Ordonnancer le trafic dans des réseaux déterministes grâce à l'apprentissage par renforcement

Adrien Roberty

ISAE-ENSMA
CEA-List

École doctorale **ED Mathématiques, Informatique, Matériaux, Mécanique, Énergétique (MIMME) n° 651**

Unité de recherche **Laboratoire des Systèmes Communicants (LSC)**

Thèse présentée par **Adrien Roberty**

Soutenue le **6 février 2024**

En vue de l'obtention du grade de docteur de l'ISAE-ENSMA

Discipline **Informatique et applications**

Ordonnancer le trafic dans des réseaux déterministes grâce à l'apprentissage par renforcement

Thèse dirigée par	Annie Geniet	directrice
	Frédéric Ridouard	co-directeur
	Siwar Ben Hadj Said	co-encadrante
	Henri Bauer	co-encadrant

Composition du jury

Rapporteurs	Katia Jaffrès-Runser	professeure à l'INP-ENSEEIHT Toulouse
	Abdelmadjid Bouabdallah	professeur à l'Université de Technologie de Compiègne
Examinateurs	Nicolas Navet	professeur à l'Université du Luxembourg
	Mireille Sarkiss	maître de conférence à Télécom SudParis
Directeurs de thèse	Annie Geniet	professeure à l'Université de Poitiers
	Frédéric Ridouard	maître de conférence HDR à l'ISAE-ENSMA
	Siwar Ben Hadj Said	ingénieur-chercheur au CEA-List
	Henri Bauer	maître de conférence à l'ISAE-ENSMA

Colophon

Mémoire de thèse intitulé « Ordonnancer le trafic dans des réseaux déterministes grâce à l'apprentissage par renforcement », écrit par Adrien Roberty, achevé le 15 février 2024, composé au moyen du système de préparation de document LaTeX et de la classe yathesis dédiée aux thèses préparées en France.

ISAE-ENSMA
CEA-List

École doctorale **ED Mathématiques, Informatique, Matériaux, Mécanique, Énergétique (MIMME) n° 651**

Unité de recherche **Laboratoire des Systèmes Communicants (LSC)**

Thèse présentée par **Adrien Roberty**

Soutenue le **6 février 2024**

En vue de l'obtention du grade de docteur de l'ISAE-ENSMA

Discipline **Informatique et applications**

Ordonnancer le trafic dans des réseaux déterministes grâce à l'apprentissage par renforcement

Thèse dirigée par	Annie Geniet	directrice
	Frédéric Ridouard	co-directeur
	Siwar Ben Hadj Said	co-encadrante
	Henri Bauer	co-encadrant

Composition du jury

Rapporteurs	Katia Jaffrès-Runser	professeure à l'INP-ENSEEIHT Toulouse
	Abdelmadjid Bouabdallah	professeur à l'Université de Technologie de Compiègne
Examinateurs	Nicolas Navet	professeur à l'Université du Luxembourg
	Mireille Sarkiss	maître de conférence à Télécom SudParis
Directeurs de thèse	Annie Geniet	professeure à l'Université de Poitiers
	Frédéric Ridouard	maître de conférence HDR à l'ISAE-ENSMA
	Siwar Ben Hadj Said	ingénieur-chercheur au CEA-List
	Henri Bauer	maître de conférence à l'ISAE-ENSMA

ISAE-ENSMA
CEA-List

Doctoral School **ED Mathématiques, Informatique, Matériaux, Mécanique, Énergétique (MIMME) n° 651**

University Department **Laboratoire des Systèmes Communicants (LSC)**

Thesis defended by **Adrien Roberty**

Defended on **February 6, 2024**

In order to become Doctor from ISAE-ENSMA

Academic Field **Computing and applications**

Scheduling the traffic in deterministic networks through reinforcement learning

Thesis supervised by	Annie Geniet	Supervisor
	Frédéric Ridouard	Co-Supervisor
	Siwar Ben Hadj Said	Co-Monitor
	Henri Bauer	Co-Monitor

Committee members

Referees	Katia Jaffrès-Runser	Professor at INP-ENSEEIHT Toulouse
	Abdelmadjid Bouabdallah	Professor at Université de Technologie de Compiègne
Examiners	Nicolas Navet	Professor at Université du Luxembourg
	Mireille Sarkiss	Associate Professor at Télécom SudParis
Supervisors	Annie Geniet	Professor at Université de Poitiers
	Frédéric Ridouard	HDR Associate Professor at ISAE-ENSMA
	Siwar Ben Hadj Said	Research engineer at CEA-List
	Henri Bauer	Associate Professor at ISAE-ENSMA

Mots clés : apprentissage par renforcement (intelligence artificielle), apprentissage profond, industrie 4.0, intelligence artificielle, ordonnancement (informatique), temps réel (informatique), industrie 5.0, time-sensitive networking (tsn), time-aware shaper (tas)

Keywords: reinforcement learning, deep learning, industry 4.0, artificial intelligence, computer scheduling, real-time data processing, industry 5.0, time-sensitive networking (tsn), time-aware shaper (tas)

Cette thèse a été préparée dans les laboratoires suivants.

Laboratoire des Systèmes Communicants (LSC)
Centre d'intégration Nano-INNOV
2 Boulevard Thomas Gobert
91120 Palaiseau
France

Site https://list.cea.fr

Laboratoire d'Informatique et d'Automatique pour les Systèmes (LIAS)
Laboratoire LIAS - ISAE-ENSMA
Téléport 2 - 1 avenue Clément Ader
BP 40109
86961 Chasseneuil
France

☎ +33 5 49 49 80 63
✉ secretariat-lias@ensma.fr
Site https://www.lias-lab.fr/fr/

Ordonnancer le trafic dans des réseaux déterministes grâce à l'apprentissage par renforcement

Résumé

L'un des changements les plus perturbateurs apportés par l'industrie 4.0 est la mise en réseau des installations de production. De plus, les discussions portant sur l'Industrie 5.0 montrent la nécessité d'un écosystème industriel intégré, combinant IA et jumeau numérique. Dans cet environnement, les équipements industriels fonctionneront de manière transparente avec les travailleurs humains, nécessitant une latence minimale et une connectivité haut débit pour la surveillance en temps réel. Afin de répondre à ces exigences, l'ensemble de standard Time-Sensitive Networking (TSN) a été introduit. Cependant, la configuration de TSN dans un réseau industriel complexe pose de nouveaux défis. Par exemple, les standards TSN permettent une certaine flexibilité et modularité dans le plan de données, néanmoins, les mécanismes définis dans ces standards dépendent de nombreux paramètres (tels que la topologie du réseau, le routage, etc.) ce qui rend le travail de conception difficile. IEEE 802.1Q est l'un des principaux standards TSN qui fournit plusieurs mécanismes pour atteindre une latence déterministe. L'un d'eux s'appelle le Time-Aware Shaper (TAS). Un commutateur avec une fonction TAS divise le trafic de données, à travers plusieurs priorités, en plusieurs files d'attente organisées selon un ordonnancement régulier. Les principales manières d'organiser ce processus sont basées sur des méthodes exactes ou heuristiques. Ceux-ci sont bons pour les réseaux fermés (lorsque tous les flux sont identifiés à l'avance et que la topologie du réseau est fixe). Cependant, dans un réseau ouvert (où plus de flux sont ajoutés au réseau et la topologie du réseau est dynamique), l'ordonnancement dans TSN peut entraîner des problèmes NP-difficile. L'objectif de cette thèse est de proposer une solution pour traiter l'ordonnancement dans TSN en utilisant l'apprentissage par renforcement profond avec l'utilisation de simulations pour entraîner et évaluer l'agent de configuration.

Mots clés : apprentissage par renforcement (intelligence artificielle), apprentissage profond, industrie 4.0, intelligence artificielle, ordonnancement (informatique), temps réel (informatique), industrie 5.0, time-sensitive networking (tsn), time-aware shaper (tas)

Scheduling the traffic in deterministic networks through reinforcement learning

Abstract

One of the most disruptive changes brought by Industry 4.0 is the networking of production facilities. Furthermore, the discussions on Industry 5.0 show the need for an integrated industrial ecosystem, combining AI and the digital twin. In this environment, industrial equipment will work seamlessly with human workers, requiring minimal latency, high-speed connectivity for real-time monitoring. In order to meet this requirement, the Time-Sensitive Networking (TSN) set of standards was introduced. However, configuring TSN in a complex industrial network poses new challenges. For example, the TSN standard allow some flexibility and modularity in the data plane, however, the mechanisms defined by these standards depends on many parameters (such as network topology, routing, etc.) which makes the design work difficult. IEEE 802.1Q is one of the main TSN standards that provides several mechanisms to achieve deterministic latency. One of them is called Time-Aware Shaper (TAS). A switch with a TAS function divides the data traffic, through multiple priorities, into multiple queues arranged in a regular schedule. The main way to organize this process is based on exact or heuristic methods. These are good for closed networks (when all streams are identified in advance and the network topology is fixed). However, in an open network (where more streams are added to the network and the network topology is dynamic), scheduling in TSN can lead to NP-hard problems. The goal of this thesis is to propose a solution to process the scheduling in TSN using Deep Reinforcement Learning with the use of simulations to train and evaluate the configuration agent.

Keywords: reinforcement learning, deep learning, industry 4.0, artificial intelligence, computer scheduling, real-time data processing, industry 5.0, time-sensitive networking (tsn), time-aware shaper (tas)

Laboratoire des Systèmes Communicants (LSC) – Centre d'intégration Nano-INNOV – 2 Boulevard Thomas Gobert – 91120 Palaiseau – France
Laboratoire d'Informatique et d'Automatique pour les Systèmes (LIAS) – Laboratoire LIAS - ISAE-ENSMA – Téléport 2 - 1 avenue Clément Ader – BP 40109 – 86961 Chasseneuil – France

Remerciements

Je tiens à exprimer ma sincère gratitude envers toutes les personnes qui ont contribué à la réalisation de cette thèse. Leurs encouragements, leur soutien et leurs conseils ont été d'une aide inestimable.

Tout d'abord, je souhaite exprimer ma profonde gratitude envers mes encadrants, dont les conseils éclairés et le soutien indéfectible ont été d'une importance capitale tout au long de cette aventure intellectuelle. Je tiens à remercier chaleureusement Mme. Annie Geniet, professeure à l'Université de Poitiers, ma directrice de thèse, M. Frédéric Ridouard, maître de conférence HDR à l'ISAE-ENSMA, mon co-directeur de thèse et M. Henri Bauer, maître de conférence à l'ISAE-ENSMA, mon encadrant universitaire, pour leur expertise, leur patience et leur disponibilité. Leurs précieux conseils et leur engageante passion pour la recherche ont été des piliers solides sur lesquels je me suis appuyé tout au long de ma thèse. Je suis également redevable à Mme. Siwar Ben Hadj Said, ingénieur-chercheur au CEA-List, mon encadrante au sein du CEA, pour son soutien inestimable, ses conseils pratiques et son expertise qui ont grandement enrichi mon travail.

J'adresse tous mes remerciements à Mme. Katia Jaffrès-runser, professeure à l'INP-ENSEEIHT Toulouse, ainsi qu'à M. Abdelmadjid Bouabdallah, professeur à l'Université de Technologie de Compiègne, de l'honneur qu'ils m'ont fait en acceptant d'être rapporteurs de cette thèse. J'exprime également ma gratitude à M. Nicolas Navet, professeur à l'Université du Luxembourg et à M. Mireille Sarkiss, maître de conférence à Télécom SudParis, qui ont bien voulu être examinateurs.

Je tiens à exprimer ma gratitude à tous les membres de mon laboratoire d'accueil au CEA, le LSC. Leur collaboration, leur expertise et leur volonté de partager leurs connaissances ont grandement enrichi mon expérience de recherche. Leur soutien, tant sur le plan technique que sur le plan humain, a été inestimable. Je tiens tout d'abord à exprimer ma gratitude envers le chef de laboratoire, Alexis Olivereau, pour son accueil et sa bonne humeur, qui font de ce laboratoire un lieu si chaleureux. Je suis également reconnaissant envers les autres membres du laboratoire, en particulier Pierre, Mounir et Minh-Thuyen, pour leur collaboration précieuse et les échanges scientifiques enrichissants que nous avons partagés, ainsi qu'envers Thomas, Adrien, Chuan, Hamdi, Christina, Luc et Lucas et les anciens membres Julien, Minh, Thomas, Fabien et Quentin pour leur bonne humeur et leur disponibilité. Leur soutien et leur expertise ont joué un rôle déterminant dans la réalisation de ce travail de thèse. Mes remerciements vont pareillement à Annick et Valérie, les assistantes du laboratoire. Leur professionnalisme, leur disponibilité et leur dévouement ont grandement facilité mes démarches et m'ont permis de me consacrer pleinement à mes travaux de recherche.

Je souhaite également exprimer ma reconnaissance envers les membres de l'association des doctorants du CEA, ACTIF, qui a été un pilier central de ma vie doctorale. Grâce à leur engagement et à leurs initiatives, j'ai pu bénéficier d'une communauté soudée et solidaire. Les échanges, les sessions de formation et les événements organisés par l'association ont non

seulement enrichi mon réseau professionnel, mais aussi nourri mon épanouissement personnel. Je tiens à adresser mes remerciements particuliers à Valentin, qui a relancé l'association après les années difficiles marquées par les confinements successifs, ainsi qu'à Julien et Jonathan, avec qui j'ai pu organiser un voyage scientifique au centre CEA de Cadarache et à ITER, voyage qui restera comme un très bon souvenir. Mes remerciements vont aussi à Clément, Guillaume, Marc et Simon, membres actifs de l'association, ainsi qu'à tous ceux, trop nombreux pour les citer, grâce à qui l'association fonctionne. Ils travaillent sans relâche pour améliorer notre environnement de recherche au CEA et je leur en suis très reconnaissant. Je tiens à étendre ces remerciements à Fabien et Nan, que j'ai rencontrés pour la première fois lors de l'école d'été temps-réel à Poitiers. Nous avons continué à nous rencontrer régulièrement ainsi qu'à échanger sur nos sujets de thèse respectifs, celles-ci partageant en commun le temps-réel. Ces échanges ont toujours été très enrichissant.

Je souhaite exprimer ma gratitude envers ma famille et mes amis qui m'ont soutenu et encouragé tout au long de cette aventure. Leur confiance en moi, leurs mots d'encouragement et leur soutien moral ont été des ressources essentielles pour surmonter les défis et rester concentré sur mon travail.

Enfin, je voudrais adresser mes mots de remerciements à toutes les personnes qui, de près ou de loin, ont contribué à la réalisation de cette thèse, notamment au CEA, qui a financé ce projet.

Leurs contributions ont été inestimables et je leur suis profondément reconnaissant.

Acronymes

A | B | C | D | G | I | L | M | O | P | R | S | T | V

A

ANSI American National Standards Institute. 11, *Voir* ANSI
API Application Programming Interface. 69, 80, 91, *Voir* API
ATS Asynchronous Traffic Shaper. *Voir* ATS

B

BCMA Best Master Clock Algorithm. 20, *Voir* Best Master Clock Algorithm
BE Best Effort. 14, 62, 71, 72, 78, 79, 81, 84, 86, 90

C

CBS Credit-Based Shaper. 19, *Voir* CBS
CNC Centralized Network Configuration. 17–19, 93, 94
CUC Centralized User Configuration. 18, 19

D

DQL Deep Q-learning. 43–45
DRL Deep Reinforcement Learning. 27, 33, 39, 41, 49, 53, 54
DSE Design-Space Exploration. 51, 54, *Voir* algorithmes DSE

G

GCL Gate Control List. 15, 53, 92
GNN Graph Neural Network. 54
gPTP generalized Precision Time Protocol. 20, 61, 93

I

IA Intelligence Artificielle. 3, 21, 25, 26, 53, 54, 73
IEEE Institute of Electrical and Electronics Engineers. 11, 12, 68
ILP Integer Linear Programming. 51, *Voir* Integer Linear Programming

L

LAN Local Area Network. 12

M

MARL Multi-Agent Reinforcement Learning. 90, 91
MDP Markov Decision Process. 35–37, 66, 68, 79, 89
ML Machine Learning. 25, 26, 35, 49, 53
MSDU Maximum Service Data Unit. 15
MTU Maximum Transmission Unit. 62, *Voir* MTU

O

OPC UA OPC Unified Architecture. 18, *Voir* OPC Unified Architecture
OSI Open Systems Interconnection. 11, 12, 53, *Voir* modèle OSI

P

PCP Priority Code Point. 13, 14
PMC Perceptron MultiCouche. 28, 29, 31
PPO Proximal Policy Optimization. 45, 46, 49, 63, 80, 81
PTP Precision Time Protocol. 20

R

RL Reinforcement Learning. 26, 27, 35–37, 42, 45, 49, 53, 54, 63, 65, 73, 77, 78, 89–91
RNA Réseaux de Neurones Artificiels. 27, 36

S

SAC Soft Actor-Critic. 46, 49, 63, 68, 80, 81
SDN Software-Defined Networking. 19, 52, 93, *Voir* Software-defined networking
SMT Satisfiability Modulo Theories. 51, 52, *Voir* Satisfiability Modulo Theories

T

TAS Time-Aware Shaper. 3, 14–16, 19, 20, 26, 49, 51, 52, 54, 61, 63, 65–67, 69, 71, 73, 77–80, 89, 90, 92–94
TDMA Time-Division Multiple Access. 14, *Voir* Time-Division Multiple Access
TSN Time Sensitive Networking. 2, 3, 9, 11–13, 15–18, 20, 26, 49, 51, 53, 54, 62, 63, 65, 68, 78, 79, 89, 90, 92, 93

V

VLAN Virtual Local Area Network. 12

Symboles

a une action, et a_t l'action à l'instant t; de manière générale, la lettre a est liée aux actions 38, 43

$\mathbb{A}$ espace des actions 36, 38

$\mathbb{X}$ espace des entrées d'un réseau de neurones 27

$\mathbb{S}$ espace des états 36, 38

$\mathbb{O}$ espace des observations 38

$\mathbb{W}$ espace des poids possibles d'un réseau de neurones 27

$\mathbb{Y}$ espace des sorties d'un réseau de neurones 27

s un état, et s_t l'état à l'instant t; de manière générale, la lettre s est liée aux états 37, 42, 43

$R(s,a,s')$ récompense obtenue par l'agent après être passé de l'état s vers l'état s' en effectuant l'action a 36

$T(s',r|s,a)$ probabilité, en partant de l'état s et en effectuant l'action a, de passer à l'état s' en touchant une récompense r 36

$\nabla\phi(w)$ gradient de f par rapport à w au point x avec les poids w, il s'agit d'un vecteur 30

o une observation 37

w poids d'un lien 28

π fonction de politique 39

π^* fonction de politique optimale 40

$Q(s,a)$ fonction de valeur état-action pour l'état s et l'action a 39, 43

$Q^*(s,a)$ fonction de valeur optimale état-action pour l'état s et l'action a 39, 42, 43

r une récompense, et r_t la récompense à l'instant t; de manière générale, la lettre r est liée aux récompenses 38

y sortie attendue d'un réseau de neurones ($y \in \mathbb{Y}$) 28

$f(x,w)$ sortie calculée par le réseau de neurones pour l'entrée x avec les poids w ($f(x,w) \in \mathbb{Y}$) 27, 29

$V(s)$ fonction de valeur pour l'état s 39, 42

$V^*(s)$ fonction de valeur optimale pour l'état s 39, 42

Sommaire

Liste des tableaux

Table des figures

Introduction générale

Sommaire du présent chapitre

Contexte de la thèse

L'industrie 4.0 et l'industrie 5.0 font référence aux dernières phases de la révolution industrielle, entraînées par l'intégration des technologies numériques et de l'automatisation dans les processus de fabrication.

L'industrie 4.0 est l'industrie née de la quatrième révolution industrielle, qui se concentre sur la numérisation des processus de fabrication (on rappelle que la première révolution industrielle est celle de la mécanisation, la seconde celle de la production de masse et la troisième celle de l'automatisation). Il comprend des technologies telles que l'Internet des objets (IoT), le cloud computing, l'analyse de mégadonnées, l'intelligence artificielle et les systèmes cyberphysiques. Les principaux objectifs de l'industrie 4.0 sont d'améliorer l'efficacité, la flexibilité et la productivité dans le secteur manufacturier, mais également de réduire les interventions humaines ainsi que les coûts et la consommation d'énergie au strict minimum. Pour ce faire, l'industrie 4.0 s'appuie sur les innovations liées à l'Internet des objets et aux technologies du numérique (Lo Bello et Steiner 2019 ; Silva et al. 2019). D'un point de vue réseau, cette révolution présente de nombreux défis. On peut citer :

— L'interopérabilité : étant donné que divers appareils et systèmes sont connectés dans l'industrie 4.0, il est nécessaire d'assurer une communication transparente entre les différentes plateformes et technologies.
— La sécurité : la connectivité accrue soulève aussi des préoccupations au sujet de la cybersécurité, car les réseaux industriels deviennent des cibles potentielles pour les cybermenaces.
— La communication en temps réel : l'industrie 4.0 exige des latences faibles et une communication déterministe pour prendre en charge les applications critiques en temps. Les réseaux Ethernet traditionnels ont souvent du mal à répondre à ces exigences.

L'industrie 5.0, quant à elle, représente la phase future de la révolution industrielle, en se concentrant sur la collaboration entre les humains et les technologies avancées comme la robotique et l'intelligence artificielle. Elle vise à créer une relation symbiotique entre les humains et les machines pour débloquer de nouveaux niveaux de productivité et d'innovation. Cette vision englobe des concepts comme les jumeaux numériques, la communication centrée sur

l'humain et l'intégration de l'intelligence artificielle (IA). Dans cet environnement avancé, les équipements industriels devraient collaborer de manière transparente avec les travailleurs humains, nécessitant une connectivité à faible latence et à haut débit de données pour la surveillance en temps réel. L'industrie 5.0 étant encore un concept en discussion, les défis liés au réseau restent à définir (Xu et al. 2021). Cependant, certains défis potentiels pourraient inclure :

— Des interactions homme-machines transparentes : concevoir des réseaux qui prennent en charge la communication en temps réel et fiable entre les humains et les machines, permettant une collaboration intuitive.
— Edge computing et latence : l'industrie 5.0 peut nécessiter des capacités informatiques plus proches des périphériques edge pour réduire la latence et accélérer la prise de décision.
— L'évolutivité du réseau : construire des réseaux qui peuvent facilement s'adapter à un nombre croissant d'appareils connectés, tout en maintenant les performances et le déterminisme.

