Configuring the IEEE 802.1Qbv Time-aware shaper with deep reinforcement learning

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

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

ADRIEN ROBERTY, University Paris-Saclay, CEA, List, France and ISAE-ENSMA - LIAS, France
SIWAR BEN HADJ SAID, University Paris-Saclay, CEA, List, France
FREDERIC RIDOUARD, ISAE-ENSMA - LIAS, France
HENRI BAUER, ISAE-ENSMA - LIAS, France
ANNIE GENIET, University of Poitiers - ISAE-ENSMA - LIAS, France

One of the breaking changes induced by Industry 4.0 will be the networking of production equipment. To achieve this, the Time-Sensitive Networking (TSN) set of network standards has been developed. However, this new networking paradigm will create new challenges. For example, TSN standards allow a certain level of flexibility and modularity in the data plane, however, the configuration of these standards depends on many parameters (e.g., network topology, routing strategy, critical flows requirements, etc.) making the configuration task cumbersome. The IEEE 802.1Qbv standard is among the main TSN standards that propose a mechanism allowing to achieve deterministic latency when it is appropriately configured. Today's main approach to configure this mechanism relies on exact or heuristic methods. These are adequate for closed network (when all flows are known beforehand and the network topology is fixed). However, in open networks (where flows are added to the network in an incremental way and the network topology is dynamic), the scheduling in IEEE 802.1Qbv can lead to a NP-hard problem. In this paper, we address open networks such as TSN in industrial networks with reconfigurable production lines. We propose a solution to configure the IEEE 802.1Qbv standard by using Deep Reinforcement Learning (DRL). We use simulations to train and evaluate the configuration agent.

CCS Concepts: • **General and reference** → Experimentation; • **Networks** → *Network simulations*; **Network management**; *Wired access networks*; • **Computing methodologies** → **Planning and scheduling**; **Reinforcement learning**.

Additional Key Words and Phrases: Time Sensitive Networking, Industry 4.0, Deep Reinforcement Learning

ACM Reference Format:
Adrien Roberty, Siwar Ben Hadj Said, Frederic Ridouard, Henri Bauer, and Annie Geniet. 2018. Configuring the IEEE 802.1Qbv Time-Aware Shaper with Deep Reinforcement Learning. In . ACM, New York, NY, USA, 15 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

One of the primary aspects of Industry 4.0 is the interconnection of production equipment, which includes machines, production lines, robots, and storage and conveying systems. This will enable production equipment to control, configure, and share information, but places high demands on communication devices in terms of reliability, latency, and longevity. The most crucial objective for Industry 4.0 from a network perspective is the real-time performance of communications, or the addition of quality of service to the network.

CoNEXT 2023, December 5-8, 2023, Paris, France

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

Time Sensitive Networking (TSN) is a collection of standards that aims to include real-time features to wired Ethernet networks [7, 8, 17]. 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 on the IEEE 802.1Qbv [11] amendment to the IEEE 802.1Q [12] standard (*Amendment 25: Enhancements for Scheduled Traffic*). The objective of this standard is to minimize the queuing delay in switches for important cyclic 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. The 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 [14] standard.

In the anticipated scenarios of industry 4.0, the production lines can be reconfigured, resulting in a dynamic network topology and flow. This poses a challenge for configuring TSN mechanisms. One of the primary issues with TSN standards pertains to configuring TSN networks. The scheduling in IEEE 802.1Qbv alone can lead to a well-known NP-hard problem [3, 15, 30]. The prevalent approach in literature is based on engineering tools such as simulation tools like RTaW Pegase[2]or mathematical optimization tools like Integer Linear Programming (ILP) formulations or Satisfiability Modulo Theories (SMT) solvers [4]. However, these engineering tools are unsuitable for the configuration of IEEE 802.1Qbv. To provide a good Qbv configuration to deploy, these tools require prior knowledge of all the flows that could be present in the network. Even if these tools are used for each new flow, it would result in latency in the configuration decision and deployment process, because the configuration process involves retrieving the list of existing flows in the network, the topology, and the characteristics of the new flow. This information is then provided to the engineering tools to determine the new Qbv configuration to deploy. This process can take several hours before the flow can be accepted/rejected, and the decided configuration can be deployed. Therefore, existing engineering tools are better suited for closed networks (where the flows are known in advance, and no new flows are expected during network operation) and for offline use (before deploying the network).

In order to overcome the aforementioned obstacles, there is a need for a rapid and efficient scheduling algorithm that can determine the IEEE 802.1Qbv configuration. This algorithm must possess the ability to promptly respond to any new occurrence (such as the emergence of a new flow, changes in topology, or alterations in flow configuration) by selecting the appropriate scheduling to be implemented throughout the network. Additionally, the algorithm must be capable of accommodating the gradual implementation of application flows within the TSN network, where not all flows are predetermined. Furthermore, unlike conventional analytical optimization tools, the algorithm must be able to execute quickly (within seconds). As a result, we have employed Reinforcement Learning (RL) methodologies to devise a TSN scheduling algorithm.

