 9. (2004).
[4] Silviu S Craciunas, Ramon Serna Oliver, Martin Chmelík, and Wilfried Steiner. 2016. Scheduling real-time communication in IEEE 802.1 Qbv time sensitive networks. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*. 183–192.
[5] Jonathan Falk, David Hellmanns, Ben Carabelli, Naresh Nayak, Frank Dürr, Stephan Kehrer, and Kurt Rothermel. 2019. NeSTiNg: Simulating IEEE time-sensitive networking (TSN) in OMNeT++. In *2019 International Conference on Networked Systems (NetSys)*. IEEE, 1–8.
[6] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).
[7] Nick McKeown. 1999. The iSLIP scheduling algorithm for input-queued switches. *IEEE/ACM transactions on networking* 7, 2 (1999), 188–201.
[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).
[9] Ahmed Nasrallah, Akhilesh Thyagaturu, Ziyad Alharbi, Cuixiang Wang, Xing Shao, Martin Reisslein, and Hesham ElBakoury. 2018. Ultra-low latency (ULL) networks: A comprehensive survey covering the IEEE TSN standard and related ULL research. *arXiv preprint arXiv:1803.07673* (2018).
[10] Ramon Serna Oliver, Silviu S Craciunas, and Wilfried Steiner. 2018. IEEE 802.1 Qbv gate control list synthesis using array theory encoding. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 13–24.
[11] Jonathan Prados-Garzon, Tarik Taleb, and Miloud Bagaa. 2020. LEARNET: Reinforcement learning based flow scheduling for asynchronous deterministic networks. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)*. IEEE, 1–6.
[12] William Stallings. 1998. *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*. Addison-Wesley Longman Publishing Co., Inc.
[13] Giorgio Stampa, Marta Arias, David Sánchez-Charles, Victor Muntés-Mulero, and Albert Cabellos. 2017. A deep-reinforcement learning approach for software-defined networking routing optimization. *arXiv preprint arXiv:1709.07080* (2017).
[14] Andras Varga. 2010. OMNeT++. In *Modeling and tools for network simulation*. Springer, 35–59.
[15] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8, 3-4 (1992), 279–292.
[16] Martin Wollschlaeger, Thilo Sauter, and Juergen Jasperneite. 2017. The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0. *IEEE industrial electronics magazine* 11, 1 (2017), 17–27.
[17] Junta Wu and Huiyun Li. 2020. Deep Ensemble Reinforcement Learning with Multiple Deep Deterministic Policy Gradient Algorithm. *Mathematical Problems in Engineering* 2020 (2020).