De plus, une usine 4.0/5.0 est composée de postes de travail mobiles et flexibles qui peuvent être agencés de manières différentes en fonction de la production. Ainsi, on peut très bien imaginer vouloir déplacer un capteur vers une autre chaîne de production. Cette reconfiguration des postes de travail nécessitera une reconfiguration du réseau qui doit être dynamique. De même, à chaque fois qu'une mise à jour applicative a lieu (par exemple, on installe une application qui va requérir des informations provenant des capteurs et par conséquent, générer plusieurs flux), le réseau doit être reconfiguré dynamiquement.

Cadre de la thèse

Comme on peut le voir, une des caractéristiques principales des usines du futur est la mise en réseau des équipements de production (machines et lignes de production, robots, convoyage, stockage...), ce qui permettra aux équipements de production d'être capables de se contrôler, de se configurer et d'échanger des informations entre eux. Ceci implique donc des exigences de fiabilité, de latence et de longévité des périphériques de communication élevées. Ces objectifs peuvent être atteints grâce au Time Sensitive Networking (TSN), un ensemble de standards visant à ajouter des caractéristiques temps-réel à Ethernet en fournissant des capacités de déterminisme, de bande passante garantie et de synchronisation du temps dans les réseaux Ethernet traditionnels. TSN permet de résoudre certains des défis décrits plus haut grâce à :

— La synchronisation temporelle : TSN assure une synchronisation précise des appareils au sein d'un réseau, ce qui est crucial pour la communication et la coordination en temps réel.
— L'amélioration du déterminisme : TSN introduit des mécanismes d'ordonnancement, permettant d'assurer une latence faible et bornée.
— La standardisation de l'interopérabilité : TSN définit des protocoles communs, assurant l'interopérabilité entre différents dispositifs et systèmes, réduisant la complexité dans des environnements hétérogènes.

Cependant, l'utilisation de TSN pose de nouveaux défis. Par exemple, configurer l'ordonnancement des flux dans TSN peut mener à un problème NP-difficile (Serna Oliver, Silviu S. Craciunas et Steiner 2018 ; Silviu S. Craciunas et al. 2016 ; Tindell, Burns et Wellings 1992 ; Burns 1991 ; Leung et Whitehead 1982). De plus, cette configuration se doit d'être dynamique. Pour relever ces défis un algorithme capable de déterminer rapidement les configurations de TSN s'impose. Cet algorithme doit posséder l'agilité nécessaire pour réagir rapidement aux événements émergents, tels que l'introduction de nouveaux flux de données, les modifications de

la topologie du réseau ou les modifications des configurations de flux. En outre, il devrait exceller à accompagner le déploiement progressif des applications au sein du réseau TSN. Il convient de noter que cet algorithme devrait nettement surpasser les outils d'optimisation analytiques traditionnels, fournissant des résultats en quelques minutes plutôt qu'en quelques heures.

Une manière d'atteindre tous ces objectifs consiste à utiliser les techniques de l'Intelligence Artificielle (IA). En effet, la plupart des méthodes existantes, basées sur des solveurs ou des heuristiques, se heurtent soit au problème du temps nécessaire avant d'arriver à un résultat, soit à la dynamicité du système. L'utilisation de méthodes d'IA permet d'obtenir des résultats adaptés, tout en offrant une grande flexibilité et une prise de décision en rapide (Mammeri 2019 ; Mao et al. 2016 ; Tesauro 2007).

Le but de cette thèse est de montrer qu'un agent peut, à l'aide de méthode d'IA, décider un ordonnancement sans connaissances préalables sur les flux, en étant capable de s'adapter à un changement de topologie et dans un délai raisonnable (de l'ordre de la minute).

Aperçu de la thèse

Cette thèse est organisée en deux parties. Dans la première, nous explorons en détail la problématique de la thèse, les différentes technologies mises en œuvre ainsi que l'état de l'art. En particulier :

- Le Chapitre 1 revient sur la notion de délais de bout en bout et sur l'intérêt qu'il y a à se tourner vers des mécanismes comme le Time-Aware Shaper (TAS) de TSN
- Le Chapitre 2 donne un aperçu des standards TSN. Dans ce chapitre, nous décrivons le mécanisme TSN permettant l'ordonnancement en fonction du temps, ainsi que les différents protocoles et architectures définis dans les standards ayant un lien avec ce mécanisme.
- Le Chapitre 3 donne un aperçu des différentes méthodes d'IA. Ce chapitre permet d'explorer les fondamentaux de l'apprentissage profond, utilisé dans le chapitre suivant afin d'approximer des fonctions.
- Le Chapitre 4 explore plus en détail une méthode d'IA particulière, l'apprentissage par renforcement. Il s'agit de la méthode d'apprentissage pressentie pour pouvoir décider de l'ordonnancement.
- Le Chapitre 5 présente l'état de l'art sur le sujet de la configuration de réseaux TSN et plus spécifiquement, de l'ordonnancement dans les réseaux TSN.

La seconde partie de la thèse est dédiée aux contributions permettant de résoudre le problème de l'ordonnancement dans TSN. Ainsi :

- Le Chapitre 6 introduit les contributions, notamment en présentant les hypothèses communes ainsi que les outils utilisés.
- Le Chapitre 7 présente la première contribution. Dans cette contribution, nous cherchons à démontrer qu'à l'aide de l'apprentissage par renforcement, un agent est capable de trouver un ordonnancement commun à chaque commutateur TSN du réseau.
- Le Chapitre 8 présente la seconde contribution. L'objectif est ici de montrer qu'un agent peut trouver un ordonnancement différent sur chaque commutateur TSN.

Enfin, la Conclusion générale et perspectives résume les résultats de la thèse, et donne des perspectives de recherche, à savoir comment améliorer et exploiter l'agent de configuration.

Bibliographie du présent chapitre

Burns, Alan (1991). « Scheduling hard real-time systems : a review ». In : *Software Engineering Journal* 6.3, p. 116-128.

Craciunas, Silviu S. et al. (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*. RTNS '16. Brest, France : Association for Computing Machinery, p. 183-192. isbn : 9781450347877. doi : 10.1145/2997465.2997470. url : https://doi.org/10.1145/2997465.2997470.

Leung, Joseph Y.-T. et Jennifer Whitehead (1982). « On the complexity of fixed-priority scheduling of periodic, real-time tasks ». In : *Performance Evaluation* 2.4, p. 237-250. issn : 0166-5316. doi : 10.1016/0166-5316(82)90024-4. url : https://www.sciencedirect.com/science/article/pii/0166531682900244.

Lo Bello, Lucia et Wilfried Steiner (juin 2019). « A Perspective on IEEE Time-Sensitive Networking for Industrial Communication and Automation Systems ». In : *Proceedings of the IEEE* 107.6, p. 1094-1120. issn : 1558-2256. doi : 10.1109/JPROC.2019.2905334.

Mammeri, Zoubir (2019). « Reinforcement Learning Based Routing in Networks : Review and Classification of Approaches ». In : *IEEE Access* 7, p. 55916-55950. issn : 2169-3536. doi : 10.1109/ACCESS.2019.2913776.

Mao, Hongzi et al. (2016). « Resource Management with Deep Reinforcement Learning ». In : *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. HotNets '16. Atlanta, GA, USA : Association for Computing Machinery, p. 50-56. isbn : 9781450346610. doi : 10.1145/3005745.3005750. url : https://doi.org/10.1145/3005745.3005750.

Serna Oliver, Ramon, Silviu S. Craciunas et Wilfried Steiner (avr. 2018). « IEEE 802.1Qbv Gate Control List Synthesis Using Array Theory Encoding ». In : *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, p. 13-24. doi : 10.1109/RTAS.2018.00008.

Silva, Luis et al. (mai 2019). « On the adequacy of SDN and TSN for Industry 4.0 ». In : *2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC)*, p. 43-51. doi : 10.1109/ISORC.2019.00017.

Tesauro, Gerald (jan. 2007). « Reinforcement Learning in Autonomic Computing : A Manifesto and Case Studies ». In : *IEEE Internet Computing* 11.1, p. 22-30. issn : 1941-0131. doi : 10.1109/MIC.2007.21.

Tindell, Ken W, Alan Burns et Andy J. Wellings (1992). « Allocating hard real-time tasks : an NP-hard problem made easy ». In : *Real-Time Systems* 4.2, p. 145-165.

Xu, Xun et al. (2021). « Industry 4.0 and Industry 5.0—Inception, conception and perception ». In : *Journal of Manufacturing Systems* 61, p. 530-535. issn : 0278-6125. doi : 10.1016/j.jmsy.2021.10.006. url : https://www.sciencedirect.com/science/article/pii/S0278612521002119.

Première partie

Contexte technologique

Chapitre 1

Les délais de bout en bout

L'un des pré-requis aux industries du futur est que les communications doivent être déterministes. Un réseau déterministe ou des communications déterministes sont des systèmes dans lesquels au moins un sous-ensemble de la communication doit avoir un comportement en temps réel. En d'autres termes, cela veut dire que l'on est capable, pour un ensemble de flux considérés comme critique, de déterminer leurs latences maximales. Dans de tels systèmes, on est donc capable de prédire si ces flux pourront respecter leurs contraintes temporelles. Ce type de système est souvent utilisé dans les applications qui nécessitent une synchronisation stricte, une faible latence et une fiabilité élevée, telles que les systèmes en temps réel, les réseaux de contrôle industriels ou les réseaux de communication militaires. Ces systèmes sont conçus pour maintenir un niveau élevé de certitude et de prévisibilité afin d'assurer le bon fonctionnement des processus et des systèmes critiques.

Prenons l'exemple dépeint sur la Figure 1.1. Dans une usine 4.0/5.0, le contrôle est automatisé. On a alors un détecteur qui va déclencher une inspection du produit à son arrivée. Le détecteur va activer une caméra (dite « intelligente ») qui va se charger du contrôle du produit. Ces deux équipements (détecteur et caméra) sont reliés entre eux via le réseau, ainsi que beaucoup d'autres machines. Supposons qu'il s'agisse d'un réseau Ethernet. Si pour une raison quelconque, une congestion intervient sur le réseau, il est tout à fait possible que le message activant la caméra arrive à destination en retard, car le produit ne s'est pas arrêté. La caméra n'inspectera dans ce cas que l'arrière du produit ou pire, le tapis de convoyage. Il faut donc trouver un moyen de borner les délais de bout en bout (les latences) de certains flux.

Dans un réseau local basé sur Ethernet, le délai de bout en bout peut être défini comme le temps nécessaire pour qu'un paquet se déplace du périphérique source au périphérique de destination, y compris le temps passé dans les files d'attente et le temps passé à transmettre sur le réseau. Le délai de bout en bout dépend de quatre éléments :

1. Délai de transmission t_{trans} : le temps qu'il faut pour transmettre les données sur Ethernet. Il dépend de la vitesse de transmission du support v_{trans} et de la taille de la trame envoyée l_{trame}.
2. Délai de propagation t_{prop} : le temps qu'il faut pour qu'un signal se propage d'un nœud à un autre. Il dépend de la vitesse de propagation du média v_{prop} (par exemple, la vitesse de la lumière dans une fibre optique) et de la distance entre les nœuds d.
3. Délai de traitement t_{trait} : le temps nécessaire pour traiter les informations reçues au niveau des nœuds intermédiaires du réseau (commutateurs, routeurs, etc.). Il peut varier

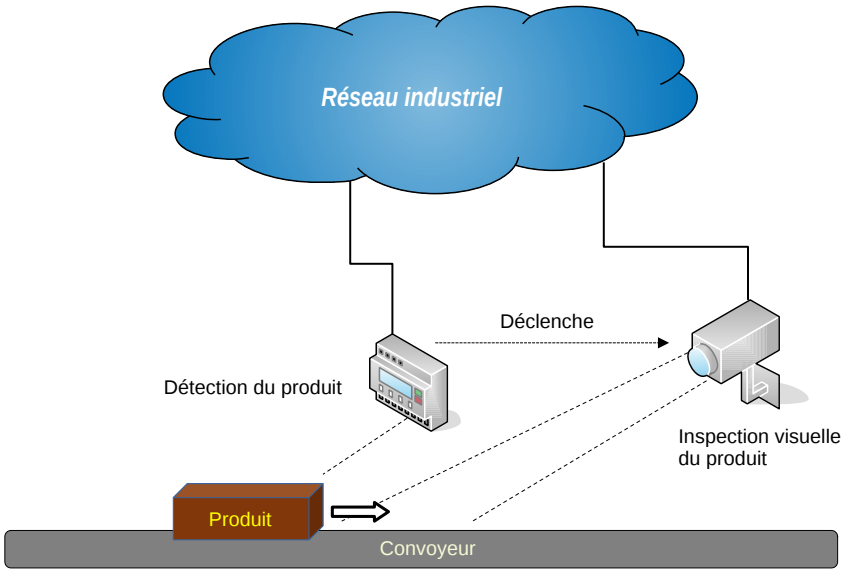


FIGURE 1.1 – Exemple de cas d'usage dans une usine 4.0/5.0

en fonction de la capacité de traitement de chaque nœud.

4. Délai de mise en file d'attente t_{file} : le temps qu'un paquet passe en attente dans une file d'attente avant d'être transmis. Ce délai dépend du trafic du réseau et de la stratégie de gestion de file d'attente utilisée.

Les trois premiers éléments sont des constantes du système, tandis que le quatrième peut subir de grandes variations entre deux trames envoyées par le même flux.

Prenons l'exemple dépeint sur la Figure 1.2. Dans cet exemple, un client communique vers un serveur via un commutateur. Les liens ont une vitesse de transmission $v_{trans} = 1$ Gbps et une vitesse de propagation $v_{prop} = 2 \times 10^8$ m/s. Le lien entre le client et le commutateur a une longueur $d_1 = 10$ m tandis que le lien entre le commutateur et le serveur fait $d_2 = 50$ m. Le client envoi des trames de longueur $l_{trame} = 1500$ octets.

On a donc :

— Le délai de transmission du paquet entre le client et le commutateur t_{trans1} :

$$\frac{\text{Taille de la trame (octets)}}{\text{Vitesse de transmission (bps)}} = \frac{l_{trame}}{v_{trans}} = \frac{1500 \times 8}{10^9} = 1,2 \times 10^{-5} \text{secondes}$$

— Le délai de propagation du paquet entre le client et le commutateur t_{prop1} :

$$\frac{\text{Distance (mètres)}}{\text{Vitesse de propagation (m/s)}} = \frac{d_1}{v_{prop}} = \frac{10}{2 \times 10^8} = 5 \times 10^{-8} \text{secondes}$$

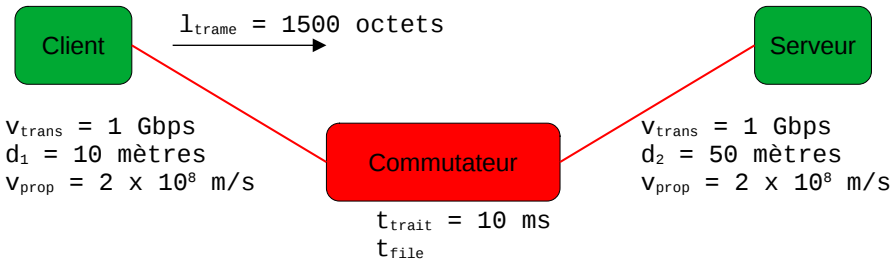


FIGURE 1.2 – Exemple de délais

— Le délai de traitement du commutateur t_{trait} est constant, il vaut : 10 ms
— Le délai de mise en file d'attente t_{file} est variable, il dépend :
  — du nombre de paquets dans la queue,
  — de l'intervalle d'arrivée des paquets,
  — du routage,
  — etc.
— Le délai de transmission entre le commutateur et le serveur t_{trans2} :

$$\frac{1500 \times 8}{10^9} = 1,2 \times 10^{-5} \text{secondes}$$

— Le délai de propagation entre le commutateur et le serveur t_{prop2} :

$$\frac{50}{2 \times 10^8} = 25 \times 10^{-8} \text{secondes}$$

Ce qui fait :

$$\text{Délai de bout en bout} = t_{trans1} + t_{prop1} + t_{trait} + t_{trans2} + t_{prop2} + t_{file} \quad (1.1)$$

$$= 1,2 \times 10^{-5} + 5 \times 10^{-8} + 0,01 + 1,2 \times 10^{-5} + 25 \times 10^{-8} + t_{file} \quad (1.2)$$

On ne peut pas jouer beaucoup sur les délais de traitement, de transmission et de propagation (autrement qu'en achetant du matériel plus performant) et de plus, ces délais sont déjà très courts. Par conséquent, il faut chercher à minimiser les délais dans les files d'attentes t_{file}. C'est ici qu'interviennent les standards TSN. Ces standards décrivent un certain nombre de mécanismes qui permettent de minimiser les délais dans les files d'attentes des commutateurs. Ce sont ces mécanismes que nous allons voir dans le chapitre suivant.

Chapitre 2

Time-Sensitive Networking

Sommaire du présent chapitre

L'effort de standardisation lié à TSN a pour objectif de fournir une solution temps-réel basée sur Ethernet. En effet, le développement d'applications critiques dans l'industrie a nécessité le développement de technologies permettant des communications en temps réel fiables et déterministes, basées sur Ethernet, afin de garantir des opérations synchronisées et coordonnées. Dans le but de répondre à cet enjeu, plusieurs solutions ont émergées, comme PROFINET (IEC 61158 et IEC 61784), TTEthernet (SAE AS6802) ou EtherCAT (IEC 61158). Le résultat est un ensemble de standards, ayant des champs d'application différents (automatisation industrielle, aérospatial, etc) et surtout incompatibles entre eux. La venue de l'industrie 4.0, où tous les équipements seront inter-connectés, a rendu nécessaire une normalisation plus globale et l'interopérabilité entre ces technologies. TSN permet de remédier à cette situation en fournissant un socle commun à toute application nécessitant du temps-réel basé sur Ethernet.

TSN est un ensemble d'une trentaine de standards [1] et d'amendements (publiés ou en cours d'élaboration, voir 2.1) pour la mise en place de transmissions, sensibles au temps, de paquets sur des réseaux Ethernet déterministes. Ils sont édictés par l'Institute of Electrical and Electronics Engineers (IEEE). L'IEEE est une organisation, basée aux États-Unis, qui promeut la connaissance et la recherche dans de nombreux domaines technologiques dont les télécommunications. En ce qui concerne les télécommunications, l'IEEE est spécialisé dans les technologies liées aux couches matérielles (couches 1 et 2) du modèle Open Systems Interconnection (OSI).

1. En français (et en Europe), on distingue les normes des standards : voir le guide *PME : pensez à allier propriété intellectuelle et normalisation !* à ce sujet, disponible par exemple à l'adresse : `http://www.eurosfaire.prd.fr/7pc/doc/1279533531_guide_pi_normalisation.pdf`. En résumé, l'IEEE édite des standards et l'American National Standards Institute (ANSI) en fait des normes.

L'institut est très actif en ce qui concerne la publication de standards. Le comité IEEE 802 en particulier s'occupe des standards relatifs aux réseaux locaux et métropolitains (on parle des réseaux 802) [2]. En son sein se trouve le groupe de travail IEEE 802.1, qui s'occupe du développement des standards ayant trait aux thèmes suivants :

— architecture des réseaux locaux et métropolitains,
— interconnexion entre les réseaux 802,
— sécurité des réseaux 802,
— gestion des réseaux 802,
— protocoles au-dessus de la couche liaison du modèle OSI.

Le groupe de travail IEEE 802.1 est lui-même décomposé en plusieurs sous-groupes de travail. L'un d'entre eux s'occupe du développement des standards TSN.

Ce chapitre commence par donner un aperçu des standards TSN. Ensuite, il présente plus en détail les deux standards TSN qui ont été considérés au cours de cette thèse : IEEE 802.1Q et IEEE 802.1AS.

2.1 Les standards TSN

Le but de TSN est de fournir des services déterministes dans des réseaux locaux (en anglais Local Area Network (LAN)). Un LAN est un réseau informatique (composé de terminaux comme des ordinateurs, mais aussi dans notre cas des machines industrielles) circonscrit physiquement à une zone géographique précise, par exemple une usine ou une maison. En particulier, deux machines du même LAN communiquent entre elles sans passer par internet. Dans le même ordre d'idée, un réseau local virtuel (en anglais Virtual Local Area Network (VLAN)) est un réseau informatique circonscrit logiquement (et non plus physiquement). Il permet aux administrateurs de réseau de segmenter un réseau en différents groupes, indépendamment de l'emplacement physique des appareils. Les appareils d'un même VLAN peuvent communiquer entre eux comme s'ils étaient connectés au même réseau physique, même s'ils sont situés dans des parties différentes de l'infrastructure du réseau. Les VLAN sont créés en configurant les commutateurs et les routeurs de manière à définir quels ports appartiennent à quels VLAN.

La Figure 2.1 montre en détail l'organisation des standards TSN en fonction de cette décomposition. Les standards TSN peuvent être décomposés en 4 catégories :

1. la synchronisation du temps,
2. la fiabilité,
3. les temps de latences garantis,
4. la gestion des ressources.