[1]Available at https://1.ieee802.org/tsn/iec-ieee-60802/

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

This article presents a solution for generating an IEEE 802.1Qbv setup. Our proposal relies on RL methods, which are ideal for decision-making. RL is frequently employed for routing in computer networks [16]. We aim to demonstrate that an RL agent can configure the scheduling of the IEEE 802.1Qbv within a reasonable timeframe. To accomplish this objective, we utilize simulations to train and assess the agent. The simulations are conducted using the OMNeT++/INET network simulator. INET is a model library with open-source features (that includes the TSN models) for the OMNeT++ simulation environment.

The remaining part of the document is organized in the subsequent manner: Section 2 gives an overview of the current state of the art while Section 3 delineates the various elements of the suggested approach. In Section 4, 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 5 wraps up the article and presents future prospects.

2 RELATED WORK

Several studies have tackled the challenge of configuring TSN. For instance, the work done by [1] 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 IEEE 802.1Qbv involves using an Integer Linear Programming (ILP) formulation [3, 21]. 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 [19]. This solution, which is based on the IEEE 802.1Qcc standard [13], 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 [28].

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 [16, 27]. Nonetheless, there have been limited efforts to investigate the use of AI techniques for TSN configuration. For example, the authors of [23] 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 standard. In [20], 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 [32] 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 [31] has utilized DRL to support their IEEE 802.1Qbv scheduling algorithm. Nevertheless, their approach adopts the no-wait model for TSN scheduling, which was initially introduced in [5]. 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 [29]. 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. 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). Proximal Policy Optimization [26] is a well-known on-policy model-free algorithm, whereas Deep Q-learning [18] is a well-known off-policy model-free algorithm. Our agent employs the Soft Actor-Critic (SAC) [9] learning algorithm (refer to subsection 3.3), which aims to balance the strengths and weaknesses of both algorithm families.

3 RL COMPONENTS FOR SCHEDULING IEEE 802.1QBV

The aim of this article is to suggest a solution based on RL algorithm for making decisions regarding scheduling in IEEE 802.1Qbv. In this section, we describe the environment taken into account and the assumptions made to simplify the training process. We also introduce the formalization of RL that will enable the configuration of IEEE 802.1Qbv.

3.1 Network modelization

The arrangement of the network can be delineated by a directed graph $G = (V, E)$, where V is a collection of vertices made up of end points (such as sensors, actuators, PLC controllers, and so on) and TSN switches. E is a set of edges that includes full-duplex connections linking all components of the network. A model of network topology is depicted in Figure 1. All topologies are configured to ensure that the IEEE 802.1Qbv is uniform on all switches.

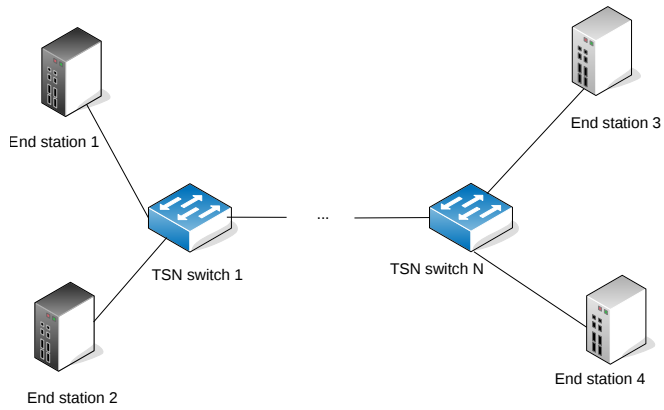

Fig. 1. General topology of the considered networks (where N can be equal to 1)

The IEEE 802.1Qbv 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 2, the Qbv mechanism is illustrated within a TSN switch. The standard assigns a queue for every flow priority and a logical gate before 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 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 eight different priorities, starting from seven (highest) to zero (lowest), requiring the presence of eight queues.

The network's TSN functionality is illustrated in Figure 3. The figure depicts two types of traffic: critical traffic and 'normal' traffic, also known as Best Effort (BE) traffic, with low priority. This

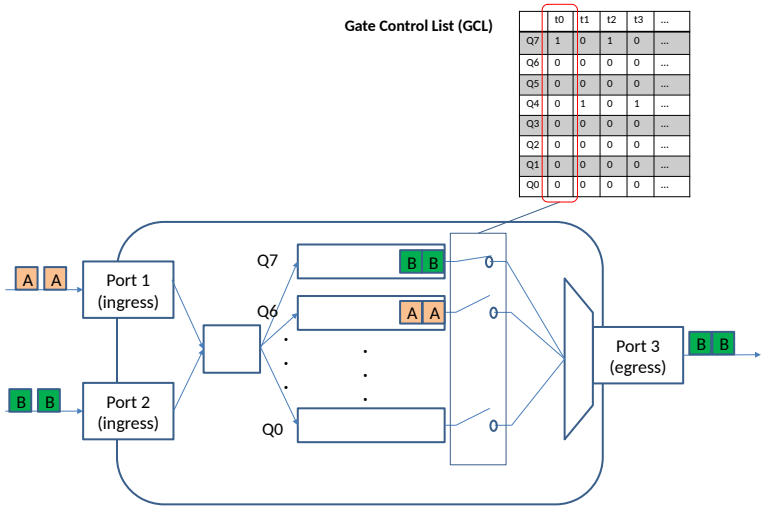

Fig. 2. IEEE 802.1Qbv scheduling mechanism

simplifies the scheduling of the streams, resulting in only two time sequences. The first sequence is reserved for critical traffic, while the second sequence is for all other traffic. We assume that the talkers and listeners at the end stations are stationary. Two talkers and two listeners were considered:

(1) a talker that generates packets for critical traffic;
(2) a talker that generates packets for BE traffic;
(3) a listener that receives packets from critical traffic;
(4) a listener that receives packets from BE traffic.

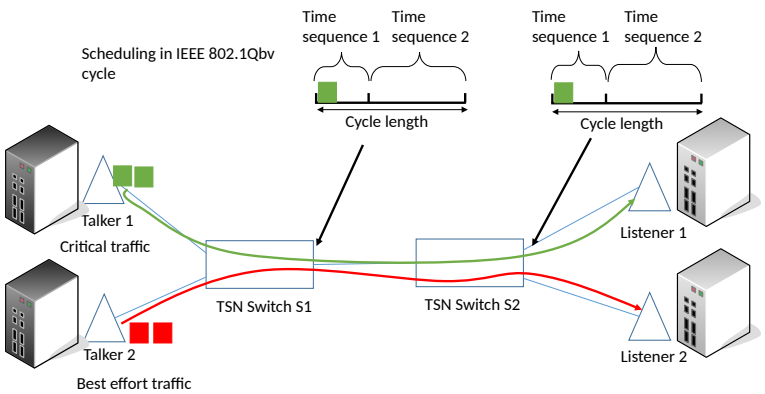

Fig. 3. An illustrative example of two talkers (Talker 1 for critical traffic and Talker 2 for BE traffic) exchanging with two listeners (Listener 1 for critical traffic and Listener 2 for BE traffic) via a TSN network composed of two TSN switches

For successful reinforcement learning training, it is essential that the setting changes between each occurrence. In our instance, the number of links (i.e. the number of jumps) that connect talkers

to listeners and the theoretical capacity of connections fluctuate. The size of the TSN payload is arbitrarily chosen in every occurrence. Additionally, the TSN period differs from one occurrence to the next. Moreover, a new latency deadline is randomly calculated at the commencement of every occurrence. As outlined in the draft of the TSN Profile for Industrial Automation, version 1.2[3], Table 1 lists the primary environment parameters and their corresponding range of values.

Table 1. Environment parameters

Parameter	Value
Number of switches	1 - 10
Links' capacity	100 or 1000 Mbps
Payload size	30 - 1500 B
Traffic period	240 - 1000 μs
Maximum latency of critical traffic	100 μs - 20 ms

To assess if the IEEE 802.1Qbv timetable guarantees deterministic communication, two crucial guidelines must be followed for every critical stream:

(1) the ultimate permissible end-to-end delay (i.e., deadline) should not be surpassed;
(2) the fluctuation in end-to-end delay among packets of the identical stream (i.e., jitter) should approach zero. This assures that the delay is foreseeable and definite.

In Ethernet TSN networks, the end-to-end delay is composed of the time it takes for data to pass through network interfaces of end stations and switches, the delay caused by data transmission over network links, and the delay caused by processing and queuing in TSN switches. It is assumed that all switches have the same processing time and all links have the same capacity, resulting in the same transmission and propagation delays. Therefore, a well-designed IEEE 802.1Qbv schedule will minimize queueing delay in switches for critical traffic, ultimately reducing the end-to-end delay.

To ensure that critical traffic deadlines are respected, the time sequence reserved for such traffic should be twice the duration of one frame emission. This means that determining the appropriate cycle duration (in nanoseconds) is a crucial aspect of configuring IEEE 802.1Qbv.

For the purposes of this study, perfect time synchronization is assumed, meaning that IEEE 802.1AS is configured and functioning properly. This assumption is reasonable as the focus of this paper is on configuring IEEE 802.1Qbv, which relies on the proper functioning of IEEE 802.1AS.

3.2 RL formulation

The RL depends on five main components (Figure 4): the environment and the agent, along with the RL formulation (reward, state, action). The scheduling issue of IEEE 802.1Qbv can be depicted as a Markov Decision Process (MDP), which is characterized by a tuple (S, A, R), where S and A signify state and action spaces, and R represents the reward by performing action $a \in A$ at state $s \in S$. In the following, we elaborate on the states, actions, rewards, and the RL algorithm that we have utilized while designing our agent. The agent-environment interactions are discrete: at each timestep t, the agent receives a new state and a reward, and then takes an action. An episode is a sequence of interactions between the agent and the environment that terminates with a terminal state. Each episode has a different length, i.e., an episode requires a different number of timesteps before arriving at a terminal state.

