Reinforcement Learning for time-aware shaping (IEEE 802.1Qbv) in Time-Sensitive Networks

Adrien Roberty, Siwar Ben Hadj Said, Frederic Ridouard, Henri Bauer, Annie Geniet

Reinforcement Learning for Time-Aware Shaping (IEEE 802.1Qbv) in Time-Sensitive Networks

Adrien Roberty
University Paris-Saclay, CEA, List,
F-91191, Palaiseau, France
Email: adrien.roberty@cea.fr

Siwar Ben Hadj Said
University Paris-Saclay, CEA, List,
F-91191, Palaiseau, France
Email: siwar.benhadjsaid@cea.fr

Frederic Ridouard
ISAE-ENSMA - LIAS
Futuroscope Cedex, France
Email: frederic.ridouard@ensma.fr

Henri Bauer
ISAE-ENSMA - LIAS
Futuroscope Cedex, France
Email: henri.bauer@ensma.fr

Annie Geniet
University of Poitiers - ISAE-ENSMA - LIAS
Poitiers, France
Email: annie.geniet@univ-poitiers.fr

***Abstract*—Industry 4.0 involves the networking of production equipment. This can be achieved thanks to the Time-Sensitive Networking (TSN) set of network standards. However, this new paradigm brings new challenges because TSN features optimization relies on the dynamic characteristics of the underlying communication network (e.g., network topology, routing strategy, critical flows requirements, etc.). This paper focuses on the case of the IEEE 802.1Qbv standard by exploring the applicability of a Deep Reinforcement Learning (DRL) approach in order to reduce the configuration time of the TSN-specific parameters, compared to exact or heuristic methods.**

I. Introduction

One of the main features of Industry 4.0 is the networking of production equipment. From a network perspective, the most important objective for Industry 4.0 remains the real-time performance of communications. Time Sensitive Networking (TSN) is a set of standards aimed at adding real-time characteristics to wired Ethernet networks. The first benefit of TSN is its ability to ensure higher bandwidth and deterministic communications (high synchronicity, bounded latency and strict latency). The second benefit is its configurable mechanisms that allow supporting a mix of heterogeneous traffic constraints on the same medium. In this paper, we are particularly interested in the IEEE 802.1Qbv standard - *Amendment 25: Enhancements for Scheduled Traffic* [1]. The aim of the standard is to reduce the queuing delay in switches for critical cyclic traffic allowing then to achieve low and bounded end-to-end latency.

In the envisaged scenarios for industry 4.0 the productions lines are reconfigurable. This implies a certain dynamic in network topology or network flows, which makes the configuration of TSN mechanisms challenging. Alone the scheduling in IEEE 802.1Qbv can lead to a NP-hard well-known problem [2]. In the literature, the main approach is generally based on engineering tools: simulation tools like RTaW Pegase[1]or mathematical optimization tools like Integer Linear Programming (ILP) formulations or Satisfiability Modulo Theories (SMT) solvers [3]. These engineering tools are not suitable for dynamic configuration of IEEE 802.1Qbv because they need a pre-knowledge of all the flows that could be in the network. In fact, the configuration process includes retrieving the list of existing flows in the network, the topology, the characteristics of the new flow, providing this information to the engineering tools in order to find the new Qbv configuration to deploy. This could take a few hours before being able to accept/reject the flow and deploy the decided configuration. For these reasons, existing engineering tools are rather suitable for closed network (i.e. where the flows are known in advance and there will be no new flows during the operation of the network) and for offline use (i.e. before deploying the network).

To solve the above challenges, we need a method able to quickly respond to any new event (new flow, change in topology, change in flows configuration, etc.) by deciding the adequate scheduling to be deployed in the network. This method should have a small execution time (in seconds). For these reasons, we leverage the Reinforcement Learning (RL) techniques to design a TSN scheduling algorithm.

RL is already widely used for routing in computer networks [4]. Our objective is to show that an RL agent is able to configure the scheduling of the IEEE 802.1Qbv within a reasonable time. To achieve this goal, we use simulations to train and evaluate the agent. The simulations are done with the OMNeT++/INET network simulator. INET is an open-source model library (that provides the TSN models) for the OMNeT++ simulation environment.