En pratique, les différents standards TSN peuvent être combinés entre eux ((Migge et al. 2018)). Les articles cités dans (Farkas, Bello et Gunther 2018), en particulier (Finn 2018) et (Messenger 2018), fournissent une introduction aux standards TSN en général ainsi que des exemples de cas d'applications plus détaillés. Dans (Nasrallah, Thyagaturu et al. 2019), les auteurs présentent un aperçu des standards TSN dans un contexte technologique récent. À noter que, comme on peut le voir sur la figure 2.1, il existe aussi des profils TSN. Ces profils (qui sont aussi des standards IEEE) spécifient, dans un cas d'application précis, quels standards TSN et fonctionnalités utiliser et comment. Un profil qui sera intéressant pour l'industrie quand il sera finalisé est le profil IEC/IEEE 60802 - *TSN Profile for Industrial Automation*, qui traitera de l'automatisation industrielle. Ce profil définira les exigences et les spécifications

2. Plus d'information sur le comité IEEE 802 : https://www.ieee802.org/
3. Image issue de https://1.ieee802.org/tsn/

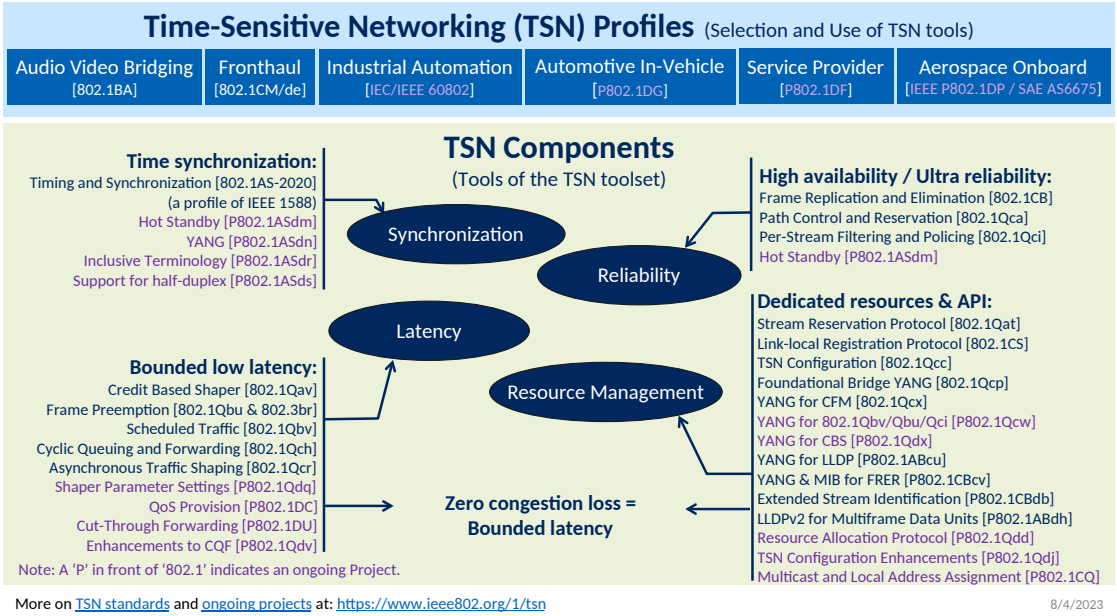


FIGURE 2.1 – Organisation des standards TSN en août 2023 [3]

(les caractéristiques, les options, les configurations, les valeurs par défaut, les protocoles, les procédures, etc) pour la communication en temps-réel entre les différents dispositifs et systèmes utilisés dans l'automatisation industrielle, tels que les automates programmables, les capteurs, les actionneurs, etc.

Dans la suite de ce chapitre, nous allons voir les standards TSN permettant d'atteindre notre objectif : ordonnancer le trafic afin de protéger certains flux.

2.2 IEEE 802.1Q - IEEE Standard for Local and Metropolitan Area Network – Bridges and Bridged Networks

Le standard IEEE 802.1Q - *IEEE Standard for Local and Metropolitan Area Network – Bridges and Bridged Networks* (« IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks » 2022) définit une méthode d'encapsulation des trames où l'en-tête de trame est complétée par une balise de VLAN (VLAN tag) sur 4 octets (32 bits), comme illustré sur la Figure 2.2. La balise de VLAN inclut deux champs, encodés chacun sur deux octets :

1. *Tag Protocol Identifier* (TPID) : permet d'identifier la trame comme une trame IEEE 802.1Q.
2. *Tag Control Information* (TCI) : champ contenant les informations suivantes :
   (a) Priority Code Point (PCP) : encodé sur 3 bits, permet de définir des classes de trafic en fonction de priorités (entre 0 et 7).
   (b) Drop Eligible Indicator (DEI) : encodé sur 1 bit, indique les trames pouvant être jetées en cas de congestion.
   (c) VLAN Identifier (VID) : encodé sur 12 bits, spécifie le VLAN auquel la trame appartient.

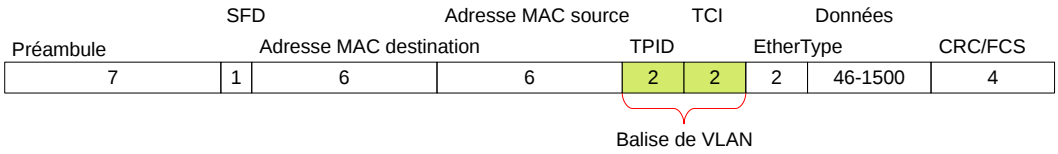


FIGURE 2.2 – Une trame 802.1Q, avec la taille de chaque champ en octets. En vert, les champs ajoutés par le standard IEEE 802.1Q

Le champ PCP est celui qui permet d'introduire une notion de qualité de service dans le réseau. Le Tableau 2.1 montre la mise en correspondance des différentes classes de trafic avec les PCP telles que définies dans l'annexe I du standard IEEE 802.1Q (« IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks » 2022). Il peut y avoir jusqu'à 8 classes de trafic et par conséquent, 8 PCP différents. À noter que la priorité 0 est la priorité par défaut, de même, le trafic de type meilleur effort (en anglais Best Effort (BE)) est le type de trafic par défaut. De ce fait, la priorité 0 est assignée au trafic meilleur effort, tandis que la priorité 1 est assignée au trafic en arrière-plan, moins prioritaire.

Priorité	Type de trafic
1	arrière-plan
0 (par défaut)	meilleur effort (par défaut)
2	excellent effort
3	application critique
4	vidéo (latence et gigue < 100ms)
5	voix (latence et gigue < 10ms)
6	Administration inter-réseau
7	Administration réseau

TABLEAU 2.1 – Mise en correspondance des priorités avec le type de trafic

Afin de pouvoir exploiter les possibilités offertes par le champ PCP, il est nécessaire qu'il y ait au sein des commutateurs, pour chaque interface ou port, au moins autant de queues (logiques) qu'il y a de classes de trafic (jusqu'à 8, donc). Chaque classe de trafic sera alors affectée à une queue.

Le standard IEEE 802.1Q a subi de nombreuses modifications via différents amendements. Ces amendements font maintenant pleinement partie du standard. Un de ces amendements traite de la manière d'ordonnancer le trafic.

2.2.1 IEEE 802.1Qbv - Amendment 25 : Enhancements for Scheduled Traffic

L'amendement IEEE 802.1Qbv - *Amendment 25 : Enhancements for Scheduled Traffic* (« IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks - Amendment 25 : Enhancements for Scheduled Traffic » 2016) présente différents mécanismes permettant d'ordonnancer le trafic. L'un d'entre eux s'appelle TAS. Il s'agit d'un mécanisme similaire dans l'idée au protocole Time-Division Multiple Access (TDMA). Le temps de transmission est divisé en cycles de durée constante. Ce cycle est lui-même divisé en séquences de temps de durées variables. Puis, on définit dans quelles séquences une trame est autorisée à être envoyée en

fonction de sa classe de trafic. Dit autrement, on définit pour chaque queue une porte, qui peut être soit ouverte, soit fermée. Le trafic est géré à partir d'une liste, appelée Gate Control List (GCL), qui définit quelles classes de trafic sont autorisées (quelles portes sont ouvertes) et pendant combien de temps (à quel moment on ouvre/ferme une porte). Quand plusieurs classes de trafic sont transmises en même temps, c'est la priorité qui détermine l'ordre de transmission. La Figure 2.3 illustre ce mécanisme au sein d'un commutateur TSN. Dans cet exemple, deux flux arrivent dans le commutateur : le flux A par le port 1, le flux B par le port 2. Le flux A a une priorité de 6 et est donc redirigé vers la queue 6, le flux B a une priorité de 7 et va dans la queue 7. Dans la GCL, à t_0, la porte est ouverte pour la queue 7 et le flux B peux passer. À t_1, la porte pour la queue 7 se ferme et celle pour la queue 4 s'ouvre, et ainsi de suite jusqu'à t_n, où les portes ouvertes sont celles devant les queues 6 et 4. On repart alors à t_0, et le cycle se répète.

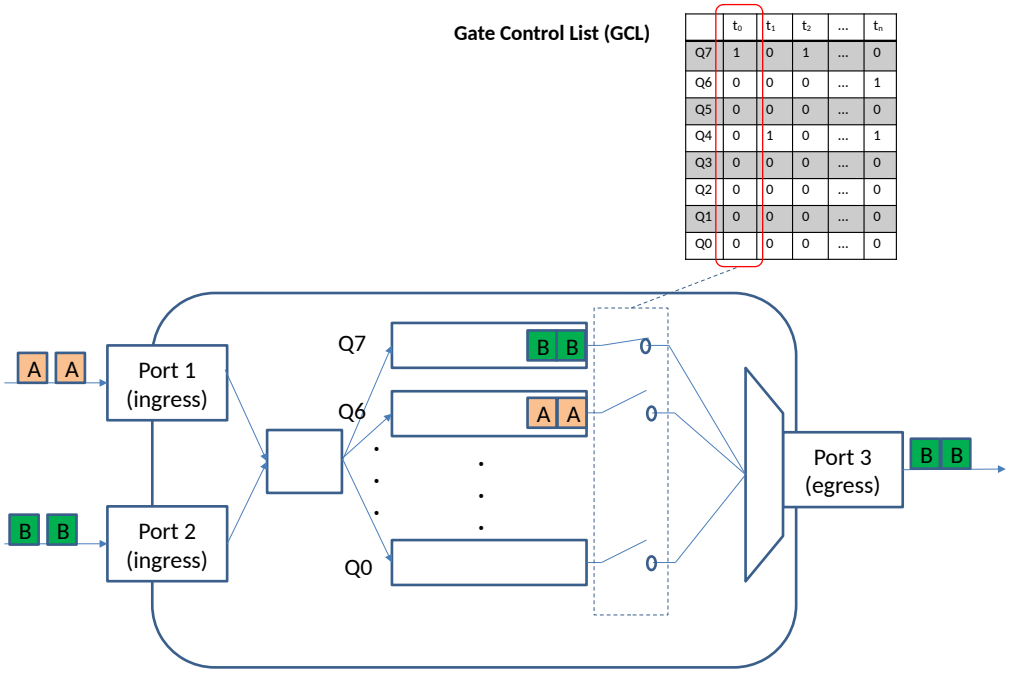


FIGURE 2.3 – Le mécanisme d'ordonnancement TAS

Le TAS doit être configuré sur chaque commutateur TSN du réseau, comme montré sur la Figure 2.4. Les paramètres importants afin de configurer le TAS sont, sur chaque commutateur du réseau :

1. la durée du cycle,
2. la durée des différentes séquences de temps,
3. l'assignation du trafic à ces séquences de temps,
4. le Maximum Service Data Unit (MSDU), c.-à-d., la taille de la plus grande trame pouvant être envoyée (il y en a une par queue).

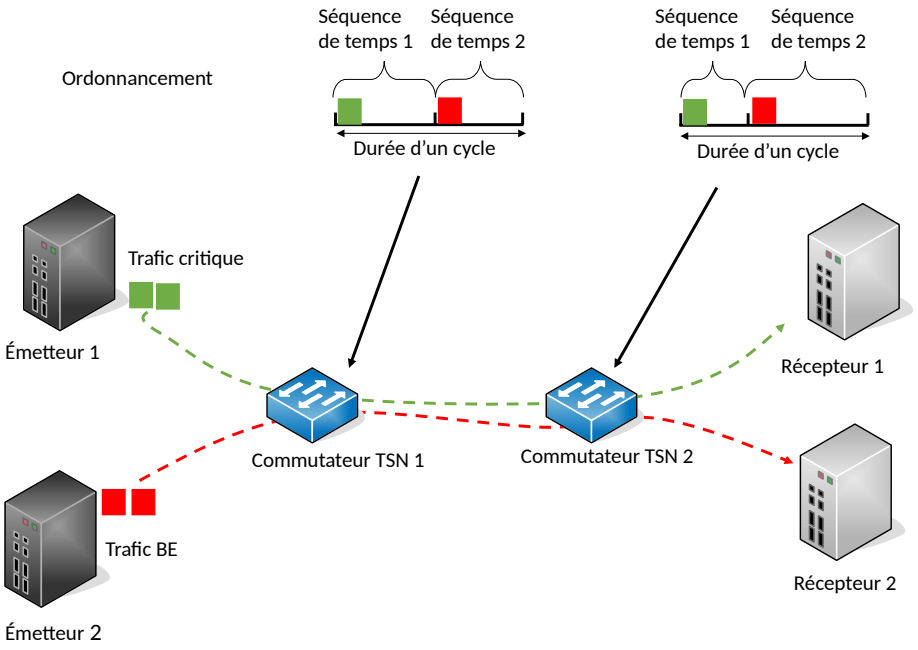


FIGURE 2.4 – Exemple illustratif de 2 émetteurs (Émetteur 1 pour le trafic critique et Émetteur 2 pour le trafic BE) échangeant avec 2 récepteurs (Récepteurs 1 pour le trafic critique et Récepteurs 2 pour le trafic BE) via un réseau TSN linéaire composé de deux commutateurs TSN

Afin d'éviter qu'une transmission ne déborde sur la séquence de temps suivante (surtout si celle-ci est allouée à du trafic critique), on peut protéger certaines séquences en ajoutant avant celles-ci une séquence supplémentaire (appelée *guard band* en anglais), pendant laquelle aucune nouvelle transmission de trame Ethernet ne peut être démarrée, seules les transmissions déjà en cours peuvent être terminées (Figure 2.5). La durée de cette bande de protection est alors égale à la durée nécessaire pour transmettre la plus grande trame devant être envoyée. Cette méthode a cependant l'inconvénient de faire globalement baisser la bande passante, étant donné qu'il y a des moments où aucun trafic n'est envoyé.

Le calcul de l'ordonnancement du trafic dans les commutateurs dépend de plusieurs paramètres :

— la topologie,
— le routage,
— les flux (taille des paquets, intervalle d'émission),
— les contraintes liées aux flux.

Tout cela rend l'ordonnancement difficile.

Dans cette thèse, nous allons nous concentrer sur le calcul de configurations pour le TAS. Ainsi, il est important d'avoir un aperçu des différentes architectures de configurations prévues dans les standards TSN.

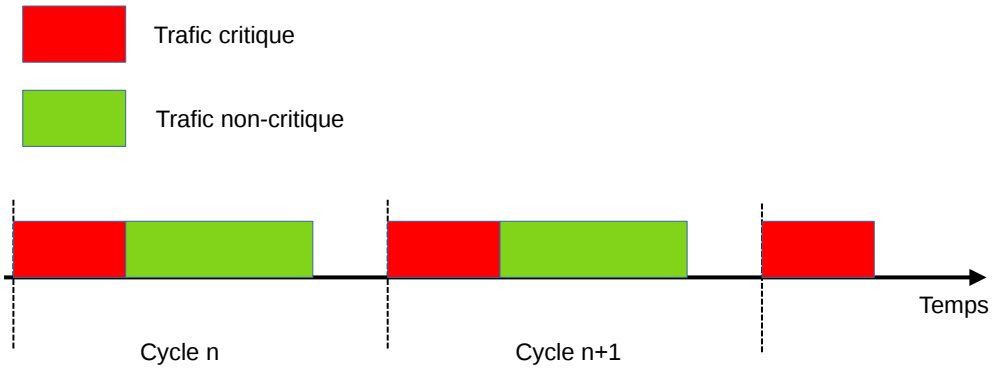


FIGURE 2.5 – Mécanisme de protection des flux critiques

2.2.2 IEEE 802.1Qcc - Amendment 31 : Stream Reservation Protocol (SRP) Enhancements and Performance Improvements

L'amendement IEEE 802.1Qcc - *Amendment 31 : Stream Reservation Protocol (SRP) Enhancements and Performance Improvements* (« IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks – Amendment 31 : Stream Reservation Protocol (SRP) Enhancements and Performance Improvements » 2018) traite de la gestion des ressources réseau. Il définit (entre autres) trois architectures permettant de configurer des réseaux TSN. Ces trois architectures sont respectivement :

1. entièrement distribuée,
2. réseau centralisé/application distribuée,
3. entièrement centralisée.

Dans une architecture entièrement distribuée, les stations finales (émetteurs et récepteurs) communiquent les besoins des applications directement via le réseau, comme illustré sur la Figure 2.6. Le réseau est configuré de manière distribuée, sans entité de configuration de réseau centralisée. La configuration distribuée du réseau est effectuée à l'aide d'un protocole qui propage les informations de configuration TSN le long de la topologie active pour le flux. Comme les besoins des applications se propagent à travers le réseau, la gestion de chaque commutateur est effectuée localement. Cette gestion locale se limite à l'information dont le commutateur a connaissance et n'inclut pas nécessairement la connaissance de l'ensemble du réseau.

L'architecture réseau centralisé/application distribuée est similaire à celle entièrement distribuée dans la mesure où les stations communiquent leurs exigences directement via le réseau. En revanche, dans cette architecture, les informations de configuration sont dirigées vers/depuis une entité de configuration réseau centralisée (Centralized Network Configuration (CNC)), comme on peut le voir sur la Figure 2.7. Toute la configuration des commutateurs TSN est effectuée par cette entité CNC. Cette dernière a une vue complète de la topologie physique du réseau ainsi que des capacités de chaque commutateur. Cela lui permet de centraliser des calculs complexes. La CNC peut exister dans une station d'extrémité ou un pont.

Enfin, la dernière architecture est celle entièrement centralisée, illustrée sur la Figure 2.8. La configuration réseau centralisée CNC est l'entité qui est responsable de la configuration

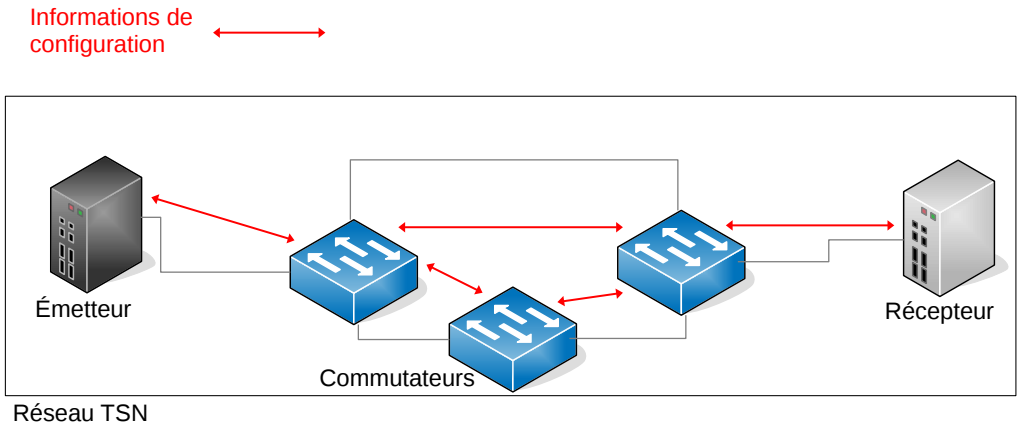


FIGURE 2.6 – Architecture entièrement distribuée

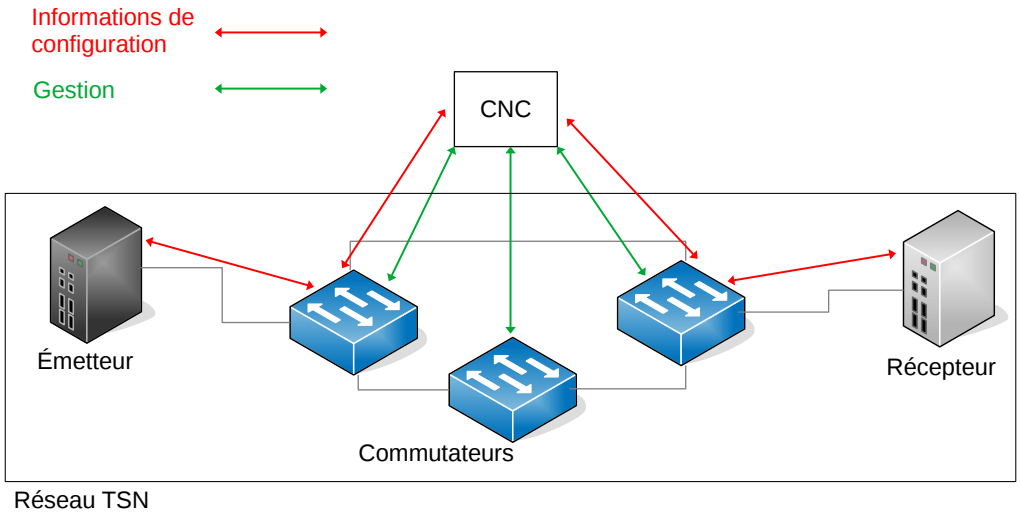


FIGURE 2.7 – Architecture réseau centralisé/application distribuée

des ressources du réseau pour le compte des applications TSN. La CNC gère des informations réseau. La configuration utilisateur centralisée (Centralized User Configuration (CUC)) est l'entité qui identifie les stations finales, en récupère les capacités et les besoins applicatifs et y configure les fonctionnalités TSN. Elle traite des informations applicatives. Un protocole de transmission doit être utilisé entre la CUC et les différentes stations. Étant donné que ce protocole est application à application, ses informations de configuration sont considérées comme hors du champ d'application du standard. Le protocole OPC Unified Architecture (OPC UA) est un

candidat potentiel. La CUC échange des informations avec la CNC afin de pouvoir configurer les options TSN sur les stations. Ce modèle est dit *complètement centralisé*, puisque la CUC contrôle tout, même les stations.