[3]Available at https://1.ieee802.org/tsn/iec-ieee-60802/

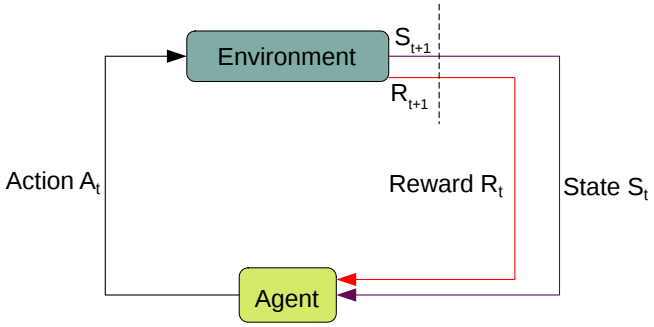

Fig. 4. The interactions between the agent and the environment

3.2.1 State. The state of the environment can be viewed as a snapshot of the environment at a particular moment. The decision-making process of the agent is based on this snapshot. It comprises data pertaining to the network topology, flow specifics based on priority, and end-to-end latency measurements for critical traffic. The state also includes the Qbv configuration. The Qbv configuration and measured end-to-end latency distinguish the state at timestep t from that at timestep $t + 1$. Two kinds of terminal states are identified:

- the agent succeeds, implying that the problem is solved and a Qbv configuration that allow latencies to respect the deadline is found;
- the agent fails, implying that the problem is unsolvable. This may happen in certain cases:
  (1) the agent tries to explore an invalid area of the state space (after more than 100 attempts, we can assume that the agent is lost);
  (2) the new state lies outside the state space.
  In such cases, we we consider that the agent reached a terminal state (otherwise, the training would take an excessive amount of time), and the agent receives a very bad reward.

In our case, the state space (which covers all potential states) is huge. This is due to the vast range of possible values (such as latencies).

3.2.2 Action. The action consists in modifying the IEEE 802.1Qbv configuration. In this work, we focus on one parameter, the Qbv cycle length. This parameter is essential to the Qbv configuration as it is needed before computing the scheduling. The objective of the agent is to determine an appropriate cycle duration. The action therefore consists in adjusting this parameter by either increasing or decreasing it in each iteration.

3.2.3 Reward. The reward is utilized to assess the new Qbv configuration decided by the agent. It is established based on the measured end-to-end delay for critical traffic. It is formulated to enable the agent to find a solution quickly. To achieve this, the agent is only provided with non-positive rewards, which encourages it to accelerate the process (as the agent's primary objective is to maximize its long-term reward), except when the agent finds the solution. The agent gets:

- a very bad negative reward if it gets into a terminal state that doesn't solve the problem;
- a negative reward if the new state is worse than the previous one;
- a neutral reward if the new state is better than the previous one;
- a positive reward if it gets into a terminal state that solves the problem

The determination of the reward depends on the critical flow's latency and the percentage of received packets for each flow.

3.3 Agent

The agent is where the learning algorithm will occur. Because of a huge state space, it is impractical to map every state to an action in a policy, so we must use an approximation algorithm. As demonstrated in [10], we can utilize a neural network to approximate a function, which is the fundamental notion of DRL. Furthermore, we must adequately explore the state space; otherwise, the agent will be unable to resolve all problems. Among the learning algorithms used in RL, we consider that Soft Actor Critic (SAC) is the most adequate algorithm for our problem. In fact, SAC is an algorithm that employs neural networks to estimate an optimal policy and encourages exploration of the state space by using the policy's entropy (i.e., making the venue unpredictable during early training stages). The SAC algorithm relies on various parameters, the majority of which are linked to the neural networks. At present, we have used the default values as specified in [9].

4 EVALUATION

In this part, we provide an assessment of our TSN scheduling approach based on RL. We also explain our implementation as shown on Figure 5. The training loop is made up of the following three components:

(1) an environment, which is based on the use of a simulation tool allowing to model and simulate TSN networks;
(2) an RL agent, which is a python implementation of the SAC algorithm, utilizing the RL Baselines3 Zoo [24] training framework;
(3) the environment's interface, named OmnetppEnv, utilizing the OpenAI Gym API [2], that facilitates interactions between the RL agent and the environment, and defines the MDP.

4.1 Environment

The environment where the agent will undergo training is based on the OMNeT++[4]network simulator and its extensions INET and NeSTiNg [6]. The IEEE 802.1 TSN Working Group has recommended the use of OMNeT++ and NeSTiNg for carrying out TSN network simulations.

The OMNeT++ simulation can be launched via command line by providing specific parameters as arguments. This feature facilitates the creation of a parametric simulation. The agent can launch the simulation while specifying various parameters, such as link capacity, switch number, critical traffic characteristics, and Qbv configuration.

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. These data are crucial for calculating the reward and the new state. Additionally, the SQLite database contains a vector of Qbv gate states, the number of packets waiting in each queue, and other relevant information that aids in evaluating the agent's schedule.

