Configuring the IEEE 802.1Q timeaAware shaper with deep reinforcement learning

Adrien Roberty, Quentin Besnard, Siwar Ben Hadj Said, Frédéric Ridouard, Henri Bauer, Annie Choquet-Geniet

Configuring the IEEE 802.1Q Time-Aware Shaper with Deep Reinforcement Learning

Adrien Roberty*, Quentin Besnard*, Siwar Ben Hadj Said*,
Frederic Ridouard†, Henri Bauer† and Annie Geniet‡
*Université Paris-Saclay, CEA, List, F-91191, Palaiseau, France
†ISAE-ENSMA - LIAS Futuroscope Cedex, France
‡Université de Poitiers - ISAE-ENSMA - LIAS Poitiers, France
Email: {adrien.roberty, quentin.besnard, siwar.benhadjsaid}@cea.fr,
{frederic.ridouard, henri.bauer}@ensma.fr, annie.geniet@univ-poitiers.fr

***Abstract*—Discussions surrounding Industry 5.0 emphasize the need for a fully interconnected industrial ecosystem, integrating AI and digital twins. In this environment, industrial equipment must collaborate seamlessly with human workers, requiring low-latency, high-data-rate connectivity for real-time monitoring. To address this demand, Time-Sensitive Networking (TSN) standards have been developed. However, configuring TSN in dynamic industrial networks poses challenges. The IEEE 802.1Q standard offers mechanisms like the Time-Aware Shaper (TAS) to achieve deterministic latency when configured correctly. In this paper, we tackle the configuration of TAS in dynamic networks, like reconfiguration of production lines to fit production objectives or the deployment of new applications in the production line resulting in adding new flows in the network. Our solution employs Deep Reinforcement Learning (DRL), trained and evaluated through simulations, offering adaptability to changing network conditions and dynamic production line reconfigurations.**

I. Introduction

The industrial landscape is undergoing a profound transformation, with Industry 4.0 and its successor, Industry 5.0, leading the way. In the context of Industry 4.0, activities such as monitoring the production line, including tasks like gathering temperature and humidity data from sensors, can reasonably accommodate relatively higher latency levels, often in the range of 50 to 100 ms. However, when it comes to real-time control of machines and robots, the performance requirements become more stringent. These tasks necessitate ultra-low latency such ad 100μs [1]. Achieving such rapid responsiveness is critical for precision and safety in industrial operations. Moreover, according to discussions on Industry 5.0, there is a strong push toward establishing a fully interconnected industrial ecosystem. This vision encompasses concepts like digital twins, human-centric communication, and the integration of artificial intelligence (AI) [2]. In this advanced environment, industrial equipment is anticipated to seamlessly collaborate with human workers, necessitating low-latency, high-data-rate connectivity for real-time monitoring.

Time Sensitive Networking (TSN) is a collection of standards that aims to include real-time features to wired Ethernet networks [3]–[5]. TSN's first advantage is its ability to guarantee high bandwidth and deterministic communications, with high synchronicity, bounded and strict latency. The second advantage is its configurable mechanisms that can handle a combination of diverse traffic constraints on the same medium. These mechanisms can efficiently adapt to a wide range of services and ensure that the network can be customized to meet the unique requirements and constraints of different applications.

TSN also provides profiles that specify the TSN standards to use and how to use them in particular application scenarios. For example, the IEC/IEEE 60802 standard project provides a TSN profile for industrial automation [1].

The main focus of this article is IEEE 802.1Q [6] standard and more precisely, on its IEEE 802.1Qbv [7] amendment (*Amendment 25: Enhancements for Scheduled Traffic*). The objective of this standard is to minimize the queuing delay in switches for critical traffic, thereby achieving a low and consistent end-to-end latency. To accomplish this, the standard proposes a time-sensitive queue draining approach that schedules frame transmission relative to a recognized timescale named Time-Aware Shaper (TAS). This mechanism involves the installation of gates in front of queues that can be opened or closed based on a configurable cycle time. This approach enables the scheduling of frame transmissions using timing derived from the IEEE 802.1AS [8] standard. Throughout the remainder of the paper, we will use the abbreviation "TAS" to refer to the scheduling mechanism outlined in the IEEE 802.1Qbv standard.