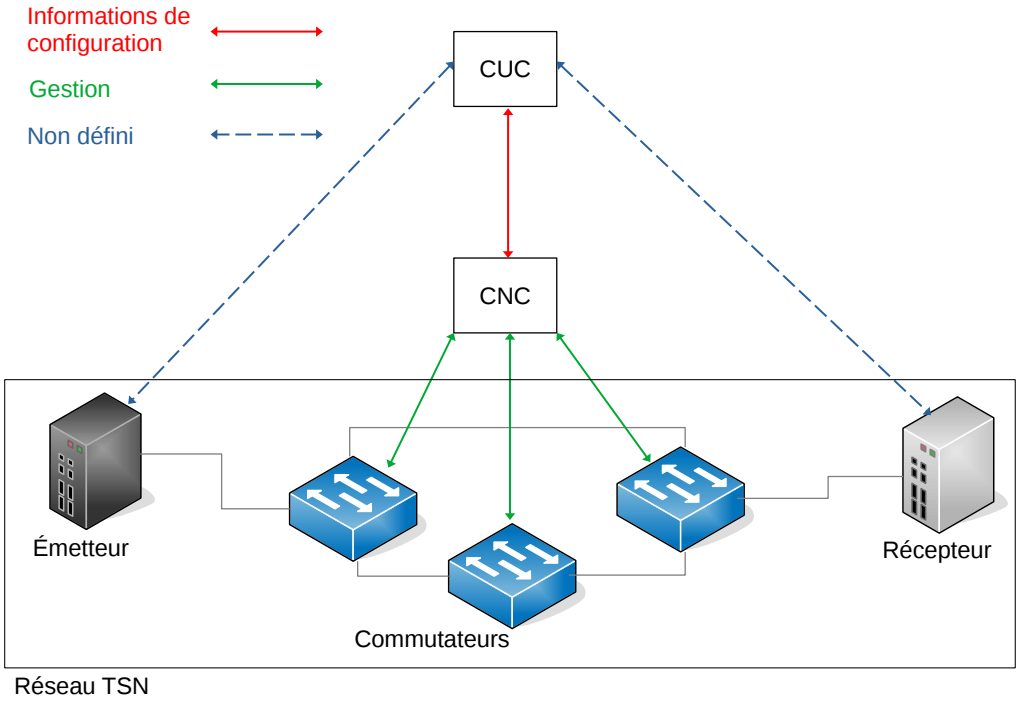


FIGURE 2.8 – Architecture entièrement centralisée (en rouge, non défini dans le standard)

L'architecture entièrement distribuée est adaptée à la configuration du mécanisme TSN Credit-Based Shaper (CBS), tandis que les deux autres architectures peuvent être utilisées pour configurer plus de mécanismes, dont le TAS. En effet, une connaissance centralisée de la topologie du réseau, comme au sein de la CNC, permet le calcul et la mise en œuvre des ordonnancements plus facilement. Ces deux architectures sont donc celle appropriée afin de configurer le TAS, cependant, l'architecture entièrement centralisée est souvent préférée, en particulier, car elle s'adapte bien au paradigme Software-Defined Networking (SDN). De ce fait, c'est au sein d'une telle architecture que notre agent de configuration du TAS prendra place.

Il ne manque plus qu'une chose afin que le TAS puisse fonctionner : une notion commune du temps.

2.3 IEEE 802.1AS - IEEE Standard for Local and Metropolitan Area Networks – Timing and Synchronization for Time-Sensitive Applications

Le standard IEEE 802.1AS - *IEEE Standard for Local and Metropolitan Area Networks – Timing and Synchronization for Time-Sensitive Applications*(« IEEE Standard for Local and Metropolitan Area Networks–Timing and Synchronization for Time-Sensitive Applications » 2020) définit une adaptation du Precision Time Protocol (PTP) (définit dans (« IEC/IEEE International Standard - Precision Clock Synchronization Protocol for Networked Measurement and Control Systems » 2021)), pour les réseaux TSN. Cette adaptation est parfois appelée generalized Precision Time Protocol (gPTP). Ce protocole permet de synchroniser les horloges de systèmes sensibles au temps à travers un réseau.

Dans les faits, les horloges des différents périphériques réseau peuvent différer les unes des autres. Dans les applications sensibles au temps, cette dérive peut être réduite grâce à la synchronisation du temps. Dire que le temps est synchronisé entre deux horloges signifie que la différence de temps entre elles est bornée. On peut y parvenir en synchronisant périodiquement le temps entre les horloges (c'est-à-dire avant que la différence de temps ne devienne trop importante). Montrer que la dérive d'horloge est bornée peut être un problème compliqué (Bailleul 2023).

gPTP est basé sur le paradigme maître-esclave. L'heure de plusieurs horloges esclaves est synchronisée avec l'heure d'une horloge maîtresse au sein d'un domaine temporel gPTP. Un réseau peut avoir plusieurs domaines temporels gPTP, c'est-à-dire qu'un nœud peut synchroniser plusieurs horloges esclaves avec plusieurs horloges maîtresses à des fins de redondance (si l'une des horloges maîtresses tombe en panne ou se déconnecte en raison d'un échec de connexion, par exemple, les différents nœuds pourront toujours être synchronisés). Chaque domaine temporel possède une horloge maîtresse et un certain nombre d'horloges esclaves. Le système synchronise les horloges esclaves avec l'horloge maîtresse en envoyant des messages de synchronisation du nœud d'horloge maître au nœud d'horloge esclave. L'horloge maîtresse est choisie grâce à l'algorithme Best Master Clock Algorithm (BCMA), défini dans (« IEC/IEEE International Standard - Precision Clock Synchronization Protocol for Networked Measurement and Control Systems » 2021). BCMA détermine également la manière dont les informations de synchronisation sont propagées vers les horloges esclaves du réseau.

Pour configurer le TAS, il est nécessaire que le gPTP soit mis en place. En effet, ordonnancer le trafic en fonction du temps nécessite que tous les appareils du réseau aient la même définition du temps. Si un commutateur n'est pas synchronisé aux autres nœuds, son ordonnancement sera décalé par rapport aux autres, ce qui aura un impact sur tous les flux qui le traversent.

2.4 Conclusion

La combinaison des standards TSN rend la configuration d'un réseau TSN difficile, car ces interactions ne sont pas forcément triviales. D'autant qu'il y a toujours un plus grand nombre de configurations possibles (l'ensemble de toutes les configurations) que de configurations faisables (qui permettent de respecter les contraintes applicatives). En particulier, la question de la reconfiguration de l'ordonnancement en ligne dans TSN demeure un défi de taille et ouvert (Stüber et al. 2023). Une partie du problème vient de ce que le simple fait de définir l'ordonnancement sur chaque appareil du réseau conduit à un problème NP-difficile (Serna Oliver, Silviu S. Craciunas et Steiner 2018 ; Silviu S. Craciunas et al. 2016 ; Tindell, Burns

et Wellings 1992 ; Burns 1991 ; Leung et Whitehead 1982). Une solution possible consiste à utiliser les techniques de l'IA, comme nous allons le voir dans les chapitres suivants.

Bibliographie du présent chapitre

Bailleul, Quentin (nov. 2023). « Dimensioning TSN network synchronization in different embedded contexts ». Theses. Institut National Polytechnique de Toulouse - INPT. URL : https://theses.hal.science/tel-04400599.

Burns, Alan (1991). « Scheduling hard real-time systems : a review ». In : *Software Engineering Journal* 6.3, p. 116-128.

Craciunas, Silviu S. et al. (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*. RTNS '16. Brest, France : Association for Computing Machinery, p. 183-192. ISBN : 9781450347877. DOI : 10.1145/2997465.2997470. URL : https://doi.org/10.1145/2997465.2997470.

Farkas, Janos, Lucia Lo Bello et Craig Gunther (juin 2018). « Time-Sensitive Networking Standards ». In : *IEEE Communications Standards Magazine* 2.2, p. 20-21. ISSN : 2471-2833. DOI : 10.1109/MCOMSTD.2018.8412457.

Finn, Norman (juin 2018). « Introduction to Time-Sensitive Networking ». In : *IEEE Communications Standards Magazine* 2.2, p. 22-28. ISSN : 2471-2833. DOI : 10.1109/MCOMSTD.2018.1700076.

« IEC/IEEE International Standard - Precision Clock Synchronization Protocol for Networked Measurement and Control Systems » (juin 2021). In : *IEC/IEEE 61588-2021*, p. 1-504. DOI : 10.1109/IEEESTD.2021.9456762.

« IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks - Amendment 25 : Enhancements for Scheduled Traffic » (mars 2016). In : *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)*, p. 1-57. DOI : 10.1109/IEEESTD.2016.8613095.

« IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks » (déc. 2022). In : *IEEE Std 802.1Q-2022 (Revision of IEEE Std 802.1Q-2018)*, p. 1-2163. DOI : 10.1109/IEEESTD.2022.10004498.

« IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks – Amendment 31 : Stream Reservation Protocol (SRP) Enhancements and Performance Improvements » (oct. 2018). In : *IEEE Std 802.1Qcc-2018 (Amendment to IEEE Std 802.1Q-2018 as amended by IEEE Std 802.1Qcp-2018)*, p. 1-208. DOI : 10.1109/IEEESTD.2018.8514112.

« IEEE Standard for Local and Metropolitan Area Networks–Timing and Synchronization for Time-Sensitive Applications » (juin 2020). In : *IEEE Std 802.1AS-2020 (Revision of IEEE Std 802.1AS-2011)*, p. 1-421. DOI : 10.1109/IEEESTD.2020.9121845.

Leung, Joseph Y.-T. et Jennifer Whitehead (1982). « On the complexity of fixed-priority scheduling of periodic, real-time tasks ». In : *Performance Evaluation* 2.4, p. 237-250. ISSN : 0166-5316. DOI : 10.1016/0166-5316(82)90024-4. URL : https://www.sciencedirect.com/science/article/pii/0166531682900244.

Messenger, John L. (juin 2018). « Time-Sensitive Networking : An Introduction ». In : *IEEE Communications Standards Magazine* 2.2, p. 29-33. ISSN : 2471-2833. DOI : 10.1109/MCOMSTD.2018.1700047.

Migge, Jörn et al. (2018). « Insights on the Performance and Configuration of AVB and TSN in Automotive Ethernet Networks ». In : *Proc. Embedded Real-Time Software and Systems (ERTS 2018)*.

Nasrallah, Ahmed, Akhilesh S. Thyagaturu et al. (2019). « Ultra-Low Latency (ULL) Networks : The IEEE TSN and IETF DetNet Standards and Related 5G ULL Research ». In : *IEEE Com-*

munications Surveys & Tutorials 21.1, p. 88-145. ISSN : 1553-877X. DOI : 10.1109/COMST.2018.2869350.

Serna Oliver, Ramon, Silviu S. Craciunas et Wilfried Steiner (avr. 2018). « IEEE 802.1Qbv Gate Control List Synthesis Using Array Theory Encoding ». In : *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, p. 13-24. DOI : 10.1109/RTAS.2018.00008.

Stüber, Thomas et al. (2023). « A Survey of Scheduling Algorithms for the Time-Aware Shaper in Time-Sensitive Networking (TSN) ». In : *IEEE Access* 11, p. 61192-61233. ISSN : 2169-3536. DOI : 10.1109/ACCESS.2023.3286370.

Tindell, Ken W, Alan Burns et Andy J. Wellings (1992). « Allocating hard real-time tasks : an NP-hard problem made easy ». In : *Real-Time Systems* 4.2, p. 145-165.

Chapitre 3

Apprentissage automatique

Sommaire du présent chapitre

L'apprentissage automatique (en anglais Machine Learning (ML)) est un sous-domaine de l'IA dont le but est de donner à un ordinateur la capacité d'apprendre à partir de données. Le besoin est né de la constatation que les systèmes experts utilisés jusque-là en IA étaient en train de devenir beaucoup trop compliquées. Un système expert est un programme informatique capable de résoudre des problèmes en utilisant des connaissances spécialisées dans un domaine donné. Il repose sur une base de connaissances (des règles logiques) préalablement codifiée (par un expert humain) et sur un ensemble de règles d'inférence qui permettent de traiter les informations et d'apporter des réponses ou des solutions adaptées aux problèmes posés par les utilisateurs (par exemple, à quelle(s) maladie(s) correspond(ent) tels symptômes ?). Lorsque le nombre de règles ou de cas à traiter devient trop élevé, cette méthode montre ses limites. D'où l'idée d'acquérir ces connaissances directement à partir des données (Tesauro 2007). À noter qu'une IA entrainée en suivant un paradigme de ML sera (potentiellement) très bonne dans son domaine, mais, à la différence d'un être humain, elle sera absolument incapable d'effectuer une autre tâche.

Dans ce chapitre, nous allons voir un aperçu des différents paradigmes de ML. En particulier, nous parlerons de l'apprentissage profond, une méthode qui sert souvent de support à tous ces paradigmes.

3.1 Paradigmes d'apprentissage

Tous les paradigmes d'apprentissage de ML ont en commun le fait que le système qui les implante passe en général par deux phases : la première est l'apprentissage (l'entrainement), la seconde est l'inférence (l'application). Il existe 3 types d'apprentissages différents :

— supervisé ;
— non supervisé ;
— par renforcement.

À ces trois paradigmes s'ajoute parfois l'apprentissage semi-supervisé.

3.1.1 Apprentissage supervisé et non supervisé

La plupart des paradigmes de ML sont basés sur l'exploitation de données. La première étape, commune à tous ces paradigmes, consiste à trier ces données (brutes) pour en extraire des propriétés ou des caractéristiques sur lesquelles on puisse faire des analyses ou des prédictions. Cette étape s'appelle en anglais *Feature Engineering*.

Quand ces données sont étiquetées par un expert (un oracle), c'est-à-dire que l'on peut connaître à l'avance quel sera le comportement que l'on cherche à prédire (telle image représente un chat, par exemple), on parle *d'apprentissage supervisé*. Dans ce type d'apprentissage, l'IA va chercher à apprendre un modèle ou un comportement à partir de données dont la nature est connue. Cette façon de faire convient bien à des problèmes de classification (classer des données dans des catégories) et de régression (prédire une valeur en fonction de paramètres divers).

Quand les données ne sont pas étiquetées, il s'agit *d'apprentissage non supervisé*. L'IA va chercher à trouver des similarités entre ces données afin de les regrouper entre elles. Cette approche convient bien aux problèmes de partitionnement (regrouper des données en fonction de similarités) et de détection d'anomalies (détecter une donnée très différente des autres).

Enfin, il existe des techniques *d'apprentissage semi-supervisé* lorsque certaines étiquettes ne sont pas disponibles. Cette méthode est un compromis entre les avantages et inconvénients des deux précédentes.

Les techniques d'apprentissages automatiques sont déjà très répandues dans les réseaux informatiques traditionnels (M. Wang et al. 2018 ; Boutaba et al. 2018 ; Ahmad et al. 2020). Ses champs d'applications vont du routage à la sécurité en passant par la qualité de service (contrôle des congestions, etc).

Toutes ces techniques d'apprentissage semblent ne pas convenir a priori pour configurer le mécanisme TAS des réseaux TSN. En effet, on ne cherche pas à travailler sur des jeux de données, mais plutôt à travailler et agir sur un environnement.

3.1.2 Apprentissage par renforcement

Le cas de *l'apprentissage par renforcement* (en anglais Reinforcement Learning (RL)) est différent. Dans le RL, un *agent* interagit avec un *environnement*. L'idée est que l'agent va faire des *observations* de l'environnement et lui appliquer des *actions*, qui vont déterminer une *récompense*. Le but de l'agent est de maximiser ses récompenses *sur le long terme*. La principale différence avec les techniques mentionnées plus haut est que le RL ne s'appuie pas sur des données, mais sur des retours d'expériences.

Le RL est la technique d'apprentissage automatique adaptée à la prise de décision. Dans le cas de la configuration de l'ordonnancement dans TSN, c'est donc l'apprentissage par renforcement qui semble le plus prometteur. En effet, l'objectif est bien de prendre une décision (changer la

configuration) lors d'un évènement donné (changement de topologie, nouveaux flux applicatifs, etc.).

Le fonctionnement du RL ainsi que les algorithmes utilisés sont détaillés dans le chapitre suivant (Chapitre 4). Une famille d'algorithme de RL, appelée Deep Reinforcement Learning (DRL), utilise l'apprentissage profond afin de faire des approximations de fonctions. Ce sont ces algorithmes que nous allons utiliser. La section suivante présente les fondamentaux de l'apprentissage profond.

3.2 Apprentissage profond

L'apprentissage profond (Goodfellow, Bengio et Courville 2016 ; Pouyanfar et al. 2018) (en anglais *Deep Learning*) est une méthode d'apprentissage utilisée par les différents paradigmes précédents. Historiquement, il s'agissait de s'inspirer du fonctionnement du cerveau humain afin de faire apprendre des machines, d'où l'utilisation du terme de « neurones ». À l'époque, dans les années 1940-1960, on parlait de « cybernétique », puis, en 1980–1995 de « connexionnisme », et enfin, depuis 2006, le terme utilisé est celui de Réseaux de Neurones Artificiels (RNA). Les RNA (ou plus simplement *réseaux de neurones*) désignent l'ensemble des architectures d'apprentissage profond. Aujourd'hui, il n'y a plus grand-chose en commun avec le fonctionnement du cerveau tant et si bien que l'on remplace parfois le terme de « neurones » par celui « d'unités ». On parle « d'apprentissage profond » pour désigner cet ensemble de méthodes, car de nos jours, la tendance est d'ajouter de la profondeur (c.-à-d., avoir beaucoup de couches) plutôt que de la largeur (c.-à-d., avoir beaucoup de neurones par couches) aux réseaux de neurones. L'apprentissage profond est principalement utilisé dans l'apprentissage supervisé et non supervisé : c'est un moyen (parmi d'autres) de calculer la probabilité qu'un objet (donné en entrée) appartienne à une classe. Il est cependant aussi utilisé dans certain des algorithmes d'apprentissages par renforcement.

Pour comprendre le fonctionnement des réseaux de neurones, le plus simple est de revenir aux origines (et à l'architecture la plus simple) : le *perceptron*.

3.2.1 Perceptron

Un perceptron est composé d'une seule couche de neurones, située entre les entrées et la sortie (Figure 3.1). Les entrées lisent les données, elle est composée d'autant d'entrées que de variables d'entrée. À ces entrées peut s'ajouter un neurone de biais, dont la valeur est toujours 1. La couche de neurones reçoit la somme des valeurs des d neurones qui lui sont reliés, pondérée par des poids de connexion, c'est-à-dire :

$$w_b + \sum_{i=1}^{d} w_i x_i \tag{3.1}$$

avec w_b le poids lié au biais. Ce neurone applique alors une *fonction d'activation* à cette somme, définie en fonction du problème, ce qui donne un résultat (par exemple, 0 ou 1) en sortie. La *fonction de décision* $f(x, w) : \mathbb{X} \times \mathbb{W} \to \mathbb{Y}$ donne le résultat en sortie du réseau de neurones. La sortie de la fonction de décision est notée $\hat{y}$. Dans le cas du perceptron, elle est de la forme suivante :

$$\hat{y} = f(x, w) = a(w_b + \sum_{i=1}^{d} w_i x_i) \tag{3.2}$$

avec a la fonction d'activation, w_b le poids du biais et d le nombre de neurones qui composent la couche d'entrée. Le biais est utilisé pour déplacer le résultat de la fonction d'activation vers le côté positif ou négatif. Dit autrement, le biais sert à translater la courbe produite par la fonction d'activation sur l'axe des abscisses. Cela permet de compenser le résultat.

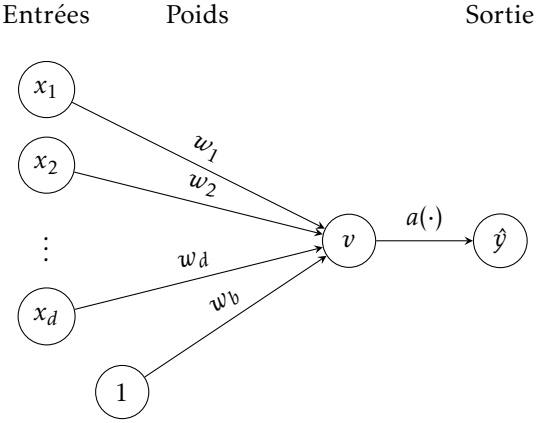


Figure 3.1 – Architecture d'un perceptron

Les poids de connexion initiaux ($w_i^{(0)}$) sont fixés arbitrairement. Puis, ils sont modifiés par le réseau de neurones jusqu'à ce que ces valeurs deviennent optimales. C'est l'entraînement du perceptron. Le but du jeu est de réduire l'erreur de la prédiction (la sortie) en modifiant les poids de connexion. Les outils pour y parvenir sont les *fonctions de perte* et les *algorithmes d'optimisation*.

La fonction de perte calcule la différence entre la sortie et le résultat attendu. Ce qui implique que l'apprentissage doit se faire sur des données étiquetées : pour chaque donnée en entrée, on connait le résultat attendu (y).

L'algorithme d'optimisation, quant à lui, permet de mettre à jour les poids. Si l'on calculait l'erreur pour chaque poids possible, dans le cas de notre perceptron, la courbe représentant l'erreur en fonction de la valeur des poids de connexion serait convexe, plus précisément, elle aurait la forme d'un bol. Le poids qui minimise l'erreur se situerait alors sur le minimum de cette courbe, au moment où la pente vaut 0. Cependant, il n'est pas réaliste de calculer la fonction de perte pour chaque valeur possible de w. C'est pour cela que l'on a besoin d'un algorithme d'optimisation. Ces algorithmes permettent de trouver les poids qui minimisent l'erreur, en utilisant une fonction de perte comme « guide » vers les bonnes valeurs. Il existe plusieurs algorithmes d'optimisation différents, cependant, ils ont en commun d'être basés sur les descentes de gradients.

Le but du réseau de neurones est de minimiser le résultat de la fonction de perte à l'aide de l'algorithme d'optimisation.

Maintenant que nous avons vu le cas simple, nous allons complexifier ce perceptron. Tout d'abord, en multipliant les sorties. Puis, en ajoutant des couches intermédiaires, dites « cachées ».

3.2.2 Perceptron multicouche et réseaux de neurones profonds

Un perceptron avec plusieurs couches de neurones est appelé un Perceptron MultiCouche (PMC). Quand il y a beaucoup de couches, on parle de *réseaux de neurones profonds* et par

conséquent, l'apprentissage associé est *l'apprentissage profond* (le nombre correspondant à « beaucoup » n'étant pas défini, on peut utiliser PMC et réseaux de neurones profonds de façon interchangeable). Peu importe la dénomination, l'idée est la même : chaque neurone intermédiaire applique sa fonction d'activation à une combinaison linéaire de ses entrées, de même que les neurones de sortie. Les sorties sont donc obtenues en appliquant la fonction d'activation des neurones de sortie à la combinaison linéaire des sorties des couches intermédiaires. Cela permet d'écrire la fonction de décision ($f(x, w)$) en fonction des variables d'entrées.