NeSTiNg allows an easy modification of the IEEE 802.1Qbv configuration through XML files, as well as result retrieval and analysis using Python. This feature enables 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 0.1 seconds per iteration. It is important to note that OMNeT++/INET simulation cannot be parallelized. Therefore, if we were to set the simulation time to 10 seconds, our training, which includes over 50000 steps, would take approximately 18 days to complete.

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

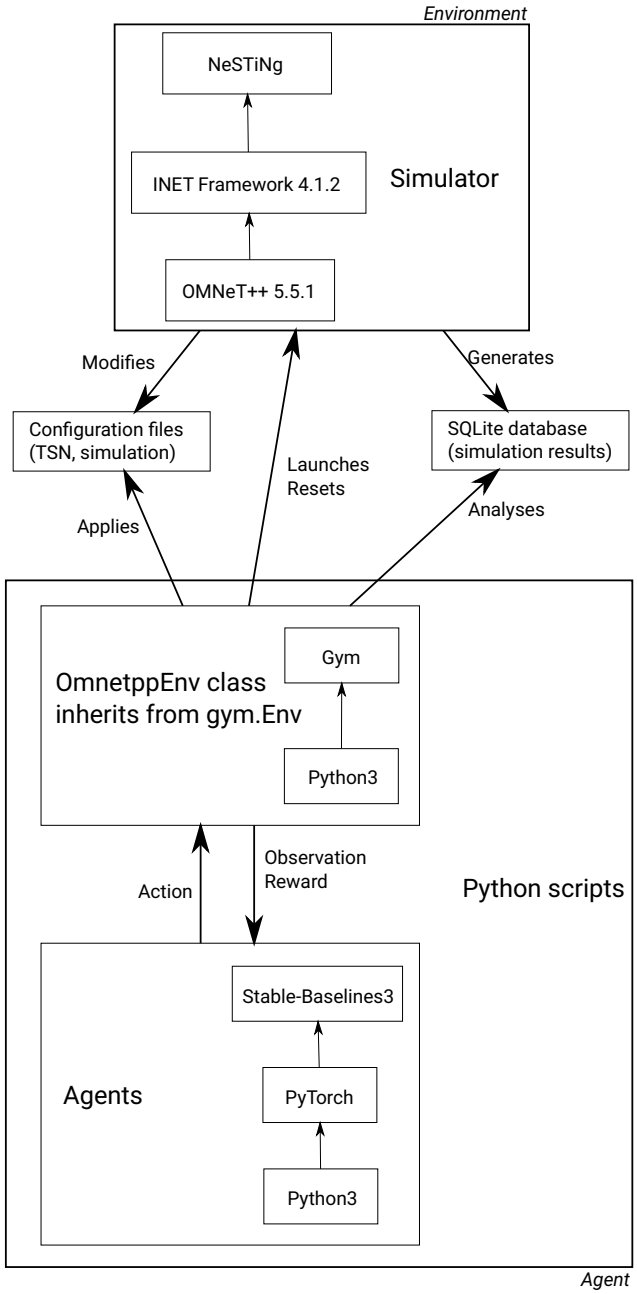

Fig. 5. Architecture of the proposed solution

4.2 RL-agent

For the implementation of the RL-agent, the RL Baselines3 Zoo [24] framework appears to be the appropriate training framework. The RL Baselines3 Zoo is constructed on top of Stable-Baselines3 [25], which is a collection of dependable and open source RL implementations in Python. The main goal

of Stable-Baselines3 is to ensure simplicity and reliability [25], and it relies on PyTorch [22] for the neural networks. The RL Baselines3 Zoo [24] empowers the optimization of the neural network's hyperparameters, and it comes with a set of scripts for training agents and visualizing the results.

One of the most challenging aspects of RL is implementing the communication between the agent and the environment. To accomplish this, one solution is to use a library named OpenAI Gym [2]. Gym is an API that enables RL problems to be modeled, and it provides a set of environments that are ready to use. However, OMNeT++ is not included in the list of supported environments. As a result, we designed and developed a module named OmnetppEnv that manages the communication between the agent and the OMNeT++ simulator. OmnetppEnv is essentially an interface between OMNeT++ and Stable-Baselines3, and it employs the OpenAI Gym API. This module has two primary responsibilities:

(1) interpret the actions provided by the agent and convert them into instructions that can be understood by OMNeT++;
(2) translate the simulation outcomes from the SQLite database into a reward and a new state that can be understood by the agent.

At the beginning of each episode, OmnetppEnv generates an environment that can be utilized by the RL-agent. It first compiles the OMNeT++/INET simulation and sets up the topology, after which the agent can launch it.

4.3 Agent's evaluation

In this section, we provide two kind of evaluations:

(1) an evaluation during the training: this allows to monitor the agent's training progress;
(2) an evaluation at the end of the training on new environment setting (i.e. not seen by the agent during the training) to see if the agent is indeed able to achieve its goals.