The rest of the paper is structured as follows: section II summarizes the state of the art and section III describes the different components in the proposed solution. Section IV explains how we set up the training loop and provides ways to evaluate the quality of the schedules decided by the agent. Section V concludes the paper and gives perspectives.

II. Related Work

The TSN configuration challenge has been addressed by several works. For example, [5] presents schedulability anal-

[1]Available at https://www.realtimeatwork.com/rtaw-pegase/

ysis for the real-time traffic crossing the TSN network. The proposed framework enables to assess whether time constraints are met. The common approach to compute a deterministic scheduling for IEEE 802.1Qbv relies on Integer Linear Programming (ILP) formulation [2], [6]. Such approach is efficient on small networks, but may take a long time to converge on much larger networks. Furthermore, they are offline methods, unsuitable for open and reconfigurable networks. Another example is [7]. This solution allows online reconfiguration of an IEEE 802.1Qbv based network. However, it relies on an admission control mechanism, they don't modify the Qbv time cycle in the switches. The online schedule reconfiguration remains an open problem in TSN [8].

The use of AI techniques seems to be promising for network management. For instance, [9] proposes to use RL technique for scheduling streams in 5G deterministic asynchronous networks. The proposed solution only configures the Asynchronous Traffic Shaper (ATS) mechanism described in the IEEE 802.1Qcr standard. In [10], authors use AI techniques to determine if a possible configuration is feasible, i.e. if it meets the application requirements. To do so, they test simple supervised and unsupervised learning algorithms in order to classify possible configurations as feasible/non-feasible. The main drawback of this solution is that the AI is only effective on a given topology. When the topology changes, their AI needs to be retrained. In addition, the proposed solution can't be used for online configuration.

Closer to our solution, [11] uses DRL to schedule and route mixed-criticality flows in a Deterministic Networking (DetNet) environment. DetNet operates at the layer 3 whereas TSN operates at layer 2. In [12], the authors use DRL to assist their IEEE 802.1Qbv scheduling algorithm. However, their work uses the no-wait model for TSN scheduling, introduced in [13]. In this model, the scheduling is done in the clients, which implies to have a pre-knowledge about each flow.

III. RL-BASED SCHEDULING FOR IEEE 802.1QBV

Here, we detail the considered environment as well as the different assumptions taken to facilitate the training task, then we present the RL formalization that will allow to configure IEEE 802.1Qbv.

A. Network and traffic Model

An example of a TSN network is shown in Figure 1. All topologies are done in a way that allows the IEEE 802.1Qbv to be the same on each switch.

The IEEE 802.1Qbv mechanism is similar in idea to the Time-Division Multiple Access (TDMA) method. The transmission time is divided into cycles of constant duration. This cycle is itself divided into time sequences of varying lengths. Then, each sequence is assigned to a given traffic class.

Figure 1 shows an example of how TSN works on the network. As shown on this figure, we assume two classes of traffic: critical traffic with high priority and 'normal' traffic, known as Best Effort (BE), with low priority. This simplifies the scheduling of the streams: there are in fact only two time

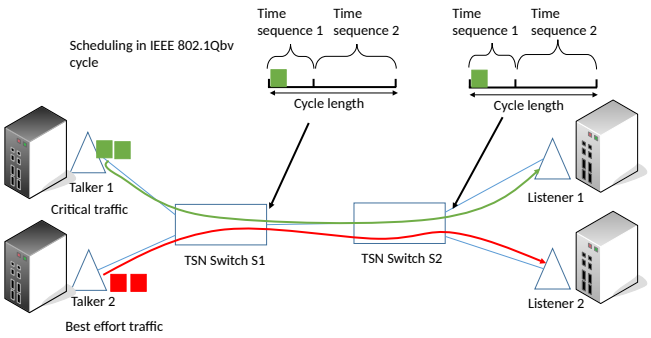

Figure 1. An illustrative example of two talkers (Talker 1 for critical traffic and Talker 2 for BE traffic) exchanging with two listeners (Listener 1 for critical traffic and Listener 2 for BE traffic) via TSN network composed of two TSN switches

sequences. The first is used by critical traffic while the second is for the rest of the traffic. We made this assumption because of the difficulty of making DRL work. It makes it possible to have simple actions. We assume that end stations acting as talkers and listeners are fixed. We considered two talkers and two listeners.