In industry 5.0 environment, the production lines can be reconfigured over time and even new application can be deployed as containers in the existing industrial networks, resulting in a dynamic network where topology and flows are varying. This poses a challenge regarding developing a new algorithm capable to react to network change by computing new TAS configuration within a limited time.

The TAS scheduling alone can lead to a well-known NP-hard problem [9]–[11]. The common approach in literature is based on engineering tools such as mathematical optimization tools like Integer Linear Programming (ILP) formulations or Satisfiability Modulo Theories (SMT) solvers [12]. However, these engineering tools are unsuitable for configuring TAS in dynamic network. First, to provide a good TAS configuration to deploy, these tools require prior knowledge of all the flows

that could be present in the network. In that sense, if a change happen in the network, the computed TAS schedule is no more effective. In case new flows are added to the network, generating news TAS schedule through these engineering tools can take several hours [13] before the new flows can be accepted/rejected, and the decided configuration can be deployed.

To surmount the challenges mentioned earlier, there arises a crucial need for an algorithm capable of swiftly determining TAS configurations. This algorithm must possess the agility to promptly respond to emerging events, such as the introduction of new data flows, alterations in network topology, or changes in flow configurations. Furthermore, it should excel in accommodating the gradual deployment of applications within the TSN network.

In this paper, we harness Reinforcement Learning (RL) methodologies to develop a new TAS scheduling algorithm capable to compute efficient TAS schedules within a reasonable time frame. In particular, this work's contributions are as follows:

- We propose an RL-based TAS scheduling that is capable to produce a TAS schedule by just observing the environment state.
- We train the proposed solution over simulations using OMNET++/INET network simulator. INET is a model library with open-source features that includes the TSN models.
- We show and compare the learning rate for three kinds of topology. We show as well the capability of the algorithm to accommodate the changing number of flows.

The remaining part of the document is organized as follows: Section II gives an overview of the current state of the art while Section III delineates the various elements of the suggested approach. In Section IV, we explain the methodology we employed for setting up the training loop and furnish the evaluation of the schedules determined by the agent. Finally, Section V concludes the paper and presents perspectives.

II. Related Work

Several studies have tackled the challenge of configuring TSN. For instance, the work done by [14] offers an analysis of the real-time traffic that traverses the TSN network, allowing for an assessment of whether time constraints are met. The conventional method of computing a deterministic schedule for TAS involves using an Integer Linear Programming (ILP) formulation [9], [15]. While this approach is efficient for small networks, it can take a significant amount of time to converge for larger networks. Additionally, these are offline techniques that are not suitable for open and reconfigurable networks. Another example is [16]. This solution, which is based on the IEEE 802.1Qcc standard [17], allows for online reconfiguration of an IEEE 802.1Qbv-based network. However, it relies on an admission control mechanism and does not modify the Qbv time cycle in the switches. The issue of online schedule reconfiguration remains an outstanding challenge in TSN [18].

The utilization of AI methods appears to hold great potential for managing networks. It has the capability to predict network congestion and make decisions regarding routing strategies [19], [20]. Nonetheless, there have been limited efforts to investigate the use of AI techniques for TSN configuration. For example, the authors of [21] suggests using RL techniques to schedule streams in 5G deterministic asynchronous networks. The proposed solution exclusively configures the Asynchronous Traffic Shaper (ATS) mechanism described in the IEEE 802.1Qcr amendment to the IEEE 802.1Q standard. In [13], the authors employ AI methods to determine the feasibility of a potential configuration, i.e. whether it satisfies the application requirements. To accomplish this, they experiment with simple supervised and unsupervised learning algorithms to classify possible configurations as feasible or non-feasible. The primary disadvantage of this solution is that the AI is only effective on a specific topology. When the topology changes, their AI necessitates retraining. Moreover, the proposed solution is unsuitable for online configuration.