La fonction d'activation permet de rendre le fonctionnement de chaque neurone non-linéaire, c'est-à-dire que la valeur de sortie du neurone n'est pas proportionnelle à celles en entrée. En effet, sans cela, une suite de couches linéaires équivaudrait simplement en une couche linéaire unique. Utiliser une fonction d'activation permet au réseau de neurones d'apprendre des schémas plus complexes.

Comme pour le perceptron, on calcule l'erreur sur le neurone de sortie en comparant la sortie finale $\hat{y}$ avec le résultat attendu y. De même, on utilise un algorithme d'optimisation afin de trouver les valeurs optimales des poids. La différence tient dans la *rétro-propagation des erreurs*. En effet, il y a plusieurs couches cachées entre les entrées et les sorties, par conséquent, il faut décomposer l'erreur entre toutes ces couches. On utilise pour cela le *théorème de dérivation des fonctions composées*. D'après ce théorème, si une variable y dépend d'une variable z, qui dépend quant à elle d'une variable x, alors on a :

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial x} \tag{3.3}$$

On distingue la propagation avant (le calcul de la sortie de chaque nœud, de la couche d'entrée vers la couche de sortie) de la rétro-propagation ou propagation arrière (le calcul de la correction à apporter pour chaque poids, de la couche de sortie vers la couche d'entrée).

Les réseaux de neurones dépendent de toute une série d'hyperparamètres.

3.2.3 Hyperparamètres

Un hyperparamètres est un paramètre utilisé afin de contrôler le comportement de l'algorithme. Les hyperparamètres suivants reviennent dans tous les algorithmes utilisant de près ou de loin l'apprentissage profond :

— la fonction d'activation a ;
— la fonction de perte L ;
— l'algorithme d'optimisation $\mathcal{A}$;
— le pas (ou taux) d'apprentissage η.

Il est important de ne pas confondre l'algorithme d'optimisation (l'algorithme d'apprentissage) avec le modèle, qui, dans l'apprentissage profond, est l'architecture du réseau de neurones. Voici des fonctions et algorithmes utilisées souvent par défaut comme hyperparamètres :

La fonction d'activation ReLU

ReLU (pour Rectifier Linear Unit) est une fonction d'activation régulièrement utilisée. ReLU est définie comme $\text{ReLU}(x) = \max(0, x)$. Autrement dit, ReLU remplace toutes les valeurs négatives par des 0.

La fonction de perte de Huber

La fonction de perte de Huber est définie comme :

$$L_\delta(\hat{y}, y) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & \text{si } |y - \hat{y}| \leq \delta \\ \delta \cdot (|y - \hat{y}| - \frac{1}{2}\delta) & \text{sinon} \end{cases} \tag{3.4}$$

À noter que δ est un hyperparamètre supplémentaire.

Algorithme d'optimisation : la descente de gradient

La descente de gradient est l'algorithme d'optimisation le plus classique, dont dérivent la plupart des autres. Il a la particularité de ne pas avoir été développé spécifiquement pour les réseaux de neurones.

Le gradient ($\nabla\phi(w)$) représente la pente d'une fonction $\phi : \mathbb{W} \to \mathbb{R}$ en un point w, il s'agit d'un vecteur qui indique le point le plus bas de la fonction. Mathématiquement parlant, le gradient, dans un système de coordonnées cartésiennes, est la dérivée partielle d'une fonction par rapport aux coordonnées. Dans notre cas, pour des poids $w = (w_1, \ldots, w_m) \in \mathbb{W} = \mathbb{R}^m$ donnés :

$$\nabla_w L(f(x, w), y) = \left(\frac{\partial L(f(x, w), y)}{\partial w_1}, \cdots, \frac{\partial L(f(x, w), y)}{\partial w_m} \right). \tag{3.5}$$

Le but de l'algorithme est de chercher un gradient égal à (ou proche de) 0. L'algorithme du gradient est présenté dans l'Algorithme 1 pour une fonction générique $\phi(w)$ (voir la Figure 3.2a pour une illustration). Dans le cas de l'apprentissage profond, la descente de gradient est utilisée pour trouver les poids w qui minimisent la fonction de perte $L(f(x, w), y)$. Le gradient est donc calculé par rapport à w.

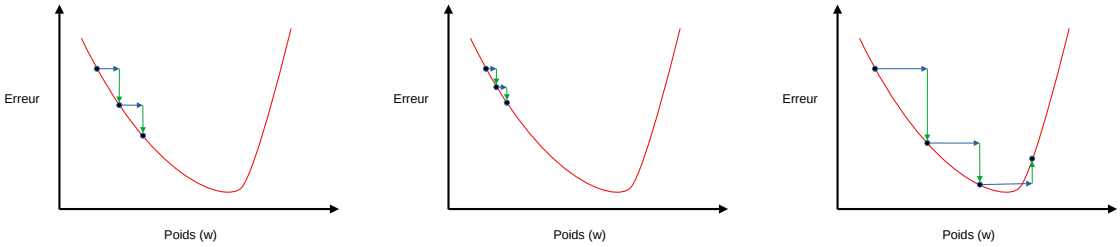
Algorithme 1 : Descente de gradient

Entrées : $\phi : \mathbb{W} \to \mathbb{R}$, $\epsilon > 0$, $w^{(0)} \in \mathbb{R}^m$ aléatoire, $\eta > 0$
Résultat : $w^{(t)}$ tel que $\|\nabla_w L(w^{(t)})\|$

$g^{(0)} \leftarrow \nabla_w L(w^{(0)})$;
$t \leftarrow 0$;
tant que $\|g^{(t)}\| > \epsilon$ **faire**
    $g^{(t)} \leftarrow \nabla_w L(w^{(t)})$;
    $w^{(t+1)} \leftarrow w^{(t)} - \eta \cdot g^{(t)}$;
    $t \leftarrow t + 1$;
fin

L'hyperparamètre η est appelé « vitesse d'apprentissage » (ou « taux d'apprentissage »). C'est un hyperparamètre important de l'algorithme de descente de gradient, car il définit de combien seront éloignés deux points entre chaque itération. Si le pas est trop faible, l'algorithme mettra beaucoup de temps à converger (Figure 3.2b), à l'inverse, s'il est trop grand, l'algorithme aura plus de chance de diverger (Figure 3.2c).

Dans l'apprentissage profond, nous avons souvent affaire à des modèles très complexes qui nécessitent une grande quantité de données d'entraînement. Par conséquent, il est difficile de calculer le gradient de la fonction de perte sur l'ensemble des données à chaque étape de la descente de gradient. C'est pourquoi il est préférable d'utiliser un petit sous-ensemble de



(a) Exemple de descente de gradient

(b) Pas d'apprentissage trop court

(c) Pas d'apprentissage trop long

FIGURE 3.2 – Exemples de descentes de gradient (la flèche bleue représente le pas d'apprentissage)

l'ensemble de données, appelé lot ou mini-lot. La procédure d'application de la descente de gradient à l'aide de lots est appelée descente de gradient stochastique.

Algorithme d'optimisation : Adam

L'algorithme Adam (Adaptive Moment Estimation), défini dans (KINGMA et BA 2014), est une technique d'optimisation pour la mise à jour des poids et des biais dans les réseaux de neurones. Il s'agit d'une extension de la descente de gradient qui utilise un taux d'apprentissage adaptatif pour chaque poids en utilisant des moyennes mobiles des gradients de premier et second ordre. L'algorithme Adam permet un apprentissage efficace et rapide des modèles de réseaux de neurones. L'algorithme est donné sur l'Algorithme 2.

Algorithme 2 : Algorithme Adam

Entrées : Données d'apprentissage $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$, taux d'apprentissage α, coefficients d'approximation exponentielle β_1 et β_2, tolérance ϵ

Output : Poids entraînés w

Initialiser les moments $\mu_0 = 0$, $v_0 = 0$, et l'itération $t = 0$;

tant que *critères de convergence non atteint* **faire**

- $t \leftarrow t + 1$;
- Calculer le gradient g_t en utilisant un mini-lot aléatoire de D ;
- $\mu_t \leftarrow \beta_1 \cdot \mu_{t-1} + (1 - \beta_1) \cdot g_t$;
- $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$;
- $\hat{\mu}_t \leftarrow \frac{\mu_t}{1-\beta_1^t}$;
- $\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$;
- Mettre à jour les poids : $w \leftarrow w - \alpha \cdot \frac{\hat{\mu}_t}{\sqrt{\hat{v}_t + \epsilon}}$;

fin

Afin d'y voir plus clair, nous allons voir le fonctionnement d'un réseau de neurones sur un exemple (en l'occurrence, un PMC).

3.2.4 Exemple de fonctionnement d'un réseau de neurones

Considérons le réseau de neurones sur la Figure 3.3. Ce réseau de neurones a comme entrées $x = (x_1, x_2) \in \mathbb{R}^2$ et comme sortie $\hat{y} = f(x, w) \in \mathbb{R}$. La sortie est comparée avec la sortie attendue

y afin de calculer la perte $L(\hat{y}, y)$. Les poids $w = (w_{111}, \ldots, w_{121})$ sont les paramètres à trouver durant l'entraînement. Cet entraînement se fait en deux phases.

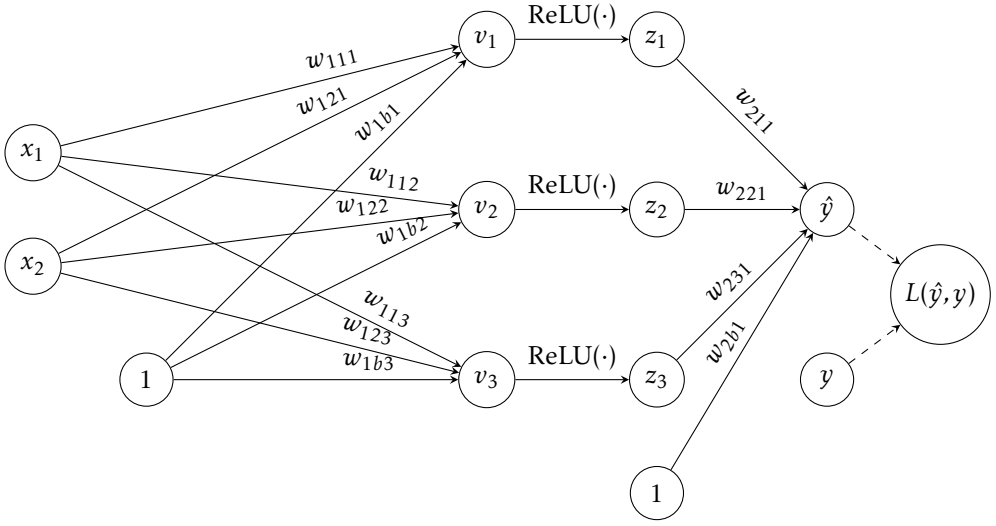


FIGURE 3.3 – Exemple de réseau de neurones

Durant la phase de propagation avant, les valeurs de x, w, et y sont fixes. Elles sont utilisées pour calculer les valeurs intermédiaires $v = (v_1, v_2, v_3)$:

$$v_1 = w_{1b1} + w_{111}x_1 + w_{121}x_2 \tag{3.6}$$
$$v_2 = w_{1b2} + w_{112}x_1 + w_{122}x_2 \tag{3.7}$$
$$v_3 = w_{1b3} + w_{113}x_1 + w_{123}x_2 \tag{3.8}$$

et $z = (z_1, z_2, z_3)$:

$$z_1 = \text{ReLU}(v_1) = \max(0, v_1) \tag{3.9}$$
$$z_2 = \text{ReLU}(v_2) = \max(0, v_2) \tag{3.10}$$
$$z_3 = \text{ReLU}(v_3) = \max(0, v_3) \tag{3.11}$$

La prédiction finale du réseau de neurone $\hat{y}$ est :

$$\hat{y} = w_{2b1} + w_{211}z_1 + w_{221}z_2 + w_{231}z_3 \tag{3.12}$$

qui est comparée avec y en calculant la perte $L(\hat{y}, y)$.

Durant la phase de propagation arrière (ou rétro-propagation), les valeurs calculées lors de la propagation avant sont utilisées pour calculer le gradient de la perte L par rapport aux poids w :

$$\nabla_w L = \left(\frac{\partial L}{\partial w_{111}}, \ldots, \frac{\partial L}{\partial w_{1b1}}, \frac{\partial L}{\partial w_{211}}, \ldots, \frac{\partial L}{\partial w_{2b1}} \right) \tag{3.13}$$

Voyons comment cela fonctionne. Dans les réseaux de neurones, les dérivées partielles suivantes sont connues et codées en dur dans la couche correspondante :

— les opérations linéaires (par exemple, la dérivée de $\alpha w_j + c$ par rapport à w_j est α) ;

— les fonctions d'activation (par exemple, la dérivée de $\text{ReLU}(z_j) = \max(0, z_j)$ par rapport à z_j est 0 si $z_j \leq 0$ et 1 sinon) ;
— la dérivée de la fonction de perte L par rapport à $\hat{y}$ (par exemple, pour la perte de Huber, elle est de $2(\hat{y} - y)$ si $|\hat{y} - y| < \delta$ et $\delta \cdot \text{sign}(\hat{y} - y)$ dans le cas contraire)

Une fois celles-ci connues, les dérivées partielles de L par rapport à chaque poids w_j peuvent facilement être trouvées en utilisant la règle de la chaîne, en construisant un chemin entre L et w_j et en calculant toutes les dérivées intermédiaires.

Par exemple, pour w_{211}, la dérivée partielle est :

$$\frac{\partial L}{\partial w_{211}} = \underbrace{\frac{\partial L}{\partial \hat{y}}}_{\text{connu}} \cdot \underbrace{\frac{\partial \hat{y}}{\partial w_{211}}}_{z_1} \tag{3.14}$$

On remarque que la valeur z_1, qui a été calculée au cours de la propagation avant, est utilisée.

Pour les poids de la première couche, la trajectoire est plus longue, et la dérivée partielle est donc plus complexe. Par exemple, pour w_{111}, la dérivée partielle est :

$$\frac{\partial L}{\partial w_{111}} = \underbrace{\frac{\partial L}{\partial \hat{y}}}_{\text{connu}} \cdot \underbrace{\frac{\partial \hat{y}}{\partial z_1}}_{w_{211}} \cdot \underbrace{\frac{\partial z_1}{\partial v_1}}_{0 \text{ ou } 1} \cdot \underbrace{\frac{\partial v_1}{\partial w_{111}}}_{x_1} \tag{3.15}$$

Nous avons vu le fonctionnement de réseaux de neurones parmi les plus simples. Il en existe beaucoup d'autres, plus compliqués, comme les réseaux de neurones convolutifs. Ils sont cependant tous basés sur les concepts vu ici. De plus, dans notre cas, nous allons nous contenter de réseaux de neurones (dits séquentiels) comme décrits dans cet exemple. Les réseaux de neurones sont principalement utilisés pour calculer des probabilités. Toutefois, dans le cadre du DRL, on peut les utiliser afin d'approximer des fonctions. Ils sont présents dans la plupart des algorithmes récents, que nous allons voir dans le chapitre suivant.

Bibliographie du présent chapitre

Ahmad, Ijaz et al. (2020). « Machine Learning Meets Communication Networks : Current Trends and Future Challenges ». In : *IEEE Access* 8, p. 223418-223460. issn : 2169-3536. doi : 10.1109/ACCESS.2020.3041765.

Boutaba, Raouf et al. (2018). « A comprehensive survey on machine learning for networking : evolution, applications and research opportunities ». In : *Journal of Internet Services and Applications* 9.1, p. 1-99.

Goodfellow, Ian, Yoshua Bengio et Aaron Courville (2016). *Deep Learning*. https://www.deeplearningbook.org. MIT Press.

Kingma, Diederik P et Jimmy Ba (2014). « Adam : A method for stochastic optimization ». In : *arXiv preprint arXiv :1412.6980*.

Pouyanfar, Samira et al. (sept. 2018). « A Survey on Deep Learning : Algorithms, Techniques, and Applications ». In : *ACM Comput. Surv.* 51.5. issn : 0360-0300. doi : 10.1145/3234150. url : https://doi.org/10.1145/3234150.

Tesauro, Gerald (jan. 2007). « Reinforcement Learning in Autonomic Computing : A Manifesto and Case Studies ». In : *IEEE Internet Computing* 11.1, p. 22-30. issn : 1941-0131. doi : 10.1109/MIC.2007.21.

Wang, Mowei et al. (mars 2018). « Machine Learning for Networking : Workflow, Advances and Opportunities ». In : *IEEE Network* 32.2, p. 92-99. issn : 1558-156X. doi : 10.1109/MNET.2017.1700200.

Chapitre 4

Apprentissage par renforcement

Sommaire du présent chapitre

Le Reinforcement Learning (RL) (Sutton et Barto 2018) est un paradigme de ML qui, plutôt que de s'appuyer sur des exemples étiquetés ou des motifs contenus dans des données, apprend grâce à des expériences sur un environnement un comportement lui permettant de résoudre un problème donné. Ce paradigme d'apprentissage est le plus approprié pour des problèmes dans lesquels la meilleure solution est inconnue, mais peut être trouvée en se basant sur les retours obtenus de l'environnement. Il s'agit d'un domaine très vaste, dont les bases théoriques ainsi que les notions fondamentales sont données dans ce chapitre.

Le RL est apparu en 1957 avec l'introduction par Richard Bellman des *processus de décisions Markoviens* (Bellman 1957) (en anglais Markov Decision Process (MDP)). Les MDP sont inspirés par les chaînes de Markov.

4.1 Introduction aux processus de décisions Markoviens

Une chaîne de Markov est un processus stochastique sans mémoire. Il possède un nombre fixe d'états et il passe d'un état à un autre en fonction de probabilités connues à l'avance. Ces probabilités dépendent uniquement de l'état présent et de l'état futur, pas des états passés (système sans mémoire). C'est ce que l'on appelle la *propriété de Markov*. Un exemple de chaîne de Markov est donné sur la Figure 4.1. Dans cet exemple, si l'état actuel est s_0, alors l'état suivant sera encore s_0 avec 60% de chances ou s_1 avec 40% de chances, et ce, de manière totalement décorrélée de la suite des états précédents. De plus, sur cette figure, s_2 est un état terminal, c'est-à-dire que l'on ne peut pas le quitter.

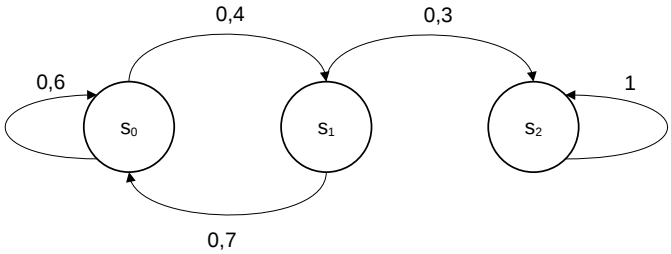


Figure 4.1 – Exemple de chaîne de Markov

Lorsqu'il a introduit les MDP, Bellman a légèrement modifié les chaînes de Markov en y ajoutant des actions, dont dépendent les probabilités, ainsi que des récompenses, pouvant être nulles. Dans un environnement déterministe, un MDP est défini comme étant un quadruplet s, a, T, R avec :

— $s \in \mathbb{S}$, et $\mathbb{S}$ l'espace des états,
— $a \in \mathbb{A}$, et $\mathbb{A}$ l'espace des actions,
— $T : \mathbb{S} \times \mathbb{A} \rightarrow \mathbb{S}$ la fonction de transition ($T(s^{'}, r|s, a)$),
— $R : \mathbb{S} \times \mathbb{A} \times \mathbb{S} \rightarrow \mathbb{R}$ la fonction de récompense ($R(s, a, s^{'})$).

À noter que la fonction de transition nécessite une connaissance initiale totale de l'environnement. Un exemple de MDP est donné sur la Figure 4.2. Dans cet exemple, si l'état actuel est s_0, alors deux actions différentes sont possibles : la première (a_1) a 40% de chances de mener vers l'état s_1, tandis qu'elle a 60 % de chances de mener vers l'état actuel (s_0) avec une récompense de +10. Inversement, si l'état actuel est s_1 et qu'on exécute l'action a_1, alors on a 100 % de chances d'aller vers s_2 avec une récompense de −40. On peut remarquer que s'il n'y a qu'une action possible et que la récompense est toujours la même, le MDP devient une chaîne de Markov. Un MDP est donc une généralisation des chaînes de Markov.

Les MDP sont les concepts mathématiques sur lesquels sont basés le RL. Cependant, les algorithmes modernes, faisant intervenir des RNA et capables d'agir sur des environnements très complexes, ont tendance à diverger de ces principes. En particulier, l'agent n'a souvent pas de connaissance initiale de l'environnement. De même, la propriété de Markov n'est pas toujours respectée. En tout état de causes, les MDP ne sont pas les seuls composants du RL.

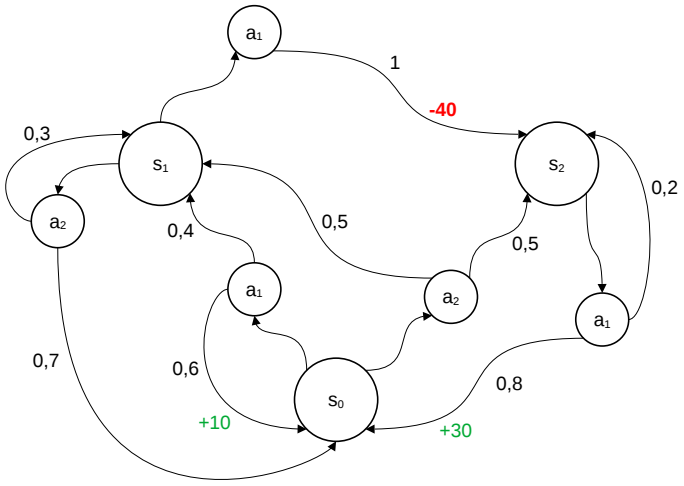


FIGURE 4.2 – Exemple de MDP

4.2 Éléments de l'apprentissage par renforcement

Dans le RL, un *agent* interagit avec un *environnement* comme sur la Figure 4.3. Dans le cas des processus de décisions markoviens, cela se traduit de la manière suivante : à chaque instant t, l'agent observe un *état* s_t de l'environnement et obtient une récompense r_t (liée à la conséquence de l'action prise à $t-1$). Il exécute une nouvelle action a_t en réaction. L'état passe alors à s_{t+1} et l'agent reçoit une récompense r_{t+1}, et ainsi de suite jusqu'à ce que l'agent gagne ou perde. On peut voir cela comme une séquence qui commencerait de la façon suivante : s_0, a_0, r_1, s_1, a_1, r_2... Le but pour l'agent étant de maximiser sa récompense sur le long terme, c.-à-d., de trouver la séquence d'action lui permettant d'obtenir une somme de récompenses maximum.