During the training, we evaluated the agent at each 10000 timesteps. In other word, we stopped the training each 10000 timesteps in order to evaluate the agent's progression. This evaluation consists in running the agent over 10 episodes and measure two main metrics:

(1) The mean episode length over the 10 episodes. It represents how much try (on average) the agent needs before finding a solution. The mean episode length is computed in the following way: $total\ number\ of\ tries\ over\ 10\ episodes/10$.
(2) The mean reward over the 10 episodes. This metric is used to show the agent's progression, as its reward is supposed to become better during the training. The mean is computed the following way: $sum\ of\ the\ reward\ gained\ over\ 10\ episodes/10$.

In overall, we conducted 5 evaluations over the whole training (which was over 50000 timesteps).

Figure 6 shows the average episode length at each evaluation. We note that the number of timesteps needed by the agent to find the adequate Qbv configuration is decreasing over time. For instance, in the 3rd evaluation, the agent required on average 27 tries before arriving at a terminal state whereas in the 5th evaluation, the agent needed 24 tries on average. This shows a learning progress of the agent.

In the same manner, Figure 7 shows the average reward that the agent gained at each evaluation. We can see that the mean of the rewards obtained by the agent gets better at each iteration. In the 3rd evaluation, the agent gained an average reward of -23 whereas in the 5th evaluation, the agent gained a reward of -20 on average. This shows that the agent gets less negative rewards over the time and therefore is doing less bad actions or find a solution quicker.

Both figures show that the agent is improving itself during the training as it becomes able to find a good solution faster.

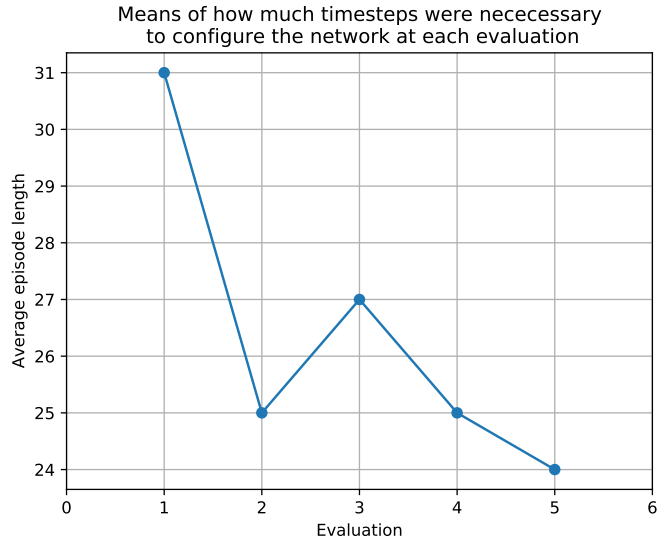

Fig. 6. Means of steps during evaluation

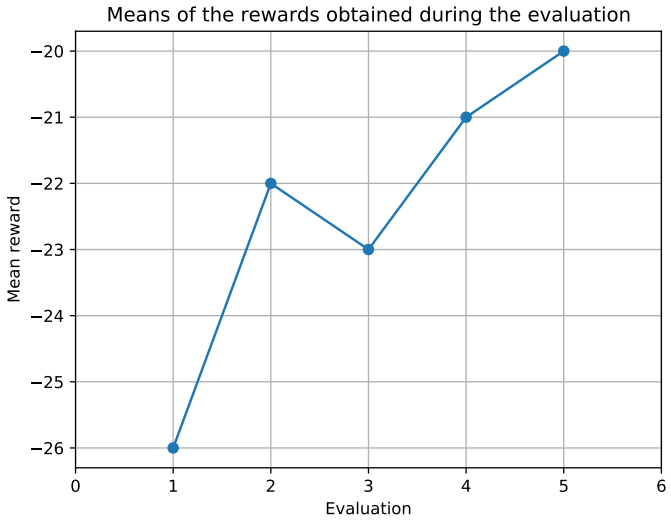

Fig. 7. Means of rewards during evaluation

At the end of the training (i.e., after 50000 timesteps), we run the agent on a test environment. This latter should be different from the environment seen by the agent during the training. Table 2 provides the setting parameters of the test environment.

Through this evaluation, we aim at verifying 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 will be based on two observations:

Table 2. Setting parameters of the test environment

Parameter	Value
Number of switches	5
Links' capacity	1 Gbps
TSN payload size	1000 B
BE payload size	500 B
TSN packet send interval	500 μs
BE packet send interval	200 μs
TSN deadline to be respected	2.5 ms

(1) whether the TSN deadline has been respected (i.e. the end-to-end latency of each TSN packets should be lower than the deadline);
(2) how much time was needed by the Agent to come out with a good configuration (i.e. the running time).

The agent was tested on a laptop with an Intel®Core™i5-8265U CPU running at 1.60GHz with 15.5 Gio RAM. For this test, we simulated 10 seconds in order to have better results.

The scheduling decided by the agent is given on Table 3. We note that the agent proposes a cycle length of 490000 nanoseconds. The gates are opened for TSN for 16000 nanoseconds, then the gates for all other traffic are opened during the rest of the time.