A study by [22] has employed DRL to manage the routing and scheduling of mixed-criticality traffic within a Deterministic Networking (DetNet) framework, which operates at layer 3. On the other hand, TSN operates at layer 2. Another research by [23] has utilized DRL to support their TAS scheduling algorithm. Nevertheless, their approach adopts the no-wait model for TSN scheduling, which was initially introduced in [24]. This approach involves scheduling being carried out in the clients, necessitating a prior understanding of each flow.

The field of Reinforcement Learning is vast. A glimpse of it can be found in [25]. To help choose the appropriate algorithm, RL algorithms can be categorized. Firstly, the algorithms are classified as model-based (where the agent can predict the outcome of each action) or model-free. At first glance, since we cannot predict the environment's development, we will use model-free algorithms. Secondly, we have to decide how the agent will learn, either on-policy (where the algorithm assesses and enhances the policy employed for selecting actions) or off-policy (where the algorithm assesses and enhances a policy that differs from the one used to select actions). We have two algorithms that catch our attention.

III. RL-based TAS scheduling

In this section, we describe our RL environment and the assumptions made. We also present the RL formulation that allows to learn and decide TAS scheduling.

A. Network model

The TAS mechanism shares a resemblance to the Time-Division Multiple Access (TDMA) technique. The transmission time is sliced into cycles of unchanging duration, which are further divided into time sequences of variable lengths. Each sequence is then allocated to a specific traffic class. In Figure 1, the TAS mechanism is illustrated within a TSN switch. The standard assigns a queue for each flow priority and a logical gate controlling tranmission over each queue. These gates can either be open (allowing frames in the associated

queue to be transmitted) or closed (preventing frames in the associated queue from being transmitted). A Gate Control List (GCL) manages these gates, defining which gates are open and for how long at each instant. When multiple queues' gates are opened simultaneously, the priority determines the frame transmission order. The standard outlines 8 different priorities, starting from 7 (highest) to 0 (lowest), requiring the presence of eight queues.

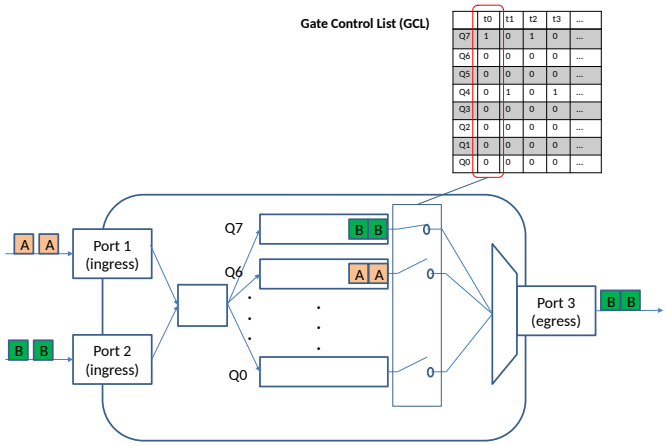

Figure 1. The TAS scheduling mechanism

The TAS operation is illustrated in Figure 2 through a simple network with two TSN switches, two talkers and two listeners. We assume two types of flows: critical flow with high priority and 'normal' flow, also known as Best Effort (BE) flow, with low priority.The critical flow is a periodic flow that requires deterministic ultra-low latency with hard deadlines and low jitter without packet loss. To guarantee these requirements, TAS allows to dedicate one time slot for the critical flow and therefore protects this flow from any interference coming from other traffic types. The TAS cycle can be divided into two time sequences. The first sequence is reserved for critical flow, while the second sequence is for all other flows.