On peut séparer en trois groupes les différents éléments composants le RL : ce que l'on définit lors de la modélisation (en lien avec l'environnement), ce que l'agent apprend et les différents algorithmes permettant l'apprentissage.

4.2.1 Éléments définis lors de la modélisation

Les différents éléments décrits ici sont ceux que l'utilisateur doit définir lors de la modélisation de son environnement.

États et observations

Un *état* de l'environnement, noté s, correspond à une image de cet environnement à un moment donné et est la notion utilisée dans la théorie des MDP. Une *observation*, notée o, correspond à un sous-ensemble de l'état. Il existe une différence sémantique entre ces 2 notions. L'état est la description complète de l'environnement, tandis qu'une observation n'est que partielle. En effet, en pratique, il n'est pas rare que l'agent n'ait pas accès à la totalité de l'environnement. Dans des environnements complexes, un agent doit généralement baser ses décisions sur des observations, car il ne connait pas l'état. Par exemple, dans le cas d'un réseau de télécommunication, l'état

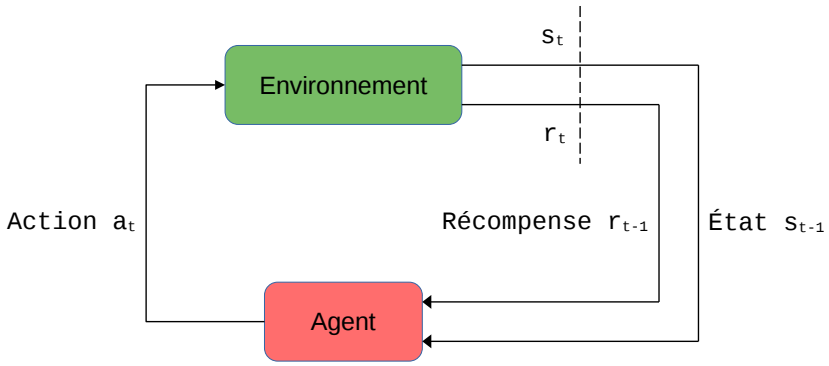


FIGURE 4.3 – Les interactions entre l'agent et l'environnement

engloberait des informations sur les topologies, les flux, les différentes configurations, etc. En plus d'être un ensemble d'informations difficile à rassembler, les avoir toutes résulterait en un espace des états immense et donc, dans des calculs très compliqués. Dans ce cas, l'agent peut se limiter à des observations relatives au problème qu'il cherche à résoudre, par exemple, s'il cherche à minimiser des latences, observer les latences de bout-en-bout de chaque flux peut suffire. Ces deux notions, état et observation, sont fréquemment mélangées, la première étant souvent utilisée dans la théorie et la seconde, dans la pratique.

L'*espace des états* ($\mathbb{S}$) est l'ensemble des états possibles tandis que l'*espace des observations*, noté $\mathbb{O}$, est l'ensemble des observations possibles. Les espaces des états et des observations peuvent être soit discret (à valeurs dans l'espace des entiers relatifs), soit continu (à valeurs dans l'espace des réels). Cela doit être pris en compte lors du choix de l'implémentation de l'algorithme d'apprentissage. En effet, toutes les implémentations ne supportent pas tous les types d'espace. De plus, certains algorithmes d'apprentissage, dans leur version originale, sont optimisés seulement pour un seul type d'espace.

Actions

Une *action*, généralement notée a, est le nom donné à l'interaction que fait l'agent avec l'environnement. Une action modifie l'environnement.

L'*espace des actions* ($\mathbb{A}$) est l'ensemble des actions possibles. Comme pour l'espace des états, l'espace des actions peut être soit discret, soit continu. Un espace des actions est discret quand le nombre d'actions est fini (par exemple, aller tout droit/à droite/en arrière/à gauche). Sinon, l'espace des actions est continu. Comme précédemment, cette distinction est importante pour certains algorithmes d'apprentissage, qui peuvent fonctionner dans un cas et pas dans l'autre.

Récompenses

L'agent reçoit de l'environnement un *signal de récompense* après avoir pris une action. Ça lui indique comment il doit interpréter l'état. C'est sur cette base que l'agent va modifier son comportement. Une récompense est généralement notée r. La récompense est fonction de l'état actuel s et de l'action prise a ainsi que du nouvel état en résultant s' ($r = R(s,a,s')$). Parfois,

la récompense ne dépend que de l'état actuel ($r = R(s)$) ou de l'état actuel et de l'action prise ($r = R(s,a)$).

Le but de l'agent est de maximiser les récompenses cumulées au long d'un cycle d'apprentissage (on note ce cumul $R_{\text{total}}(\tau)$). Ce but est en réalité très dépendant du contexte. Il peut être la somme de toutes les récompenses de ce cycle ($R_{\text{total}}(\tau) = \sum_{t=1}^{\tau} r_t$, cas le plus simple), mais on peut aussi définir un facteur d'actualisation (ou taux de rabais) γ, avec $0 \leq \gamma \leq 1$, qui sert à pondérer chaque étape : plus γ est petit, plus une récompense immédiate est privilégiée ($R_{\text{total}}(\tau) = \sum_{t=1}^{\tau} \gamma^t r_t$).

La récompense reçue par l'agent est défini par une *fonction de valeur*, qui définit la récompense sur le long terme. Elle permet de prendre en compte les états qui vont suivre et leurs récompenses. De cette façon, l'agent peut être amené à prendre une décision qui lui apportera une récompense moyenne ou basse, mais qui sera suivie par un ou des états qui impliquent de fortes récompenses. C'est une manière de faire un compromis entre les récompenses sur le court et le long terme. Estimer la valeur de la récompense sur le long terme est la partie la plus difficile de l'algorithme d'apprentissage. Les fonctions de valeur sont notées $V(s)$ ou $Q(s,a)$, tandis que $V^*(s)$ et $Q^*(s,a)$ sont des fonctions de valeur optimales. À noter que γ est aussi utilisé dans les fonctions de valeur. Lorsque c'est le cas, il s'agit d'un hyperparamètre de l'algorithme.

Épisodes

Une interaction entre l'agent et l'environnement (a_t, s_t, r_t) est appelé un *pas* (*step* en anglais). Une séquence d'interactions entre l'agent et l'environnement menant à un résultat (positif ou négatif) est appelée *épisode* (ou trajectoire ou encore déploiement), noté τ. Un épisode commence par un état généralement déterminé aléatoirement et se termine par un état particulier appelé *état terminal*. Les transitions entre les différents états dépendent des lois de l'environnement. Elles ne dépendent que de la dernière action. Ces transitions peuvent être déterministes ($s' = f(s,a)$) ou stochastique ($s' \sim P(\cdot|s,a)$).

4.2.2 Éléments appris par l'agent

Les éléments suivants sont ceux que l'agent apprend lors de l'entraînement.

Politiques

Le lien entre l'état observé et l'action à prendre s'appelle une *politique*. Une politique définit quelle action prendre en fonction de la dernière observation. On peut voir la politique comme le « cerveau » de l'agent. C'est pourquoi ces deux termes, politique et agent, peuvent être parfois un peu mélangés. Une politique peut être soit déterministe, soit stochastique. Quand la politique est déterministe, elle est normalement notée μ et l'action associée s'écrit : $a = \mu(s)$. Quand elle est stochastique, elle est notée π et l'action est : $a \sim \pi(\cdot|s)$. Cependant, dans la littérature, la politique est très souvent notée π peu importe les circonstances, étant donné que l'on peut considérer qu'une politique déterministe est un cas spécial de politique stochastique.

Une politique est une fonction mathématique définie sur l'ensemble des états $\mathbb{S}$ à valeur dans l'ensemble des actions $\mathbb{A}$ ($\pi : \mathbb{S} \to \mathbb{A}$). Elle peut aussi être une probabilité de prendre une action a pour un état s ($\pi : \mathbb{A} \times \mathbb{S} \to [0,1]$). Enfin, dans des cas compliqués, la politique peut être approximée par un réseau de neurones. C'est l'idée utilisée dans une famille d'algorithmes appelée *apprentissage par renforcement profond* (Li 2018 ; Arulkumaran et al. 2017) (DRL en anglais). Le fait que l'on puisse approximer une fonction par un réseau de neurones formé d'une seule couche intermédiaire a été démontré dans (Hornik 1991). Dans ce cas, la politique est

paramétrique, elle dépend des poids et biais du réseau de neurones. Ce paramètre est noté θ (ou parfois ϕ) et l'on a $a = \mu_\theta(s)$ ou $a \sim \pi_\theta(\cdot|s)$.

Le but de l'apprentissage est d'apprendre une *politique optimale* qui maximise la récompense (π^* ou μ^*).

À noter qu'il existe deux types de politiques différentes : une *politique cible* et une *politique comportementale*. La politique cible est la politique que l'agent apprend, autrement dit, c'est la politique qui suivra la fonction de valeur. La politique comportementale est la politique utilisée par l'agent pour sélectionner une action lors de l'entrainement, c'est la politique qui permet l'interaction avec l'environnement. Dans un algorithme de *politique en ligne* (*on-policy*), l'algorithme va évaluer et améliorer la politique utilisée pour sélectionner les actions (politique cible = politique comportementale). Un algorithme de *politique hors ligne* (*off-policy*), quant à lui, va évaluer et améliorer une politique différente de celle utilisée pour sélectionner les actions (politique cible ≠ politique comportementale). D'un point de vue pratique, la différence tient dans le fait que, lors de l'entrainement, un algorithme de politique en ligne prendra des décisions basées sur la seule expérience précédente tandis qu'un algorithme de politique hors ligne considérera toutes les expériences précédentes.

L'exploration est quand l'agent tente d'améliorer ses connaissances sur chaque action. Pour ce faire, il va commencer par sélectionner ses actions aléatoirement, puis apprendre et s'améliorer de manière itérative. *L'exploitation*, à l'inverse, est quand l'agent cherche à obtenir la meilleure récompense possible immédiatement.

On définit aussi un facteur d'apprentissage α, avec $0 \leq \alpha \leq 1$, qui est utilisé afin de déterminer l'importance de la nouvelle information apprise : si $\alpha = 0$, l'agent n'acquiert pas de nouvelles connaissances et conserve l'intégralité des anciennes connaissances, à l'inverse, si $\alpha = 1$, l'agent oublie systématiquement les connaissances antérieures et n'utilisera que l'information apprise la plus récente.

4.2.3 Classification des algorithmes d'apprentissage

Il existe énormément d'algorithmes d'apprentissage par renforcement. Chaque algorithme est adapté à un problème particulier (souvent, résoudre un jeu). Afin d'aider à trouver celui qui nous correspond le mieux, on peut les classer selon différents critères. Cela permet de réduire leur nombre.

La manière la plus simple de classer les algorithmes d'apprentissage par renforcement est de les diviser entre les méthodes *tabulaires* et *approximatives*. Ce qui fait la différence est la taille de l'espace des états. Dans le cas où elle est suffisamment petite, la fonction de valeur peut être représentée par une table, ce qui permet souvent de trouver une politique optimale. Utiliser une table est impossible dans le cas où il y a beaucoup d'état (il peut potentiellement y en avoir plus que d'atomes dans l'univers (Mnih, Kavukcuoglu, Silver, Graves et al. 2013)). Le problème est qu'il est très difficile d'apprendre à partir d'états déjà rencontrés, car la probabilité de les rencontrer de nouveau est très faible. Il faut donc généraliser à partir de ceux-ci. Pour ce faire, on utilise des méthodes utilisées pour l'apprentissage supervisé comme l'approximation de fonction ou l'apprentissage profond. Les algorithmes récents sont tous approximatifs. La raison en est l'utilisation quasi-systématique de réseaux de neurones (technologie à la mode) dans les algorithmes, lesquels sont voués à résoudre des problèmes de plus en plus complexes, que les algorithmes tabulaires ne sont plus capables de résoudre en raison d'espaces des états trop grands (ce qui sera notre cas). Pour cette raison, nous utiliserons un algorithme approximatif.

Pour classer tous ces algorithmes, on peut aussi chercher à savoir s'ils sont liés (d'une manière ou d'une autre) à un *modèle* de l'environnement (voir Figure 4.4). Ces modèles mathématiques vont imiter le comportement de l'environnement grâce à la probabilité de passer à l'état s_{t+1} et

de recevoir une récompense r_{t+1}, depuis l'état s_t et en effectuant une action a_t (on parle aussi de fonction de transition). A priori et dans la plupart des cas, l'agent ne peut contrôler que les actions, il ne sait pas comment va évoluer l'état (il n'a pas accès au modèle de l'environnement). Ces algorithmes sont dits *sans modèles* (*model-free*). Néanmoins, certains algorithmes, *basés sur des modèles* (*model-based*), vont chercher à prédire le résultat d'une action sur un état (la récompense et l'état suivant). Ces algorithmes sont plus performants, mais aussi plus compliqués et plus rares.

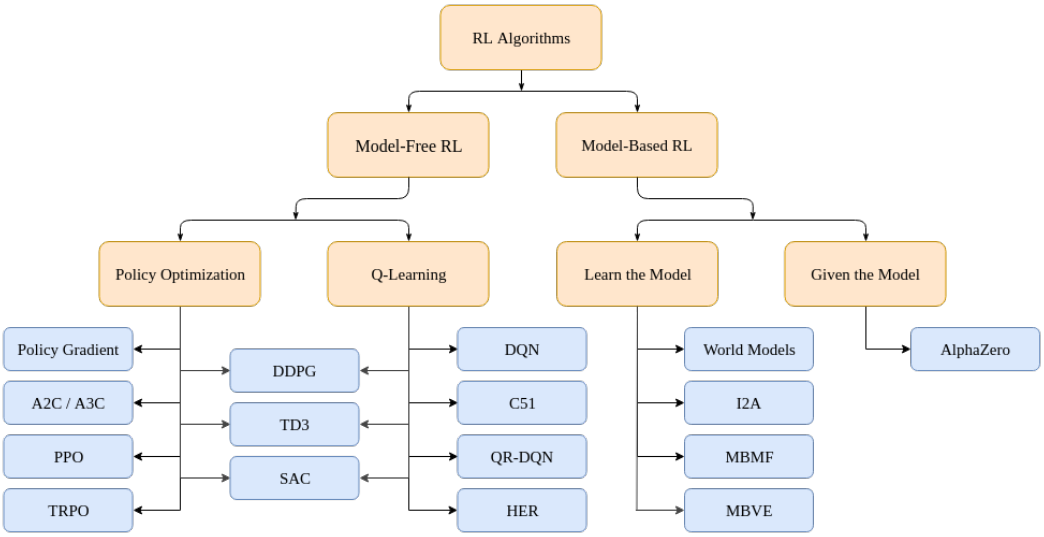


FIGURE 4.4 – Classification des algorithmes d'apprentissage par renforcement profonds faite par OpenAI [1]

Pour finir, on peut aussi classer les algorithmes selon leur manière d'évaluer la politique (en ligne ou hors ligne).

En résumé, les algorithmes d'apprentissage par renforcement peuvent être divisés en un certain nombre de catégories en fonction du contexte. Il y a 3 paramètres à prendre en compte pour choisir un algorithme d'apprentissage :

1. tabulaire/approximatif,
2. sans modèle/basé sur un modèle,
3. politique en ligne/hors ligne.

Afin d'affiner le choix, on peut aussi choisir un algorithme en se basant sur l'implémentation de la formalisation du problème et plus particulièrement, sur la modélisation des espaces des états et des actions.

Dans la suite, nous allons nous concentrer sur des algorithmes approximatifs, car notre espace des états est très grand et le problème à résoudre est compliqué. De plus, nous utiliserons des algorithmes sans modèle, parce que l'on ne connait pas les fonctions de transitions a priori. Cela nous laisse deux familles d'algorithmes de DRL différentes : celle basée sur Q-learning et celle basée sur l'optimisation de la politique.

1. Image issue de spinningup.openai.com/en/latest/spinningup/rl_intro2.html

4.3 Algorithmes basés sur Q-learning

Une des principales difficultés dans le RL consiste à trouver une stratégie consistant à maximiser la récompense au cours du temps. C'est l'objectif des différents algorithmes présentés dans la suite de ce chapitre.

La première famille d'algorithmes que nous allons voir est dite « basée sur Q-learning ». La raison est que tous les nouveaux algorithmes de cette famille « descendent » de l'algorithme Q-learning, ils ont tous en commun de chercher une approximation de $Q^*(s,a)$. Ce sont aussi les premiers à avoir utilisé des réseaux de neurones. Ce sont généralement des algorithmes de politique hors ligne.

4.3.1 Origines de Deep Q-learning

Le but pour l'agent consiste à trouver une politique lui permettant de maximiser sa récompense sur le long terme. Mathématiquement, cela revient à trouver un moyen de calculer (et représenter) les récompenses en fonction des états. Bellman a trouvé une manière d'estimer cela. Il a défini une équation récursive, *l'équation d'optimalité de Bellman*, qui permet de calculer la *valeur d'état optimale* de tout état s ($V^*(s)$). La valeur d'état optimale d'un état s est la somme de toutes les récompenses que peut espérer l'agent quand il a atteint cet état *lorsqu'il agit de façon optimale*. L'équation d'optimalité de Bellman est la suivante :

$$V^*(s) = \max_{a \in \mathbb{A}} \sum_{s' \in \mathbb{S}} T(s', r|s, a)[R(s, a, s') + \gamma \cdot V^*(s')] \tag{4.1}$$

avec $T(s^{'}, r|s, a)$ la fonction de transition.

Grâce à cette équation, on peut imaginer un algorithme qui donne une estimation de la valeur d'état optimale de chaque état possible ($V(s)$). C'est l'algorithme *d'itération sur la valeur* (Algorithme 3).

Algorithme 3 : Algorithme d'itération sur la valeur

Entrées : $\theta > 0$
Résultat : $\pi \approx \pi^*$

Initialiser arbitrairement le tableau $V[nb\_etats]$, à l'exception de $V[terminal]$;
$V[terminal] \leftarrow 0$;
tant que $\Delta \geq \theta$ **faire**
    $\Delta \leftarrow 0$;
    pour chaque $s \in \mathbb{S}$ **faire**
        $v \leftarrow V(s)$;
        $V(s) \leftarrow \max_{a \in \mathbb{A}} \sum_{s' \in \mathbb{S}} T(s', r|s, a)[R(s, a, s') + \gamma \cdot V(s')]$;
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$;
    fin
fin
$\pi(s) = \arg\max_{a \in \mathbb{A}} \sum_{s' \in \mathbb{S}} T(s', r|s, a)[R(s, a, s') + \gamma \cdot V(s')]$;

L'Algorithme 3 converge toujours vers les valeurs d'état optimales. Ces valeurs correspondent à la politique optimale que doit mener l'agent. Cependant, elles ne disent pas comment mener cette politique. C'est ici que Bellman introduit les *valeurs Q* (Q comme Quality, qualité en

anglais). La valeur Q optimale d'un couple état-action $Q^*(s,a)$ (aussi appelées valeurs état-action optimale), est la somme de toutes les récompenses que peut espérer l'agent quand il a atteint l'état s et choisi l'action associée, *mais avant son exécution* (il n'est pas dans l'état suivant). Ici aussi, on suppose que l'agent agit de façon optimale, *mais après cette action*. Dans ce cas, l'équation d'optimalité de Bellman devient :

$$Q^*(s,a) = \sum_{s' \in \mathbb{S}} T(s', r|s, a)[R(s, a, s') + \gamma \cdot \max_{a' \in \mathbb{A}} Q^*(s', a')] \tag{4.2}$$

L'algorithme à suivre étant alors fondamentalement le même que l'Algorithme 3. On obtient la politique optimale $\pi^*(s)$ à suivre lorsqu'on est dans l'état s en sélectionnant l'action a qui maximise $Q(s,a)$. Autrement dit :

$$\pi^*(s) = \arg\max_{a \in \mathbb{A}} Q^*(s,a) \tag{4.3}$$

Maintenant, nous avons une politique qui nous indique clairement la marche suivre.

Il arrive que l'agent ne connaisse pas les probabilités de transition et que les récompenses soient initialement inconnues (c'est notre cas, on est donc sur une famille d'algorithmes sans modèle). C'est ici qu'intervient *l'apprentissage par différence temporelle*. L'idée de cette famille d'algorithmes est que, puisqu'on ne connait que les états et les actions possibles, on va chercher à estimer les valeurs d'états ou les valeurs Q en fonction d'observations et d'expériences. On utilise pour cela une *politique d'exploration*, c'est-à-dire une politique permettant d'améliorer les connaissances de l'agent sur les conséquences de ses actions.

Une des méthodes d'apprentissage par différence temporelle les plus connues est *Q-learning*, définie dans (C. J. C. H. Watkins 1989) et dont la convergence est prouvée dans (C. J. Watkins et Dayan 1992). Une grande partie des algorithmes plus récents découlent de cette méthode.

Q-learning est un algorithme sans modèle. Au lieu de ça, il apprend une fonction de valeur état-action $Q : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$. On rappelle que la fonction de valeur permet de déterminer la récompense sur le long terme. La fonction Q est définie de la façon suivante (lors de l'épisode n ; l'agent a pris une action a_t en fonction d'un état s_t et reçoit en retour une récompense r_t et un nouvel état s_{t+1}) :

$$Q_n(s_t, a_t) = Q_{n-1}(s_t, a_t) + \alpha \cdot [r_{t+1} + \gamma \cdot \max_{a \in \mathbb{A}} Q_n(s_{t+1}, a) - Q_{n-1}(s_t, a_t)] \tag{4.4}$$

où $\max_{a \in \mathbb{A}} Q_n(s_{t+1}, a)$ est la meilleure valeur de Q que l'agent peut espérer obtenir pour l'état s_{t+1}.