For RL training to be effective, the environment should change from one episode to another. In our case, the number of switches (i.e. number of hops) connecting talkers to listener as well as the theoretical capacity of links varies. The critical traffic packets size is randomly determined in each episode. Also, their emission interval varies from episode to another. Furthermore, at the beginning of each episode, a new latency deadline is randomly computed.

To evaluate whether the IEEE 802.1Qbv schedule allows ensuring deterministic communications, we need to check that the maximum allowed end-to-end delay (i.e. deadline) is not exceeded. This can be achieved through a good IEEE 802.1Qbv schedule as we assume that all switches have the same processing delay and all links have the same capacity.

We assume that the duration of the time sequence reserved for critical traffic should be equal to two times the duration of the emission of one frame. We do this to ensure that critical traffic has time to reach its destination. Under these assumptions, the configuration of IEEE 802.1Qbv implies determining the adequate value for the cycle duration (in nanoseconds).

In this work, we assumed a perfect time synchronization, i.e., IEEE 802.1AS is configured and properly working. This hypothesis is reasonable as we focus on configuring IEEE 802.1Qbv, which needs IEEE 802.1AS to work.

B. RL formalization

The RL relies on five major components (Figure 2): the environment and the agent on the one hand, the RL formalization (reward, state, action) on the other hand. The interactions between the agent and the environment are discrete: at each step t, the agent receives a new state and reward, and makes an action. An episode is then a sequence of interactions between the agent and the environment, that ends with a terminal state.

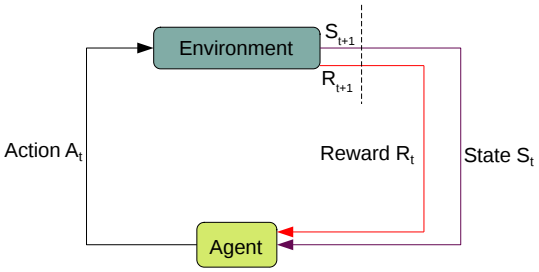

Figure 2. The interactions between the agent and the environment

1) State: The state of the environment can be seen as an image of the environment at an instant t. The agent will take decisions based on it. In our case, it includes information about the network topology, about each flow depending on its priority and about the measured end-to-end latency for critical traffics and the Qbv configuration. The difference between the state at instant t and at instant $t + 1$ is the changed Qbv configuration and the measured end-to-end latency. We define two types of terminal states: when the agent 'wins' (i.e., it solves the problem) and when the agent 'loses' (i.e., the problem can't be solved anymore). In our case, the state space (which includes all potentials states) is huge. This is because the range of the possible values (for the latencies for example) is huge, too.

2) Action: The action consists in changing the IEEE 802.1Qbv configuration. As the goal is to find an appropriate cycle length (i.e. an integer that gives the cycle duration in nanoseconds), the action consists of increasing or decreasing this parameter at each step.

3) Reward: The reward is used to evaluate the new configuration. It is determined according to the measured end-to-end latency for critical traffic as well as the ratio of received packets for each flow. We only give negative rewards to the agent, in order that it tends to quickly find a solution (as its goal is to maximize its reward over the long term).

C. Agent

The agent is the place where the learning algorithm will take place. As we have a very huge state space, we are forced to use an algorithm able to do approximation, because it is unrealistic to map each state to an action within a policy. It is showed in [14] that we can approximate a function with a neural network, which is the base idea behind DRL. Furthermore, we need to explore the state space properly; otherwise, the agent will be unable to resolve all situations. Our agent relies on the use the Soft Actor-Critic (SAC) [15] learning algorithm. SAC is an algorithm that uses neural networks to approximate an optimal policy and that will encourage the agent to explore the state space, using the entropy of the policy (i.e., it makes the agent very unpredictable during the early phases of the training).

IV. Evaluation

We present in this section the evaluation of our RL-based scheduling for TSN. We begin by explaining how we designed the training loop where the proposed RL agent will be trained, using a specific environment as depicted in Figure 3.

A. Implementation

Our implementation includes three components: the environment (is a simulation tool), the RL-agent (the Python implementation of the SAC algorithm) and the environment's interface with the RL-agent (OmnetppEnv).