Table 3. Scheduling decided by the agent

Parameter	Value
Cycle length	490000 ns
TSN sequence length	16000 ns
BE sequence length	474000 ns

Figure 8 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 9 shows the packet delay variation (i.e. jitter) for the TSN traffic during the first 1000ms of the simulation. We remark that, at some instant, TSN traffic experience some jitter (less than 0.5 ms). In fact, the agent looks for an acceptable configuration and takes the first one it finds. It doesn't look for the best configuration. This jitter is still acceptable as the TSN traffic deadline is respected.

In order to evaluate the time needed by the agent to come out with a good Qbv configuration, we measured the initiation time (i.e. the time when the agent is starting) and the ending time (i.e. the time when the agent has found the good Qbv configuration and validated it on the test environment). The difference between both of these parameters will enables to have an idea about the running time of the agent. In our case, the agent needed around 34 seconds to decide Qbv configuration, test and validate it in the test environment. However, the main part of this running time is related to the time needed to run the test simulation, that last around 33 seconds. In fact, OMNeT++/INET and NeSTiNg provide a detailed and precise network modelization which leads to high execution time.

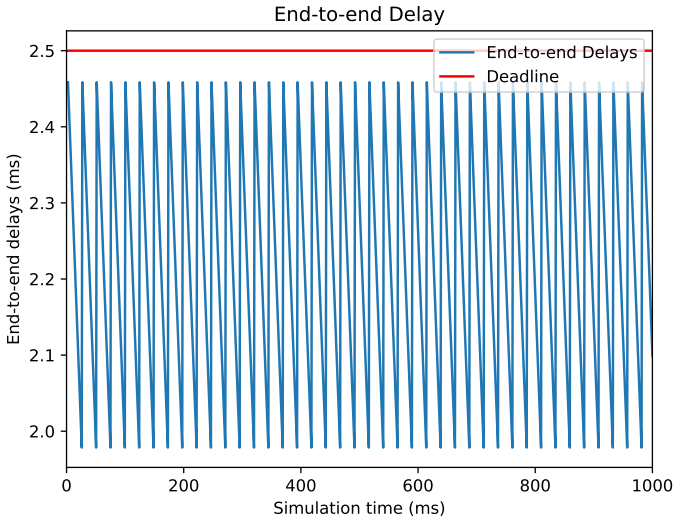

Fig. 8. End-to-end delays of the TSN flow during the first 1000ms of the simulation

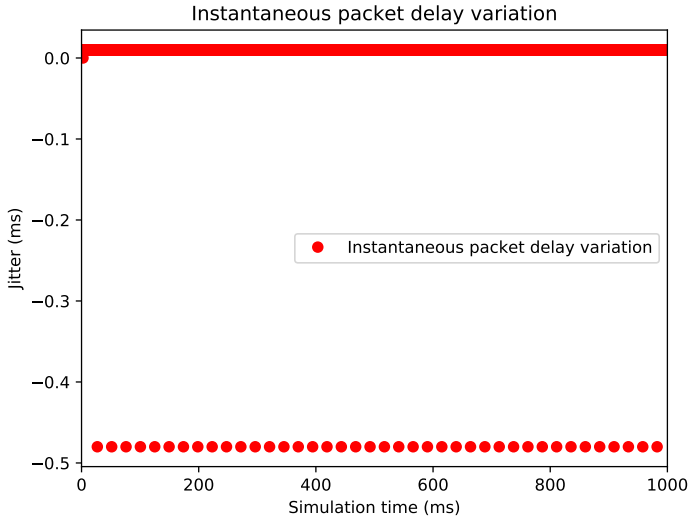

Fig. 9. Instantaneous packet delay variation of the TSN flow during the first 1000ms of the simulation

5 CONCLUSION AND FURTHER WORK

In this article, we put forth an RL-centered approach for setting up IEEE 802.1Qbv schedules in TSN networks. Our assessments indicated that the agent we designed can present an acceptable configuration, highlighting the potential of RL to create independent network configuration solutions for TSN networks. Nonetheless, there is still much work to be done in order to establish a fully functional architecture.

Our research involved some assumptions to simplify the scheduling problem, which allowed us to examine the ability of the RL agent to configure Qbv in simple scenarios. Moving forward, we aim to relax these assumptions and consider more complexes scenarios such as considering the case where we have multiple TSN flows or even more complex topology. To achieve this, we must work on enhancing our agent's capabilities to configure multiple switches differently. A potential area of exploration is multi-agent reinforcement learning. Ultimately, our goal is to train an agent that has no prior knowledge of the flows, in order to be able to configure a full open network where new flows are added in an incremental way.

REFERENCES

[1] Mohammad Ashjaei, Gaetano Patti, Moris Behnam, Thomas Nolte, Giuliana Alderisi, and Lucia Lo Bello. 2017. Schedulability analysis of Ethernet Audio Video Bridging networks with scheduled traffic support. *Real-Time Systems* 53, 4 (2017), 526–577.

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv:1606.01540* (2016).