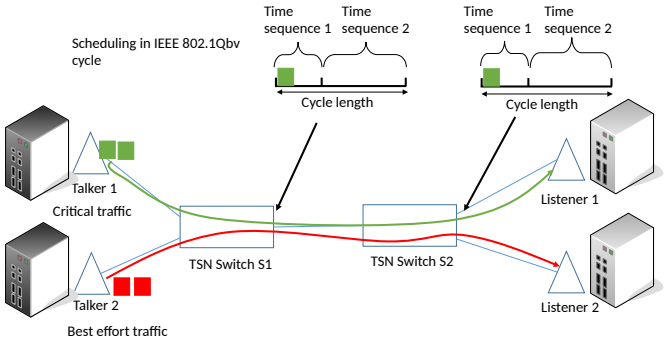

Figure 2. An illustrative example of two talkers (Talker 1 for critical traffic and Talker 2 for BE traffic) exchanging with two listeners (Listener 1 for critical traffic and Listener 2 for BE traffic) via a linear TSN network composed of two TSN switches

For the scope of this study and in order to simplify our problem, we have made the following assumption about the network:

1) We are assuming a precise time synchronization in network according to the IEEE 802.1AS standard. This assumption is justified as our paper primarily concentrates on configuring the TAS, which in turn depends on the accurate time synchronization ensured by gPTP.
2) In Ethernet TSN networks, the end-to-end delay comprises transmission delay (i.e. time it takes for data to pass through any network interfaces), propagation delay over links and processing and queuing within TSN switches. Here, we assume uniform processing delays in all switches and identical link capacities, simplifying our RL formulation. Our algorithm's focus is to determine schedules that minimize queuing delay in switches for critical traffic, thus reducing overall end-to-end delay.
3) we consider a network with a constant number of switches.It's worth noting that switch numbers in a factory setting are typically not subject to frequent changes. Consequently, this fixed setup does not pose a significant concern for our study.

In this paper, we consider three kinds of network topology: linear, ring and mesh, illustrated in Figure 3. This diversity is crucial, as it equips the RL agent with the ability to decide TAS configuration in any network topology.

B. Traffic model

We consider periodic traffic for critical traffic and sporadic exponential traffic for the BE traffic. The critical (TSN) flows adhere to a uniform distribution within the range of $[0\mu s, 400us]$. Conversely, all Best Effort (BE) flows are assumed to follow exponential distribution with a parameter of $400us$. To train over varied network configurations (i.e. varying the number of flows in the network), we employ several distributed clients that generates critical and BE traffic. Each critical flow has a deadline (i.e. the maximum accepted end-to-end latency) that should be respected. The agent's goal will then be to find a TAS schedule that will allow the critical flow's latency to be below this deadline.

C. RL formulation

As illustrated in Figure 4, the RL problem can be described as a Markov decision process (MDP), characterized by a state space denoted as S and an action space denoted as A. It is thus defined by the corresponding tuple (S, A) representing all possible network states and the corresponding set of actions. In addition, RL problem includes a reward function denoted as $R(S, A)$ which, through some observation metrics in the state, evaluates the quality of the actions taken by the agent.

The learning process of our RL takes the form of a state S on which the agent apply an action. The agent is rewarded based on the achieved end-to-end latency of the critical flow. This sequence of actions is then repeated until the agent reaches the TAS configuration that allows to achieve, for the critical flow, an end-to-end latency below the deadline. The main challenge is to define the state, action, and reward of RL

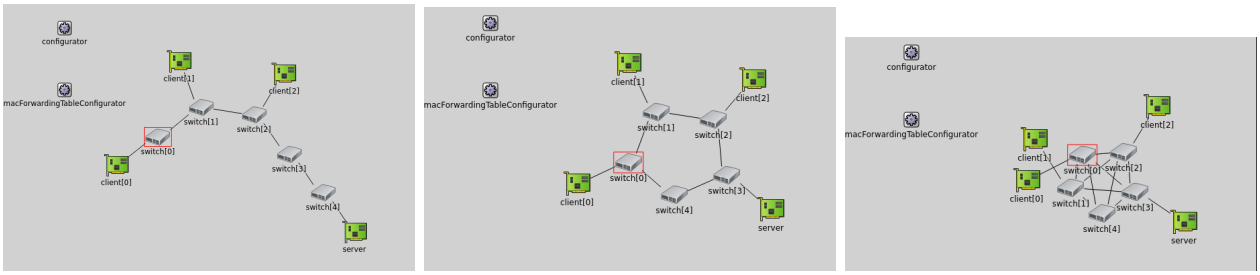

Figure 3. Network topology considered during training phase. The switch number stays the same, links and clients number may vary