Plus précisément, l'idée est la suivante : les valeurs de Q pour chaque couple état-action sont enregistrées dans un tableau. Ces valeurs sont mises à jour à chaque fois que l'on tombe dessus lors de l'entrainement. L'algorithme de Q-learning est présenté dans l'Algorithme 4.

À la fin de l'apprentissage, on obtient une fonction de valeur optimale dont on peut dériver une politique optimale. Q-learning est donc un algorithme de politique hors ligne, Q étant la politique comportementale et π la politique cible. Q-learning est aussi un algorithme tabulaire (on doit enregistrer toutes les valeurs de Q pour chaque paire état-action). En résumé Q-learning est un algorithme tabulaire, sans modèle et de politique hors ligne.

Du fait que ce soit un algorithme tabulaire, Q-learning soufre d'un handicap lorsque l'espace des états est (très) grand. C'est ce problème que résout Deep Q-learning (DQL), défini dans (Mnih, Kavukcuoglu, Silver, Graves et al. 2013 ; Mnih, Kavukcuoglu, Silver, Rusu et al. 2015). DQL utilise un réseau de neurones profond afin d'approximer la fonction Q.

Algorithme 4 : Q-learning

Entrées : $\alpha, \gamma \in]0,1]$
Résultat : Q

Initialiser arbitrairement le tableau $Q[nb\_etats, nb\_actions]$, à l'exception de $Q[terminal, *]$;
$Q[terminal, *] \leftarrow 0$;
pour chaque *episode* **faire**
    initialiser état s;
    tant que *s n'est pas l'état terminal* **faire**
        Choisir a depuis s en utilisant la politique comportementale dérivée de Q (par exemple, ϵ-glouton, défini dans l'Algorithme 5);
        Exécuter a;
        Observer r et s';
        $Q[s,a] \leftarrow Q[s,a] + \alpha \cdot (r + \gamma \cdot \max_{a' \in \mathbb{A}} Q[s',a'] - Q[s,a])$;
        $s \leftarrow s'$;
    fin
fin

4.3.2 Deep Q-learning

L'architecture de réseau de neurones convolutif entrainé grâce à DQL est appelé *Deep Q-network* (DQN), car elle est capable de combiner l'apprentissage par renforcement avec de l'apprentissage profond. DQN utilise un poids de connexion θ tel que $Q(s,a,\theta) \approx Q^*(s,a)$. À noter que l'algorithme d'apprentissage DQL est souvent appelé DQN. Il s'agit d'un raccourci. Par souci de cohérence, dans la suite, nous utiliserons DQL pour faire référence à l'algorithme d'apprentissage.

DQL utilise une politique comportementale particulière appelée ϵ-glouton. Cette méthode permet de faire un compromis entre l'exploration et l'exploitation. ϵ représente la probabilité de choisir l'exploration. La méthode ϵ-gloutonne est présentée dans l'Algorithme 5.

Algorithme 5 : Politique comportementale ϵ-gloutonne

Entrées : $\epsilon \in]0,1]$, Q, S
Résultat : a

$n \leftarrow$ nombre aléatoire $\in]0,1]$;
si $n < \epsilon$ **alors**
    $a \leftarrow$ action aléatoire;
sinon
    $a \leftarrow$ meilleur action connue pour Q dans l'état s;
fin

Enfin, DQL utilise une technique appelée *relecture d'expérience* où les expériences de l'agent à chaque itération ($< s, a, r, s' >$) sont enregistrées dans un jeu de donnée, noté D, appelée *mémoire de relecture*. On utilise ensuite une de ces expériences, tirée aléatoirement, afin d'entraîner Deep Q-network, plutôt que l'observation courante. Ceci permet de potentiellement réutiliser les mêmes exemples, permettant une plus grande efficacité des données. Cela permet aussi de casser

la corrélation entre les données, réduisant la variance entre les mises à jour. L'algorithme DQL est décrit dans l'Algorithme 6.

Algorithme 6 : Deep Q-learning avec relecture d'expériences

Entrées : $\gamma, \epsilon \in]0,1]$
Données : y : récompense cible
Résultat : Q

Initialiser la mémoire de relecture D;
Initialiser la fonction de valeur état-action Q avec des poids aléatoires;
pour chaque *episode* **faire**
  Initialiser état s;
  tant que *s n'est pas l'état terminal* **faire**
    Choisir a en utilisant la méthode 5;
    Exécuter a;
    Observer r et s';
    Enregistrer la transition $< s, a, r, s' >$ dans D;
    Récupérer une transition aléatoire $< ss, aa, rr, ss' >$ depuis D;
    $y \leftarrow \begin{cases} rr & \text{si } ss' \text{ est terminal} \\ rr + \gamma \cdot \max\limits_{a' \in \mathbb{A}} Q(ss', a', \theta) & \text{sinon} \end{cases}$
    Effectuer une étape de l'algorithme du gradient sur $(y - Q(ss, aa, \theta))^2$;
    $s \leftarrow s'$;
  fin
fin

DQL est un algorithme sans modèle et de politique hors ligne, comme Q-learning, sur lequel il est basé, mais il s'agit bien d'un algorithme approximatif, car il approxime la fonction Q grâce à un réseau de neurones.

DQL est un algorithme très connu, on pourrait même dire que c'est un classique, car il est celui qui a relancé l'apprentissage par renforcement en créant l'apprentissage par renforcement profond. Il est le premier à combiner l'apprentissage par renforcement et l'apprentissage profond.

4.4 Algorithmes basés sur l'optimisation de la politique

L'autre grande famille d'algorithmes sans modèle sont ceux basés sur l'optimisation de la politique. Là où les algorithmes basés sur Q-learning apprenaient une fonction de valeur état-action optimale $Q^*(s,a)$ (ou une approximation $Q(s,a,\theta)$), les algorithmes basés sur l'optimisation de la politique vont directement apprendre une politique paramétrique $\pi(a|s,\theta)$. Ce sont généralement des algorithmes de politique en ligne. L'algorithme Proximal Policy Optimization (PPO) en est un représentant.

4.4.1 Policy Proximal Optimization

Introduit dans (Schulman, Wolski et al. 2017), PPO est un algorithme de RL qui améliore les politiques de manière itérative en utilisant des méthodes de gradient. Cet algorithme peut être décrit comme suit :

1. Initialisation : l'algorithme commence par initialiser la politique de l'agent. Cela peut être fait avec une politique aléatoire ou avec une politique pré-entraînée.
2. Exploration de l'environnement : l'agent interagit avec l'environnement en choisissant des actions selon la politique actuelle. Les états d'entrée et les récompenses reçues sont enregistrés.
3. Collecte de l'ensemble de données : l'ensemble de données est collecté en enregistrant les observations, les actions, les récompenses et autres informations pertinentes lors de l'exploration de l'environnement.
4. Calcul de l'avantage : l'avantage est calculé pour chaque étape, ce qui permet de quantifier à quel point chaque action choisie était meilleure que les autres actions possibles à cet état.
5. Mises à jour de la politique : la politique est mise à jour en effectuant plusieurs itérations. À chaque itération, un mini-batch d'échantillons est extrait de l'ensemble de données. Les gradients sont calculés à l'aide de ratio d'action (ou probabilité ratio) entre la nouvelle politique et l'ancienne politique. Les gradients sont ensuite utilisés pour mettre à jour la politique à l'aide d'une méthode de descente de gradient stochastique.
6. Évaluation de la nouvelle politique : après chaque mise à jour, la nouvelle politique est évaluée en utilisant l'environnement. Cette évaluation permet de comparer les performances de la nouvelle politique avec celles de l'ancienne.
7. Répétition des étapes 3 à 6 : les étapes 3 à 6 sont répétées jusqu'à ce que la politique convergente soit trouvée. Cela signifie que la politique ne change plus ou que les performances se stabilisent à un certain niveau.

L'algorithme PPO est conçu pour être stable et éviter des mises à jour de la politique trop importantes à chaque itération. Il utilise une méthode appelée recoupe du ratio d'action (Clipped Action Ratio) pour limiter les écarts entre les anciennes et les nouvelles politiques. Cela permet de contrôler la taille des mises à jour de la politique et de maintenir la stabilité de l'apprentissage. L'algorithme complet est donné dans l'Algorithme 7.

4.5 Lier MDP et optimisation de la politique

Chaque famille d'algorithmes vue précédemment a des avantages et des inconvénients. Par exemple, les méthodes basées sur Q-learning ont tendance à être instables (l'agent apprend, puis il oublie ses connaissances ou stagne, voir les sections 11.2 et 11.3 de(Sutton et Barto 2018)). À l'inverse, les méthodes basées sur l'optimisation de la politique sont réputées stables. En revanche, généralement, les méthodes basées sur Q-learning sont plus efficaces, c.-à-d., elles obtiennent de meilleures performances.

Ces deux familles d'algorithmes ne sont cependant pas incompatibles entre elles. Dans certains cas, elles sont même équivalentes (voir (Schulman, Chen et Abbeel 2017)). D'où l'idée de les mélanger afin d'en tirer le meilleur tout en réduisant leurs faiblesses.

4.5.1 Soft Actor-Critic

L'algorithme Soft Actor-Critic (SAC) a été introduit dans (Haarnoja et al. 2018) et a été développé pour résoudre des problèmes d'apprentissage par renforcement avec des espaces d'actions continues, ce qui signifie que les actions ne sont pas restreintes à un ensemble fini de choix discrets. L'algorithme SAC utilise l'optimisation de la politique et de la fonction de valeur pour apprendre simultanément. Voici les principales étapes de l'algorithme SAC :

Algorithme 7 : Policy Proximal Optimization (PPO)

Données : Paramètres de l'algorithme
Résultat : Politique optimale
Initialiser les politiques π_θ et $\pi_{\theta_{\text{anc}}}$ aléatoirement;
Initialiser la fonction de valeur V_ϕ aléatoirement;
Initialiser l'ensemble des épisodes vides $D = \{\}$;
pour *itération* $= 1$ *à nombre_itérations* **faire**
    pour chaque *épisode dans nombre_épisodes* **faire**
        Collecter les transitions $(s, a, r, s', \text{done})$ en utilisant la politique π_θ;
        Ajouter l'épisode $(s, a, r, s', \text{done})$ à D;
    fin
    Optimiser la fonction de valeur V_ϕ en minimisant l'erreur quadratique moyenne sur D;
    pour *époque* $= 1$ *à nombre_époques_PPO* **faire**
        Échantillonner un mini-batch d'échantillons $(s, a, r, s', \text{done})$ à partir de D;
        Calculer les avantages normalisés $\hat{A}$ en utilisant la fonction de valeur V_ϕ;
        Mettre à jour π_θ en maximisant la fonction objectif PPO;
    fin
    Copier les politiques π_θ à $\pi_{\theta_{\text{anc}}}$;
fin

1. Initialisation : l'algorithme initialise les paramètres de la politique, de la fonction de valeur et des modèles d'entropie avec des valeurs aléatoires.
2. Collecte des données : l'agent interagit avec l'environnement pour collecter un ensemble de trajectoires d'expérience en utilisant sa politique actuelle. Ces trajectoires d'expérience seront utilisées pour mettre à jour les modèles de la politique et de la fonction de valeur.
3. Mise à jour des valeurs Q : à partir des trajectoires d'expérience collectées, l'algorithme mettra à jour la fonction de valeur Q en utilisant la méthode double-Q learning, où deux estimateurs de la fonction de valeur sont utilisés pour réduire les biais et variances de l'estimation.
4. Mise à jour de la politique : l'algorithme utilise les estimateurs de la fonction de valeur Q pour calculer le gradient de la politique. Ce gradient est ensuite utilisé pour mettre à jour les paramètres de la politique afin d'améliorer les performances de la politique.
5. Mise à jour de l'entropie : l'algorithme utilise un modèle d'entropie pour contrôler la distribution des actions produites par la politique. L'entropie est mise à jour au fur et à mesure de l'apprentissage pour réguler l'exploration de la politique.
6. Répéter : les étapes 2 à 5 sont répétées jusqu'à ce que la politique ait convergé ou qu'un certain critère d'arrêt soit atteint.

L'algorithme Soft Actor-Critic s'appuie sur la théorie de l'optimisation de l'information mutuelle pour guider l'apprentissage de la politique et de la fonction de valeur. En utilisant cette approche, l'algorithme est capable de gérer efficacement les problèmes à espace d'action continue et d'apprendre des politiques stochastiques efficaces. L'algorithme complet est décrit dans l'algorithme 8.

Algorithme 8 : Soft Actor-Critic (SAC)

Entrées : Modèle de la dynamique de l'environnement M, politique conditionnelle déterministe $\pi_\theta(a|s)$, fonction de valeur d'état $V_\phi(s)$, fonction d'avantage $A(s,a)$, taille de la mémoire de relecture N, taille minimale d'échantillon B, nombre d'itérations T

Résultat : Politique π_θ

Initialiser les paramètres θ, ϕ, θ' et ϕ'
Initialiser la mémoire de relecture D avec une taille maximale N
pour $t = 1$ **à** T **faire**
    Générer une trajectoire d'expérience $\tau = \{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^{H}$ en utilisant π_θ et M
    Stocker τ dans D
    si $|D| \geq B$ **alors**
        Échantillonner un mini-lot B de D
        Mettre à jour ϕ en utilisant la régression de moindres carrés pour minimiser l'erreur quadratique moyenne $\frac{1}{|B|}\sum_{(s,a,r,s')\in B}(V_\phi(s) - y_V(s,a))^2$, avec $y_V(s,a) = r + \gamma \mathbb{E}_{a'\sim\pi_{\theta'}(\cdot|s')}[Q_\phi(s',a')] - \alpha \log \pi_\theta(a|s)$
        Mettre à jour θ en utilisant la descente de gradient stochastique pour minimiser la fonction objective $\frac{1}{|B|}\sum_{s\in B}\mathbb{E}_{a\sim\pi_\theta(\cdot|s)}[A(s,a) - \alpha \log \pi_\theta(a|s)]$
        Mettre à jour $\phi' \leftarrow \tau\phi + (1-\tau)\phi'$ et $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$ pour les actualiser vers ϕ et θ
    fin
fin
retourner π_θ

4.6 Conclusion

Dans la suite, nous allons principalement utiliser les algorithmes SAC et PPO. Le but de ces algorithmes est de permettre l'exploration des résultats sur des espaces de données très grands et non structurés. Il est intéressant de comparer les performances de ces deux algorithmes avant de décider lequel est le plus adapté au problème. PPO utilise l'apprentissage de politiques en ligne pour apprendre le travail pratique par l'évaluation des politiques existantes et évalue l'environnement, tandis que SAC utilise l'apprentissage de politiques hors ligne et évalue l'environnement par le biais de la politique précédente. PPO soutient l'exploration en utilisant la méthode de l'entropie, tandis que SAC atteint un équilibre unique entre l'exploration et l'exploitation en ajoutant l'entropie à son but ultime. Une prévision d'entropie faible a une forte confiance dans le choix d'un événement, tandis qu'une prévision d'entropie élevée est incertaine quant à l'événement à choisir. En général, SAC est plus efficace, mais a besoin de plus de temps pour converger que PPO.

L'apprentissage par renforcement est considéré comme la forme de ML la plus difficile. Dans sa forme originelle, la principale difficulté consistait à modéliser correctement l'environnement, en particulier le trio état-action-récompense. La récompense est de loin la composante la plus difficile à définir. Le fait d'avoir ajouté des réseaux de neurones aux algorithmes n'a pas aidé. Depuis que l'algorithme Deep Q-learning (voir Sous-section 4.3.2) a introduit le DRL, la difficulté s'est accrue. Avant cela, on pouvait démontrer qu'un algorithme convergeait (c'est le cas pour Q-learning). Malheureusement, il est très difficile, voire impossible, de prévoir le comportement d'un réseau de neurones (leurs résultats sont régulièrement sources de découvertes fortuites). La combinaison de ces deux types d'apprentissage difficile (par renforcement et profond) fait qu'il est extrêmement difficile, voir, dans la plupart des cas, impossible, de prévoir le comportement d'un algorithme de DRL (Henderson et al. 2018). L'auteur de ce très intéressant post (Irpan 2018) de 2018, sans doute un peu pessimiste, résume bien l'ensemble des difficultés que l'on peut rencontrer en faisant du DRL. Malgré cela, le RL reste une technologie prometteuse afin de permettre de configurer le TAS de TSN. Ainsi, l'apprentissage par renforcement est, par exemple, d'ores et déjà utilisé pour le routage dans les réseaux informatiques (Mammeri 2019). Cependant, son usage dans les réseaux TSN est encore limité, comme nous allons le voir dans le chapitre suivant.

Bibliographie du présent chapitre

Arulkumaran, Kai et al. (nov. 2017). « Deep Reinforcement Learning : A Brief Survey ». In : *IEEE Signal Processing Magazine* 34.6, p. 26-38. issn : 1558-0792. doi : 10.1109/MSP.2017.2743240.

Bellman, Richard (1957). « A Markovian decision process ». In : *Journal of mathematics and mechanics*, p. 679-684.

Haarnoja, Tuomas et al. (2018). « Soft actor-critic : Off-policy maximum entropy deep reinforcement learning with a stochastic actor ». In : *International conference on machine learning*. PMLR, p. 1861-1870.

Henderson, Peter et al. (2018). « Deep reinforcement learning that matters ». In : *Proceedings of the AAAI conference on artificial intelligence*. T. 32. 1.

Hornik, Kurt (1991). « Approximation capabilities of multilayer feedforward networks ». In : *Neural Networks* 4.2, p. 251-257. issn : 0893-6080. doi : 10.1016/0893-6080(91)90009-T. url : https://www.sciencedirect.com/science/article/pii/089360809190009T.

Irpan, Alex (2018). *Deep Reinforcement Learning Doesn't Work Yet*. https://www.alexirpan.com/2018/02/14/rl-hard.html.

Li, Yuxi (2018). « Deep reinforcement learning ». In : *arXiv preprint arXiv :1810.06339*.

Mammeri, Zoubir (2019). « Reinforcement Learning Based Routing in Networks : Review and Classification of Approaches ». In : *IEEE Access* 7, p. 55916-55950. issn : 2169-3536. doi : 10.1109/ACCESS.2019.2913776.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves et al. (2013). « Playing atari with deep reinforcement learning ». In : *arXiv preprint arXiv :1312.5602*.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A Rusu et al. (2015). « Human-level control through deep reinforcement learning ». In : *nature* 518.7540, p. 529-533.

Schulman, John, Xi Chen et Pieter Abbeel (2017). « Equivalence between policy gradients and soft q-learning ». In : *arXiv preprint arXiv :1704.06440*.

Schulman, John, Filip Wolski et al. (2017). « Proximal policy optimization algorithms ». In : *arXiv preprint arXiv :1707.06347*.

Sutton, Richard S et Andrew G Barto (2018). *Reinforcement learning : An introduction*. MIT press.

Watkins, Christopher JCH et Peter Dayan (1992). « Q-learning ». In : *Machine learning* 8.3-4, p. 279-292.

Watkins, Christopher John Cornish Hellaby (1989). « Learning from delayed rewards ». Thèse de doct.

Chapitre 5

État de l'art

Sommaire du présent chapitre

La question de la reconfiguration de l'ordonnancement en ligne dans TSN demeure un défi de taille et ouvert (Deng et al. 2022 ; Stüber et al. 2023). En particulier, chercher un bon ordonnancement est un problème NP-difficile connu (Serna Oliver, Silviu S. Craciunas et Steiner 2018 ; Silviu S. Craciunas et al. 2016 ; Tindell, Burns et Wellings 1992 ; Burns 1991 ; Leung et Whitehead 1982). De nombreux travaux ont été menés afin de résoudre ce problème.

5.1 Méthodes conventionnelles de calcul de l'ordonnancement du trafic

Il existe des algorithmes de configurations généralistes comme ceux dits de Design-Space Exploration (DSE) qui peuvent être utilisés afin d'aider à la configuration des réseaux TSN, par exemple, ZeroConfig-TSN (ZCT). Ces algorithmes contiennent une évaluation de performances pour vérifier que les contraintes de temps sont respectées. Elle peut consister en soit une analyse d'ordonnançabilité, c.-à-d., vérifier que toutes les tâches peuvent être finies avant leur échéance, appelée aussi analyse de faisabilité, soit une simulation. Par exemple, le travail effectué dans (Ashjaei et al. 2017) offre une analyse du trafic en temps réel qui traverse le réseau TSN, permettant d'évaluer si les contraintes de temps sont respectées. La méthode conventionnelle de calcul d'un ordonnancement déterministe pour le TAS implique l'utilisation d'une formulation de programmation linéaire entière (Integer Linear Programming (ILP)) (Silviu S. Craciunas et al. 2016 ; Nayak, Dürr et Rothermel 2016) ou d'un solveur de théories de modulo (Satisfiability Modulo Theories (SMT)) (A. C. T. d. Santos, Schneider et Nigam 2019 ; Serna Oliver, Silviu S. Craciunas et Steiner 2018 ; Silviu S Craciunas, Oliver et Ag 2017). Un autre exemple est (Atallah, Hamad et Mohamed 2018), qui propose un algorithme capable

de générer une topologie résiliente aux pannes qui garantit un routage et un ordonnancement faisables pour le trafic sensible au temps. Bien que ces approches soient efficaces pour de petits réseaux, il peut falloir beaucoup de temps pour converger pour des réseaux plus grands. En outre, il s'agit de techniques hors ligne qui ne conviennent pas aux réseaux ouverts et reconfigurables (Ghotra, Helm et Jaeger 2023). Dans (Shih et al. 2023), les auteurs proposent un mécanisme d'ordonnancement dans des réseaux 5G. Leur méthode tient compte de la durée de résidence dans le système/domaine 5G, et vise à améliorer l'ordonnancement. Les auteurs formulent l'ordonnancement comme un problème SMT et proposent des contraintes pour générer un ordonnancement des flux critiques dans les réseaux TSN-5G. Ils analysent les exigences des flux critiques pour améliorer l'ordonnançabilité et les transforment en contrainte d'agrégation de flux. Cependant, ils mentionnent qu'il y a un compromis entre l'ordonnançabilité et la robustesse à l'incertitude lors du maintien du déterminisme dans les environnements sans fil. En outre, l'ordonnançabilité peut diminuer à mesure que le nombre de flux augmente.