[3] Silviu S Craciunas, Ramon Serna Oliver, Martin Chmelík, and Wilfried Steiner. 2016. Scheduling real-time communication in IEEE 802.1Qbv time sensitive networks. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*. 183–192.

[4] Aellison Cassimiro T dos Santos, Ben Schneider, and Vivek Nigam. 2019. TSNSCHED: Automated schedule generation for time sensitive networking. In *2019 Formal Methods in Computer Aided Design (FMCAD)*. IEEE, 69–77.

[5] Frank Dürr and Naresh Ganesh Nayak. 2016. No-wait packet scheduling for IEEE time-sensitive networks (TSN). In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*. 203–212.

[6] Jonathan Falk, David Hellmanns, Ben Carabelli, Naresh Nayak, Frank Dürr, and Stephan Kehrer. 2019. NeSTiNg: Simulating IEEE Time-sensitive Networking (TSN) in OMNeT++. In *Proceedings of the 2019 International Conference on Networked Systems (NetSys)*. Garching b. München, Germany.

[7] Janos Farkas, Lucia Lo Bello, and Craig Gunther. 2018. Time-sensitive networking standards. *IEEE Communications Standards Magazine* 2, 2 (2018), 20–21.

[8] Norman Finn. 2018. Introduction to time-sensitive networking. *IEEE Communications Standards Magazine* 2, 2 (2018), 22–28.

[9] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*. PMLR, 1861–1870.

[10] Kurt Hornik. 1991. Approximation capabilities of multilayer feedforward networks. *Neural networks* 4, 2 (1991), 251–257.

[11] IEEE 802.1 Working Group. 2016. 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)* (2016), 1–57. https://doi.org/10.1109/IEEESTD.2016.8613095

[12] IEEE 802.1 Working Group. 2018. IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks. *IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014)* (2018), 1–1993. https://doi.org/10.1109/IEEESTD.2018.8403927

[13] IEEE 802.1 Working Group. 2018. 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)* (2018), 1–208. https://doi.org/10.1109/IEEESTD.2018.8514112

[14] IEEE 802.1 Working Group. 2020. 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)* (2020), 1–421. https://doi.org/10.1109/IEEESTD.2020.9121845

[15] Joseph Y-T Leung and Jennifer Whitehead. 1982. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance evaluation* 2, 4 (1982), 237–250.

[16] Zoubir Mammeri. 2019. Reinforcement learning based routing in networks: Review and classification of approaches. *IEEE Access* 7 (2019), 55916–55950.

[17] John L Messenger. 2018. Time-sensitive networking: An introduction. *IEEE Communications Standards Magazine* 2, 2 (2018), 29–33.

[18] 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).

[19] Ahmed Nasrallah, Venkatraman Balasubramanian, Akhilesh Thyagaturu, Martin Reisslein, and Hesham ElBakoury. 2019. Reconfiguration algorithms for high precision communications in time sensitive networks. In *2019 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 1–6.

[20] Nicolas Navet, Tieu Long Mai, and Jörn Migge. 2019. *Using machine learning to speed up the design space exploration of Ethernet TSN networks.* Technical Report. University of Luxembourg.

[21] Naresh Ganesh Nayak, Frank Dürr, and Kurt Rothermel. 2016. Time-sensitive software-defined network (TSSDN) for real-time applications. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*. 193–202.

[22] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. 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., 8024–8035.

[23] 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.

[24] Antonin Raffin. 2020. RL Baselines3 Zoo. https://github.com/DLR-RM/rl-baselines3-zoo.

[25] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. 2021. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research* 22, 268 (2021), 1–8. http://jmlr.org/papers/v22/20-1364.html

[26] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[27] Viswanath Sivakumar, Olivier Delalleau, Tim Rocktäschel, Alexander H Miller, Heinrich Küttler, Nantas Nardelli, Mike Rabbat, Joelle Pineau, and Sebastian Riedel. 2019. Mvfst-rl: An asynchronous rl framework for congestion control with delayed actions. *arXiv preprint arXiv:1910.04054* (2019).

[28] Thomas Stüber, Lukas Osswald, Steffen Lindner, and Michael Menth. 2022. A Survey of Scheduling in Time-Sensitive Networking (TSN). *arXiv preprint arXiv:2211.10954* (2022).

[29] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction.* MIT press.

[30] Ken W Tindell, Alan Burns, and Andy J. Wellings. 1992. Allocating hard real-time tasks: an NP-hard problem made easy. *Real-Time Systems* 4, 2 (1992), 145–165.

[31] Xiaolong Wang, Haipeng Yao, Tianle Mai, Tianzheng Nie, Lin Zhu, and Yunjie Liu. 2022. Deep Reinforcement Learning aided No-wait Flow Scheduling in Time-Sensitive Networks. In *2022 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 812–817.

[32] Hao Yu, Tarik Taleb, and Jiawei Zhang. 2022. Deep Reinforcement Learning based Deterministic Routing and Scheduling for Mixed-Criticality Flows. *IEEE Transactions on Industrial Informatics* (2022).

Received 27 June 2023; revised 12 March 2009; accepted 5 June 2009