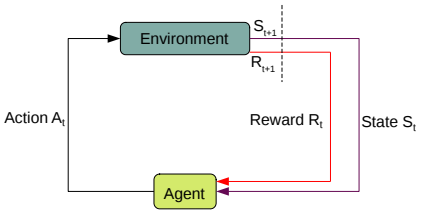

Figure 4. The interactions between the agent and the environment

that achieves our goal. The training process consists of a set of learning episodes (i.e. a sequence of interactions between the agent and the environment until a terminal state). Each learning episode has a different length, i.e., an episode requires a different number of steps before reaching at a terminal state.

1) State: The state of the model includes the entire TAS configuration parameters (i.e. the TAS cycle duration, TSN time slot duration, TSN time slot position within the cycle, BE time slot duration) as well as the TSN flow latency parameter. The agent moves from one state to another by proposing, each time, a new TAS configuration for the network that has not been explored/tested. The agent measure the quality of each transition (i.e. from a state $S_(t)$ to $S_(t+1)$) through an observation of the environment and more precisely the end-to-end critical flow latency. The agent can transition from a state with a suboptimal TAS configuration to a state with an optimal TAS configuration in just one time step, and vice versa. The challenge lies in the ability of the model to learn what is the optimal configuration for its environment. To overcome this challenge, we set up the following rules to help the agent:

- The measured end-to-end latency of the TSN flow in the current state is below the deadline (i.e. the maximum end-to-end accepted latency by the TSN flow), the reward is therefore positive, and it represents the end of the episode;
- The measured end-to-end latency of the TSN flow in the current state is greater than the deadline. The agent has two options:
  1) The time step number is below the configured maximum time steps per episode: the agent tries to explore other possibilities;
  2) The time step number is equal to the configured maximum time steps per episode (i.e. means the end of the episode): a negative reward is given, the agent updates the state and moves on to a new episode.

By doing this, we give the agent the possibility of exploring a set of configurations within each episode, but these are deliberately short to accelerate the learning. The agent is updated regularly until a high-performance configuration subspace is discovered where the agent can converge towards an optimal configuration.

2) Action: The action space consists in modifying the TAS configuration parameters. To simplify the system, we make the assumption of having only two traffic classes, namely critical and BE and therefore considers only two queues and gates. This reduction in the number of traffic classes helps streamline the action space. Hence, when configuring TAS on a single egress port of the TSN switch, the process involves setting the following parameters for each gate:

- d_{cycle}: represents the duration of the TAS cycle.
- d_{offset}: represents the duration of the offset from the beginning of the cycle till the instant of opening the gate.
- d_{open}: represents the duration of the opening of the gate.

The Figure 5 shows graphically the essential TAS configuration parameters that the agent has to configure when we consider only two gates in an egress port.

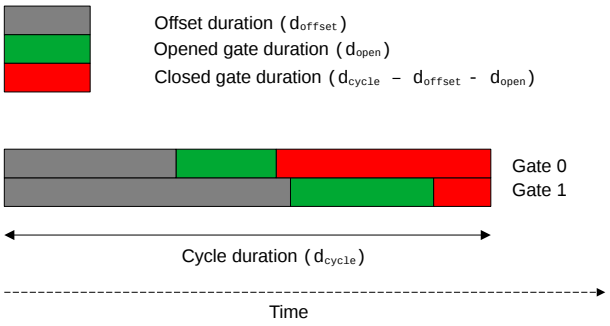

Figure 5. Essential TAS configuration parameters

3) Reward: The reward is used to evaluate the new schedule decided by the agent. We used the following formula:

$$reward = \frac{deadline - new\_latency}{10} - penalty$$

Where $new\_latency$ represents the current measured end-to-end delay of the critical flow and the $deadline$ represents

a limit beyond which the $new\_latency$ is considered unacceptable for the critical flow. Both parameters are given in milliseconds. As long as the new latency is below the deadline, the reward is negative. This allows to encourage the agent to find faster an acceptable solution, and then to improve its decisions. The $penalty$ variable is important to punish the agent when it generates an error during the simulation due to too extreme values that has been decided by the agent, or to accentuate the fact that the wasn't able to decide TAS configurations allowing to achieve good latency. If the latency obtained with the decided TAS configuration is below this deadline, then the episode comes to an end.

D. Agent

The agent is the brain of our learning environment. It is governed by a learning algorithm such as Proximal Policy Optimization (PPO), depending on the problem (i.e., how the environment is modelled, as each algorithm can't be applied to each environment). PPO is part of the family of deep reinforcement learning (DRL) algorithms. These intersect reinforcement learning (RL) and deep learning (DL) to allow the exploration of results on very large and unstructured data spaces. PPO uses policy-based learning to learn practical work through the evaluation of existing policies and evaluates the environment. It supports exploration using the entropy method. A low entropy forecast has a strong confidence in choosing an event, while a high entropy forecast is uncertain about which event to choose. PPO offers better acceptance rates and efficiency than other systems but is sensitive to change. For this reason and also because of its ability to manipulate discrete action spaces and a continuous observation space, we decided to base the agent learning process on the PPO algorithm.

IV. Evaluation

In this part, we present the setup and then provide the results of the training of our RL-based TAS scheduling solution.

A. Setup

The training setup is divided into 3 parts as shown on Figure 6:

1) The RL agent (governed by PPO);
2) The network simulator OMNeT++/INET;
3) The intermediate environment allowing the connection between the agent and simulator.

The RL agent uses the Stable-Baselines3 [26] library for the implementation of the PPO algorithm, and the RL Baselines3 Zoo [27] training framework. Stable-Baselines3 relies on PyTorch [28] for the neural networks. The agent is connected to the environment via the Gymnasium API [29]. This allows the agent to be brought into contact with the simulator by implementing a set of interactions for decision-making and observation retrieval.

The simulator where the agent will undergo training is based on the OMNeT++ network simulator and its extensions INET[1].

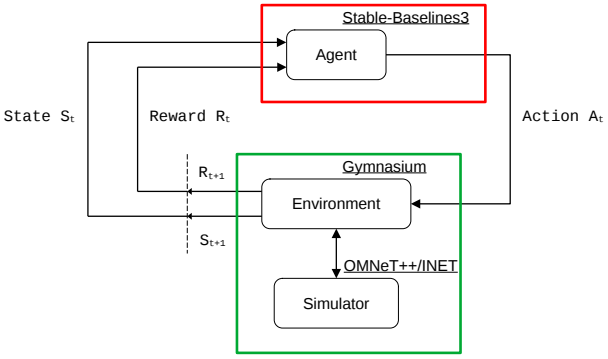

Figure 6. Training setup

The IEEE 802.1 TSN Working Group has recommended the use of OMNeT++ for carrying out TSN network simulations.

Upon completion of the simulation, the results are stored in SQLite format. The agent can retrieve the end-to-end latency per packet and the number of packets sent/received per flow from the obtained results. Additionally, the SQLite database contains a vector of TAS gate states, the number of packets waiting in each queue, and other relevant information that aids in evaluating the agent's schedule. These features enable the management of the environment from a Python script, including configuration modification, simulation initiation, and result analysis.

To speed up the training process, the simulation duration has been reduced to 1 seconds per iteration. Even so, each simulation needs on average 20 seconds to execute including the time required by OMNET++ to generate the required files to run the simulation, which is long and can lead to a long training time.