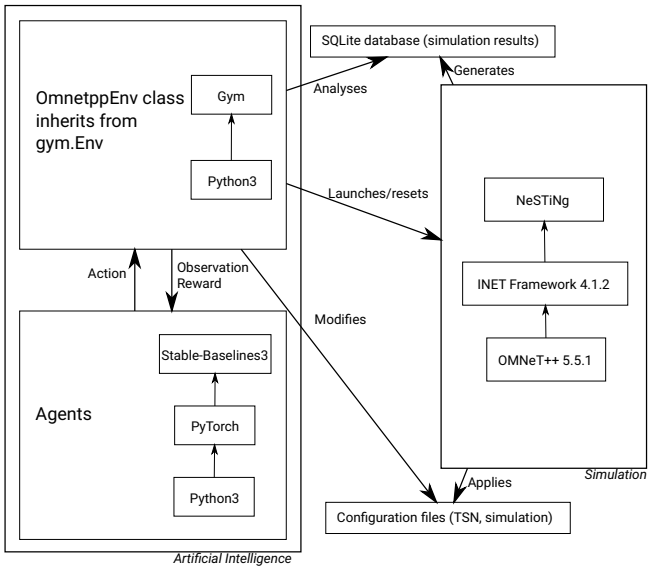

Figure 3. Architecture of the proposed solution

The environment on which the agent is trained is modeled using the network simulator OMNeT++/INET[2]and the extension NeSTiNg [16]. OMNeT++ and NeSTiNg have been recommended by the IEEE 802.1 TSN Working Group when it comes to performing TSN network simulations. The simulations are parametric. The agent can launch a simulation while providing as arguments the links' capacity, the switch number, critical traffic characteristics, and the Qbv configuration.

When a simulation ends, the agent retrieves the end-to-end latency per packet and the number of packets sent/received per flow. NeSTiNg reads the IEEE 802.1Qbv configuration through XML files. Therefore, the agent simply modifies this XML files with the schedule that it has decided. In order to accelerate the training, the simulation time is set to 0.1 second each time. In fact, OMNeT++/INET simulation does not allow parallelization.

Regarding the RL-agent implementation, we used the RL Baselines3 Zoo [17] which is build upon Stable-Baselines3 [18], a set of reliable and open source implementations of RL algorithms, written in Python.

One of the hardest tasks in RL is to implement the interactions between the agent and the environment. To achieve that, one solution is to use a library, called OpenAI Gym [19], that allows to model RL problem. We conceived and developed a module called OmnetppEnv which tasks are to interpret

[2]Available at https://omnetpp.org/ and https://inet.omnetpp.org/

the actions provided by the agent and to translate them into instructions for OMNeT++ and to translate the results of the simulation into a reward and a new state that can be interpreted by the agent.

B. Preliminary evaluation

We trained our agent over 50000 steps. At the end of the training, we tested the agent on an environment unknown by the agent. The test environment is composed of 5 switches and 4 endnodes. The Ethernet links have 1 Gbps capacity. The TSN packets have a payload size of 1000 Bytes and are sent each 500 μs. The TSN deadline is of 2.5 ms. The BE packets have a payload size of 500 bytes and are sent each 200 μs. We will verify whether the designed agent is able to find a good IEEE 802.1Qbv configuration that allow to respect the deadline imposed by TSN traffic. The evaluation is based on three observations: whether the TSN deadline has been respected, whether the proposed configuration led to a percentage of packet loss and how much time was needed by the agent to come out with a good configuration The time needed to configure the network is monitored because the objective of this study is to determine whether our solution is an interesting approach to, in the end, configure a complex industrial network.

Figure 4 shows the measured end-to-end latency for the TSN traffic during the first 1000ms of the simulation. We note that the scheduling decided by the agent respects the deadline imposed by the TSN traffic.