B. Experimentation

Figure 7 shows the learning rate of our RL agent for the three types of network topology (i.e. linear, ring and mesh) with 5 switches and 3 clients. We note the learning rate of the agent applied to a mesh network is better than when the network topology is linear or ring. The case of the linear network presents the worst case where the flow must then meet all of the switches in the network, thus finding itself significantly slowed down.

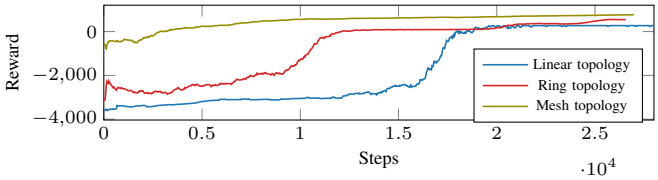

Figure 7. Training over three types of network topology (mesh, ring and linear)

Then, to tend toward a more complex network, we set up 2 new experiments, which consists of:

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

1) varying the number of clients during training on a classic linear network with 5 switches in order to observe the model's behavior while varying the number of clients between 1 and 5 after each episode;
2) varying the topology with a fixed number of clients (3 clients) in order to observe the reaction of the model to these major changes;

The Figure 8 shows the results.

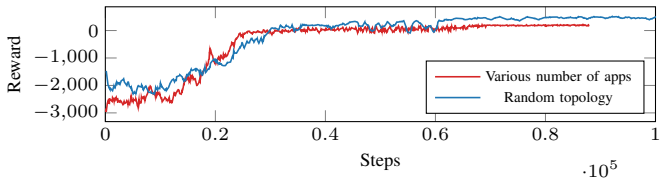

Figure 8. Training on random number of clients (1 to 5) for each episode with linear topology and random topology (mesh, ring and linear) for each episode with 3 clients

The first observation is obviously the drastic increase in training time compared to training on a simple topology (Figure 7). Beyond an increase in the necessary learning time, we see that the model always manages to find convergence towards interesting configurations. There is still a limit in this convergence because it stagnates towards an average configuration suitable for different topology without being excellent in each case. The new configurations produced are much better than a randomly generated configuration, but they are less efficient than a configuration generated with a model trained only on the dedicated topology.

To push further the complexity of a single-agent model to generate configurations, we set up an experiment representing the fusion of the two previous experiments (Figure 8) with a modification of the reward in order to aim for the minimization of a negative reward. This is based on a]-∞, 0] space and behaves to tend to 0 when the configuration minimizes network latency. The goal of this experiment is to observe the level of optimization that a model can achieve with a configuration on different topology and with a variable number of clients. The result is shown on Figure 9, which depicts the learning behavior in a very unstable environment.

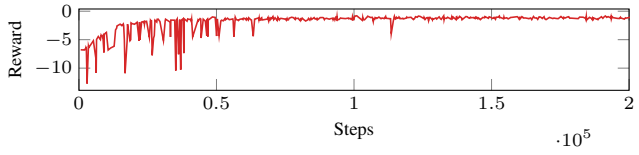

Figure 9. Cross-training between random topology (mesh, ring and linear) and random clients (1 to 5)

Despite being time-consuming to train, we can notice a nice improvement in the reward which converges towards its maximum achievable as defined in the system. As we can see, learning takes place in a consecutive step of search then stabilization until finding a new set of parameters to study in more detail. We can identify an initial improvement with parameters chosen more effectively than by luck and then stabilize the latter with a slight improvement over time. Subsequently, when discovering a new set of usable parameters, it repeats this steps until reaching a maximum which represents the optimal subset of parameters. Until the reward is completely stabilized, the learning process is not yet complete. We can also observe, through the convergence, the robustness of the model when it faces difficult environments like ours. This suggests many possibilities regarding its ability to learn on much larger environments with notably many more clients. During this last experiment, we also recorded the average episode length (Figure 10). This shows how much try (on average) the agent needs before finding a solution. We can see that our model pushes the agent to find quicker an acceptable solution. In the end of the training, the agent needs on average 2 tries before finding a solution. If we consider that a simulation lasts around 20 seconds, this means that the agent will need less than a minute to find a solution.