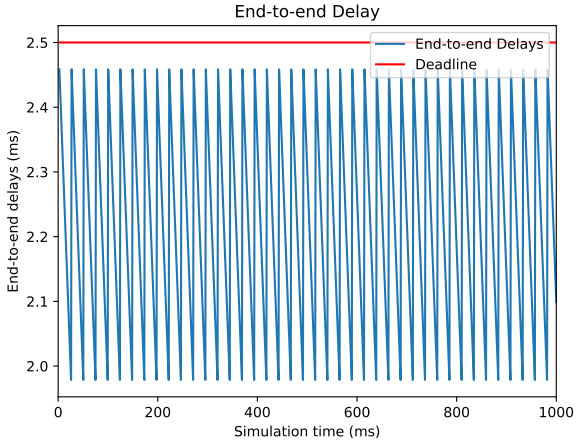

Figure 4. End-to-end delays of the TSN flow during the first 1000ms of the simulation

V. Conclusion and further work

In this article, we proposed an RL-based configuration agent for IEEE 802.1Qbv scheduling in TSN networks. The next step will consist in conducting more tests and evaluation studies on the agent. In addition, in this work, we made some assumptions to simplify the scheduling problem from the RL point of view. Those assumptions allow investigating whether the RL-agent is capable to configure Qbv in simple scenarios. In the future, we intend to relax these assumptions in order to tend toward a more realistic simulation.

References

[1] IEEE 802.1 Working Group, "IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic," *IEEE Std 802.1Qbv-2015 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, and IEEE Std 802.1Q-2014/Cor 1-2015)*, pp. 1–57, 2016.

[2] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, "Scheduling real-time communication in IEEE 802.1Qbv time sensitive networks," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, 2016, pp. 183–192.

[3] A. C. T. dos Santos, B. Schneider, and V. Nigam, "Tsnsched: Automated schedule generation for time sensitive networking," in *2019 Formal Methods in Computer Aided Design (FMCAD)*. IEEE, 2019, pp. 69–77.

[4] Z. Mammeri, "Reinforcement learning based routing in networks: Review and classification of approaches," *IEEE Access*, vol. 7, pp. 55 916–55 950, 2019.

[5] M. Ashjaei, G. Patti, M. Behnam, T. Nolte, G. Alderisi, and L. L. Bello, "Schedulability analysis of ethernet audio video bridging networks with scheduled traffic support," *Real-Time Systems*, vol. 53, no. 4, pp. 526–577, 2017.

[6] N. G. Nayak, F. Dürr, and K. Rothermel, "Time-sensitive software-defined network (tssdn) for real-time applications," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, 2016, pp. 193–202.

[7] A. Nasrallah, V. Balasubramanian, A. Thyagaturu, M. Reisslein, and H. ElBakoury, "Reconfiguration algorithms for high precision communications in time sensitive networks," in *2019 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2019, pp. 1–6.

[8] T. Stüber, L. Osswald, S. Lindner, and M. Menth, "A survey of scheduling in time-sensitive networking (tsn)," *arXiv preprint arXiv:2211.10954*, 2022.

[9] J. Prados-Garzon, T. Taleb, and M. Bagaa, "LEARNET: Reinforcement learning based flow scheduling for asynchronous deterministic networks," in *ICC 2020-2020 IEEE International Conference on Communications (ICC)*. IEEE, 2020, pp. 1–6.

[10] N. Navet, T. L. Mai, and J. Migge, "Using machine learning to speed up the design space exploration of ethernet TSN networks," University of Luxembourg, Tech. Rep., 2019.

[11] H. Yu, T. Taleb, and J. Zhang, "Deep reinforcement learning based deterministic routing and scheduling for mixed-criticality flows," *IEEE Transactions on Industrial Informatics*, 2022.

[12] X. Wang, H. Yao, T. Mai, T. Nie, L. Zhu, and Y. Liu, "Deep reinforcement learning aided no-wait flow scheduling in time-sensitive networks," in *2022 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2022, pp. 812–817.

[13] F. Dürr and N. G. Nayak, "No-wait packet scheduling for ieee time-sensitive networks (tsn)," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, 2016, pp. 203–212.

[14] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

[15] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.

[16] J. Falk, D. Hellmanns, B. Carabelli, N. Nayak, F. Dürr, and S. Kehrer, "NeSTiNg: Simulating IEEE time-sensitive networking (TSN) in OMNeT++," in *Proceedings of the 2019 International Conference on Networked Systems (NetSys)*, Garching b. München, Germany, Mar. 2019.

[17] A. Raffin, "Rl baselines3 zoo," https://github.com/DLR-RM/rl-baselines3-zoo, 2020.

[18] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[19] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv:1606.01540*, 2016.