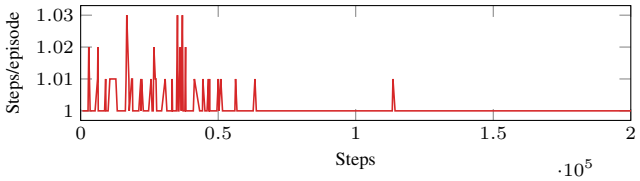

Figure 10. Average episode length during the last training

V. Conclusion and Further Work

In this paper, we introduce an RL-based approach for configuring TAS schedules in TSN networks. Our assessments have demonstrated that the agent we designed can provide an acceptable configuration within a reasonable timeframe, showcasing the potential of RL to create autonomous network configuration solutions for TSN networks without the need for prior knowledge of flow patterns.

One of the constraints brought by our single-agent is the need to know beforehand the topology and the number of switches within the network due to the rigidity of the action space. Because of this, changing the core of the network dynamically during the training is impossible. This means that the core of the production network must share a strong resemblance to the topology used for training the model. To solve this issue, one idea would be to port our current single-agent project to a multi-agent reinforcement learning (MARL) version in which each switch would correspond to an autonomous agent working on the optimization of the overall network. Porting to a multi-agent version would help to overcome this dependence on the network topology to concentrate solely on the density of communications such as the number of clients or even network speeds. Thus, the ideal of the model could adapt without topology concerns to other application networks and concentrate the optimization of the scheduler on the flows and capacities of the sub-networks.

REFERENCES

[1] IEC/IEEE, "Iec/ieee 60802 tsn profile for industrial automation - draft 2.1," https://www.ieee802.org/1/files/private/60802-drafts/d1/60802-d1-2.pdf, 2020.

[2] X. Xu, Y. Lu, B. Vogel-Heuser, and L. Wang, "Industry 4.0 and industry 5.0—inception, conception and perception," *Journal of Manufacturing Systems*, vol. 61, pp. 530–535, 2021.

[3] J. Farkas, L. L. Bello, and C. Gunther, "Time-sensitive networking standards," *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 20–21, 2018.

[4] N. Finn, "Introduction to time-sensitive networking," *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 22–28, 2018.

[5] J. L. Messenger, "Time-sensitive networking: An introduction," *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 29–33, 2018.

[6] IEEE 802.1 Working Group, "IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks," *IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014)*, pp. 1–1993, 2018.

[7] ——, "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.

[8] ——, "IEEE Standard for Local and Metropolitan Area Networks–Timing and Synchronization for Time-Sensitive Applications," *IEEE Std 802.1AS-2020 (Revision of IEEE Std 802.1AS-2011)*, pp. 1–421, 2020.

[9] 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.

[10] K. W. Tindell, A. Burns, and A. J. Wellings, "Allocating hard real-time tasks: an np-hard problem made easy," *Real-Time Systems*, vol. 4, no. 2, pp. 145–165, 1992.

[11] J. Y.-T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Performance evaluation*, vol. 2, no. 4, pp. 237–250, 1982.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] IEEE 802.1 Working Group, "IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks – Amendment 31: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements," *IEEE Std 802.1Qcc-2018 (Amendment to IEEE Std 802.1Q-2018 as amended by IEEE Std 802.1Qcp-2018)*, pp. 1–208, 2018.

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

[19] V. Sivakumar, O. Delalleau, T. Rocktäschel, A. H. Miller, H. Küttler, N. Nardelli, M. Rabbat, J. Pineau, and S. Riedel, "Mvfst-rl: An asynchronous rl framework for congestion control with delayed actions," *arXiv preprint arXiv:1910.04054*, 2019.

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

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[26] 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

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

[28] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035.

[29] M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. d. Cola, T. Deleu, M. Goulão, A. Kallinteris, A. KG, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, and O. G. Younis, "Gymnasium," Mar. 2023. [Online]. Available: https://zenodo.org/record/8127025