Dans (Xue et al. 2023), les auteurs proposent d'ordonnancer le trafic en se basant sur la virtualisation du réseau. Grâce au paradigme SDN, les queues sont virtualisées, ce qui permet une plus grande modularité. Un algorithme s'occupe alors d'ordonnancer le trafic en allouant des queues virtuelles puis en faisant un mappage avec les queues réelles. Cependant, ces outils ne sont pas adaptés à la configuration du TAS en réseau dynamique. Tout d'abord, pour fournir une bonne configuration à déployer, ces outils nécessitent une connaissance préalable de tous les flux qui pourraient être présents dans le réseau. En ce sens, si un changement se produit dans le réseau, l'ordonnancement calculé n'est plus efficace. De plus, dans le cas où de nouveaux flux sont ajoutés au réseau, la génération d'un nouvel ordonnancement via ces outils d'ingénierie peut prendre plusieurs heures (Navet, Mai et Migge 2019) avant que les nouveaux flux puissent être acceptés/rejetés, et que la configuration décidée puisse être déployée.

Les auteurs de (Gärtner, Rizk, Koldehofe, Guillaume et al. 2023) proposent une amélioration de ces méthodes en utilisant le concept de courbes flexibles, introduit dans (Gärtner, Rizk, Koldehofe, Hark et al. 2021). Ces courbes sont une notion de flexibilité de l'ordonnancement pour un chemin donné dans un réseau, indiquant le nombre d'arrangements possibles dans l'ordonnancement de goulot d'étranglement le long du chemin donné. Les auteurs proposent une méthode d'admission flexible qui permet une admission des flux plus rapide que les méthodes présentées plus haut et fournit une notion de flexibilité des ordonnancements TSN en tenant compte des délais. Cette méthode n'est cependant testée que sur un scénario simple avec un seul flux.

Les auteurs de (Samson et al. 2023) proposent quant à eux une combinaison de modélisation de réseau et de techniques de génie logiciel à base de composants pour calculer automatiquement le modèle des flux de données qui seront échangés sur le réseau. Cette approche utilise une spécification unique pour générer à la fois les modèles de flux de données et le code d'infrastructure de l'application, garantissant ainsi la cohérence entre le comportement du système réel et la configuration du réseau. Ces modèles incluent la configuration du TAS. Cependant, la spécification de l'architecture et du comportement de l'application sont à la charge de l'utilisateur. De plus, plus la topologie du réseau est grande, moins cette méthode est efficace en terme de temps de calcul.

Un autre exemple est décrit dans (Nasrallah, Balasubramanian et al. 2019). Cette solution, basée sur le standard IEEE 802.1Qcc (« IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks – Amendment 31 : Stream Reservation Protocol (SRP) Enhancements and Performance Improvements » 2018), permet la reconfiguration en ligne d'un réseau basé sur IEEE 802.1Qbv. Cependant, il repose sur un mécanisme de contrôle d'admission et ne modifie pas le cycle de temps du TAS dans les commutateurs.

Enfin, les auteurs de (Raagaard et al. 2017) proposent un algorithme heuristique pour déter-

miner les GCL au moment de l'exécution afin que les délais soient respectés et que l'utilisation de la file d'attente soit minimisée pour tenir compte du trafic non critique dans le cadre très spécifique du Fog computing.

Dans des cas plus généralistes, comme le nôtre, où dynamicité, passage à l'échelle et non-connaissance préalable des flux sont des caractéristiques du réseau, les algorithmes présentés plus haut risquent de ne pas suffire, d'où l'idée de se faire aider par une IA afin d'effectuer les calculs.

5.2 Méthodes utilisant l'IA pour le calcul de l'ordonnancement du trafic

Il existe deux manières d'utiliser une IA : la première, afin de faciliter le calcul de l'ordonnancement ; la seconde, dans le but de calculer l'ordonnancement.

5.2.1 Utiliser l'IA afin d'aider à ordonnancer

Dans (Bezerra et al. 2022), le ML est utilisé pour créer un modèle de substitution basé sur la régression qui permet la création de modèles d'optimisation pour trouver des configurations optimales pour un réseau TSN donné. Son objectif principal est de développer une solution par laquelle un ingénieur réseau, en utilisant le modèle proposé, peut prédire la latence pour une configuration réseau donnée. Les techniques d'apprentissage automatique sont appliquées pour en savoir plus sur le comportement du réseau, en extraire les principales fonctionnalités et créer des modèles représentatifs qui peuvent être utilisés pour prédire ses performances. Le modèle proposé peut prédire la latence (sortie du modèle) pour une configuration réseau donnée (paramètres d'entrée du module).

Plusieurs travaux se sont concentrés sur le lien entre routage et ordonnancement. Ainsi, dans (Pahlevan et Obermaisser 2018), les auteurs partent du constat que les solutions existantes ne considérant que les contraintes d'ordonnancement utilisent des routes fixes qui sont générées séparément. Cette abstraction simplifiée du problème d'ordonnancement peut conduire à l'échec de la génération d'ordonnancement, bien que le système soit ordonnancable. Pour résoudre ce problème, les auteurs considèrent l'impact du routage sur les contraintes d'ordonnancement et utilisent des contraintes de routage et d'ordonnancement conjointes pour résoudre le problème d'ordonnancement avec l'aide d'algorithmes génétiques. Cette approche peut cependant mener à un problème de lenteur d'exécution sur de large réseaux. Pour résoudre ce problème, les auteurs de (Min et al. 2023) étudient le problème induit par des ordonnancements basés sur des itinéraires préconfigurés sans explorer d'alternatives pour un meilleur ordonnancement, sachant que les méthodes qui considèrent conjointement le routage et l'ordonnancement nécessitent d'énormes ressources d'exécution et de calcul. Pour résoudre ce problème, une solution basée sur le RL identifie des itinéraires sur lesquels la charge est équilibrée pour un meilleur ordonnancement avec une complexité acceptable. Les auteurs de (Yang et al. 2022) traitent du même problème, mais en utilisant de l'apprentissage par renforcement profond basé sur des réseaux convolutifs de graphes afin de déterminer le routage. Enfin, une étude réalisée par les auteurs de (Yu, Taleb et Zhang 2023) a utilisé le DRL pour gérer le routage et l'ordonnancement du trafic à criticité mixte dans un cadre de réseaux déterministes DetNet, qui fonctionne à la couche 3 du modèle OSI, là où TSN fonctionne à la couche 2.

Les auteurs de (Zhou et Cheng 2021) utilisent l'algorithme de RL DDPG pour modéliser l'incertitude causée par des facteurs influençant la transmission tels que les erreurs de synchronisation temporelle. En utilisant DDPG, ils réduisent considérablement le nombre de

comportements erronés dans les scénarios TSN et améliorent les performances de retard du réseau. L'ordonnancement, quant à lui, est trouvé en résolvant un problème d'optimisation.

Dans (Zhu et al. 2023), les auteurs traitent de l'ordonnancement dans des réseaux industriels 5G. Le but est de fournir une garantie de latence pour les applications à déclenchement temporel sur 5G et TSN ainsi qu'une garantie de débit pour les applications vidéo dans les systèmes 5G. La solution proposée utilise le DRL pour décider quels flux peuvent être ordonnancés ainsi que le nombre de ressources allouées pour la livraison de paquets. Le but est de réaliser un compromis entre les applications à déclenchement temporel et les applications vidéos en optimisant conjointement l'allocation des ressources radio.

Toutes ces méthodes s'appuient sur une IA afin d'aider à l'ordonnancement, cependant, ces solutions ne permettent pas de calculer l'ordonnancement des flux.

5.2.2 Utiliser l'IA afin d'ordonnancer directement

Dans (Navet, Mai et Migge 2019 ; Mai, Navet et Migge 2019a ; Mai, Navet et Migge 2019b), les chercheurs remplacent l'analyse d'ordonnançabilité des algorithmes DSE par une IA qui doit déterminer si une configuration possible est faisable, c'est-à-dire si elle respecte les critères applicatifs. Pour ce faire, ils testent des algorithmes simples (k-NN, k-Means) d'apprentissages supervisés et non supervisés afin de classer les configurations possibles en faisables ou non faisables. Ils comparent les performances de leur IA à une analyse d'ordonnançabilité plus classique (en l'occurrence, une analyse du pire cas pour chaque flux). Il en ressort un gain de temps dans le cas où l'on a un très grand nombre de configurations possibles, ceci au prix de la précision, qui reste malgré tout bonne. Puis, dans (Mai et Navet 2020 ; Mai et Navet 2021), ils utilisent un paradigme d'apprentissage profond appelé Graph Neural Network (GNN) pour remplacer les algorithmes k-NN et K-Means et obtenir de meilleures performances. Le principal inconvénient de cette solution est que cette IA n'est efficace que pour une topologie spécifique. Lorsque la topologie change, l'IA doit être ré-entraînée. En outre, la solution proposée n'est pas adaptée à la configuration en ligne.

Une autre recherche menée dans (X. Wang et al. 2022) a utilisé le DRL afin de calculer l'ordonnancement pour le TAS. Néanmoins, cette approche adopte le modèle de non-attente pour l'ordonnancement TSN, qui a été initialement introduit dans (Dürr et Nayak 2016), dans lequel il n'y a pas de délai dans les files d'attentes. Par conséquent, l'ordonnancement du trafic sur chaque commutateur par lesquels les flux passent peut être calculé en fonction du moment où chaque paquet est transmis. Cela nécessite des connaissances préalables sur absolument chaque flux.

On le voit, il n'existe pas de solution permettant de calculer directement l'ordonnancement du TAS de manière dynamique, sans connaissances préalables sur les flux et rapidement. En d'autres termes, on souhaite qu'un agent soit capable de trouver un ordonnancement acceptable pour le trafic dans un réseau TSN à partir de seulement :

1. les contraintes applicatives,
2. la topologie du réseau.

C'est ce problème que nous allons essayer de résoudre dans la suite.

Dans la partie suivante de la thèse, nous décrivons comment un agent entraîné à l'aide du RL peut décider un ordonnancement pour les flux dans un réseau TSN. Cet ordonnancement est dans un premier temps identique pour tout le réseau, puis spécifique à chaque commutateur.

Bibliographie du présent chapitre

Ashjaei, Mohammad et al. (2017). « Schedulability analysis of Ethernet Audio Video Bridging networks with scheduled traffic support ». In : *Real-Time Systems* 53.4, p. 526-577.

Atallah, Ayman A., Ghaith Bany Hamad et Otmane Ait Mohamed (juill. 2018). « Fault-Resilient Topology Planning and Traffic Configuration for IEEE 802.1Qbv TSN Networks ». In : *2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS)*, p. 151-156. doi : 10.1109/IOLTS.2018.8474201.

Bezerra, Daniel et al. (2022). « A machine learning-based optimization for end-to-end latency in TSN networks ». In : *Computer Communications* 195, p. 424-440. issn : 0140-3664. doi : 10.1016/j.comcom.2022.09.011. url : https://www.sciencedirect.com/science/article/pii/S0140366422003504.

Burns, Alan (1991). « Scheduling hard real-time systems : a review ». In : *Software Engineering Journal* 6.3, p. 116-128.

Craciunas, Silviu S, R Serna Oliver et T Ag (2017). « An overview of scheduling mechanisms for time-sensitive networks ». In : *Proceedings of the Real-time summer school LÉcole dÉté Temps Réel (ETR)*, p. 1551-3203.

Craciunas, Silviu S. et al. (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*. RTNS '16. Brest, France : Association for Computing Machinery, p. 183-192. isbn : 9781450347877. doi : 10.1145/2997465.2997470. url : https://doi.org/10.1145/2997465.2997470.

Deng, Libing et al. (jan. 2022). « A Survey of Real-Time Ethernet Modeling and Design Methodologies : From AVB to TSN ». In : *ACM Comput. Surv.* 55.2. issn : 0360-0300. doi : 10.1145/3487330. url : https://doi.org/10.1145/3487330.

Dürr, Frank et 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*. RTNS '16. Brest, France : Association for Computing Machinery, p. 203-212. isbn : 9781450347877. doi : 10.1145/2997465.2997494. url : https://doi.org/10.1145/2997465.2997494.

Gärtner, Christoph, Amr Rizk, Boris Koldehofe, René Guillaume et al. (2023). « Fast incremental reconfiguration of dynamic time-sensitive networks at runtime ». In : *Computer Networks* 224, p. 109606. issn : 1389-1286. doi : 10.1016/j.comnet.2023.109606. url : https://www.sciencedirect.com/science/article/pii/S1389128623000518.

Gärtner, Christoph, Amr Rizk, Boris Koldehofe, Rhaban Hark et al. (juin 2021). « Leveraging Flexibility of Time-Sensitive Networks for dynamic Reconfigurability ». In : *2021 IFIP Networking Conference (IFIP Networking)*, p. 1-6. doi : 10.23919/IFIPNetworking52078.2021.9472834.

Ghotra, Vishavjeet, Max Helm et Benedikt Jaeger (2023). « TSN Qbv and Schedule Generation Approaches ». In : *Network* 29.

« IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks – Amendment 31 : Stream Reservation Protocol (SRP) Enhancements and Performance Improvements » (oct. 2018). In : *IEEE Std 802.1Qcc-2018 (Amendment to IEEE Std 802.1Q-2018 as amended by IEEE Std 802.1Qcp-2018)*, p. 1-208. doi : 10.1109/IEEESTD.2018.8514112.

Leung, Joseph Y.-T. et Jennifer Whitehead (1982). « On the complexity of fixed-priority scheduling of periodic, real-time tasks ». In : *Performance Evaluation* 2.4, p. 237-250. issn : 0166-5316. doi : 10.1016/0166-5316(82)90024-4. url : https://www.sciencedirect.com/science/article/pii/0166531682900244.

Mai, Tieu Long et Nicolas Navet (2020). *Deep Learning to Predict the Feasibility of Priority-Based Ethernet Network Configurations*. Rapp. tech. University of Luxembourg.

— (2021). *Improvements to Deep-Learning-based Feasibility Prediction of Switched Ethernet Network Configurations*. Rapp. tech. University of Luxembourg.

Mai, Tieu Long, Nicolas Navet et Jörn Migge (mai 2019a). « A Hybrid Machine Learning and Schedulability Analysis Method for the Verification of TSN Networks ». In : *2019 15th IEEE International Workshop on Factory Communication Systems (WFCS)*, p. 1-8. doi : 10.1109/WFCS.2019.8757948.

— (2019b). « On the Use of Supervised Machine Learning for Assessing Schedulability : Application to Ethernet TSN ». In : *Proceedings of the 27th International Conference on Real-Time Networks and Systems*. RTNS '19. Toulouse, France : Association for Computing Machinery, p. 143-153. isbn : 9781450372237. doi : 10.1145/3356401.3356409. url : https://doi.org/10.1145/3356401.3356409.

Min, Junhong et al. (2023). « Reinforcement learning based routing for time-aware shaper scheduling in time-sensitive networks ». In : *Computer Networks* 235, p. 109983. issn : 1389-1286. doi : 10.1016/j.comnet.2023.109983. url : https://www.sciencedirect.com/science/article/pii/S1389128623004280.

Nasrallah, Ahmed, Venkatraman Balasubramanian et al. (déc. 2019). « Reconfiguration Algorithms for High Precision Communications in Time Sensitive Networks ». In : *2019 IEEE Globecom Workshops (GC Wkshps)*, p. 1-6. doi : 10.1109/GCWkshps45667.2019.9024705.

Navet, Nicolas, Tieu Long Mai et Jörn Migge (2019). *Using machine learning to speed up the design space exploration of Ethernet TSN networks*. Rapp. tech. University of Luxembourg.

Nayak, Naresh Ganesh, Frank Dürr et 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*. RTNS '16. Brest, France : Association for Computing Machinery, p. 193-202. isbn : 9781450347877. doi : 10.1145/2997465.2997487. url : https://doi.org/10.1145/2997465.2997487.

Pahlevan, Maryam et Roman Obermaisser (sept. 2018). « Genetic Algorithm for Scheduling Time-Triggered Traffic in Time-Sensitive Networks ». In : *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*. T. 1, p. 337-344. doi : 10.1109/ETFA.2018.8502515.

Raagaard, Michael Lander et al. (oct. 2017). « Runtime reconfiguration of time-sensitive networking (TSN) schedules for Fog Computing ». In : *2017 IEEE Fog World Congress (FWC)*, p. 1-6. doi : 10.1109/FWC.2017.8368523.

Samson, Maxime et al. (sept. 2023). « Computing Data Streams in Real-Time Networks from Component-Based Software Engineering ». In : *2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA)*, p. 1-8. doi : 10.1109/ETFA54631.2023.10275469.

Santos, Aellison Cassimiro T. dos, Ben Schneider et Vivek Nigam (oct. 2019). « TSNSCHED : Automated Schedule Generation for Time Sensitive Networking ». In : *2019 Formal Methods in Computer Aided Design (FMCAD)*, p. 69-77. doi : 10.23919/FMCAD.2019.8894249.

Serna Oliver, Ramon, Silviu S. Craciunas et Wilfried Steiner (avr. 2018). « IEEE 802.1Qbv Gate Control List Synthesis Using Array Theory Encoding ». In : *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, p. 13-24. doi : 10.1109/RTAS.2018.00008.

Shih, Yuan-Yao et al. (sept. 2023). « Scheduling of Integrated 5G and Time Sensitive Network for Deterministic Communication ». In : *2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA)*, p. 1-8. doi : 10.1109/ETFA54631.2023.10275610.

Stüber, Thomas et al. (2023). « A Survey of Scheduling Algorithms for the Time-Aware Shaper in Time-Sensitive Networking (TSN) ». In : *IEEE Access* 11, p. 61192-61233. issn : 2169-3536. doi : 10.1109/ACCESS.2023.3286370.

Tindell, Ken W, Alan Burns et Andy J. Wellings (1992). « Allocating hard real-time tasks : an NP-hard problem made easy ». In : *Real-Time Systems* 4.2, p. 145-165.

Wang, Xiaolong et al. (avr. 2022). « Deep Reinforcement Learning aided No-wait Flow Scheduling in Time-Sensitive Networks ». In : *2022 IEEE Wireless Communications and Networking Conference (WCNC)*, p. 812-817. doi : 10.1109/WCNC51071.2022.9771665.

Xue, Junli et al. (2023). « Scheduling Time-Critical Traffic with Virtual Queues in Software-Defined Time-Sensitive Networking ». In : *IEEE Transactions on Network and Service Management*, p. 1-1. issn : 1932-4537. doi : 10.1109/TNSM.2023.3287634.

Yang, Liu et al. (déc. 2022). « Joint Routing and Scheduling Optimization in Time-Sensitive Networks Using Graph-Convolutional-Network-Based Deep Reinforcement Learning ». In : *IEEE Internet of Things Journal* 9.23, p. 23981-23994. issn : 2327-4662. doi : 10.1109/JIOT.2022.3188826.

Yu, Hao, Tarik Taleb et Jiawei Zhang (août 2023). « Deep Reinforcement Learning-Based Deterministic Routing and Scheduling for Mixed-Criticality Flows ». In : *IEEE Transactions on Industrial Informatics* 19.8, p. 8806-8816. issn : 1941-0050. doi : 10.1109/TII.2022.3222314.

Zhou, Boyang et Liang Cheng (2021). « Mitigation of Scheduling Violations in Time-Sensitive Networking Using Deep Deterministic Policy Gradient ». In : *Proceedings of the 4th FlexNets Workshop on Flexible Networks Artificial Intelligence Supported Network Flexibility and Agility*. FlexNets '21. Virtual Event, USA : Association for Computing Machinery, p. 32-37. isbn : 9781450386340. doi : 10.1145/3472735.3473385. url : https://doi.org/10.1145/3472735.3473385.

Zhu, Yuan et al. (2023). « Deep Reinforcement Learning-Based Joint Scheduling of 5G and TSN in Industrial Networks ». In : *Electronics* 12.12. issn : 2079-9292. doi : 10.3390/electronics12122686. url : https://www.mdpi.com/2079-9292/12/12/2686.

Deuxième partie

Contributions

Chapitre 6

Introduction aux contributions

Sommaire du présent chapitre

Durant cette thèse, plusieurs séries d'expériences ont été menées. Ces expériences reposent cependant sur un socle commun.

6.1 Hypothèses communes aux contributions

Dans le contexte de ces travaux, nous utilisons un modèle simplifié du réseau TSN afin d'améliorer la vitesse de l'entrainement.

Les hypothèses suivantes sont formulées :

1. Il n'y a aucune interférence extérieure. Cette hypothèse est crédible dans la mesure où les réseaux sont filaires
2. Une synchronisation temporelle parfaite est supposée, ce qui signifie que le protocole gPTP est configuré et fonctionne correctement. Cette hypothèse est également crédible dans la mesure où la configuration du TAS repose sur le bon fonctionnement du gPTP.
3. Tous les commutateurs ont le même délai de traitement et toutes les liaisons ont la même capacité, ce qui se traduit par les mêmes délais de transmission et de propagation. Par conséquent, un ordonnancement bien conçu minimisera le délai d'attente dans les commutateurs pour le trafic critique, ce qui réduira en fin de compte le délai de bout en bout.
4. On n'utilise que deux classes de trafic différentes et donc deux priorités de trafic différentes (0 et 1 : 0 pour le trafic BE, 1 pour le trafic critique). On procède ainsi pour limiter les simulations et, par conséquent, accélérer le processus d'entraînement.

Dans cette thèse, on considère trois types de topologies différentes : linéaire, en anneau et maillée (comme sur la Figure 6.1). Cette diversité est cruciale, car elle permet à l'agent de décider de la configuration du TAS quelle que soit la topologie. La taille des paquets est déterminée

aléatoirement à chaque nouvel épisode. Elle est comprise entre 94 octets (40 octets de données + 54 octets d'en-tête) et 1 500 octets (pour ne pas dépasser le Maximum Transmission Unit (MTU)) et reste constant pour chaque flux au cours de l'épisode. Cet intervalle provient des tables 5 à 13 du brouillon du profil TSN pour l'automatisation industrielle, dans sa version 1.2. [1]

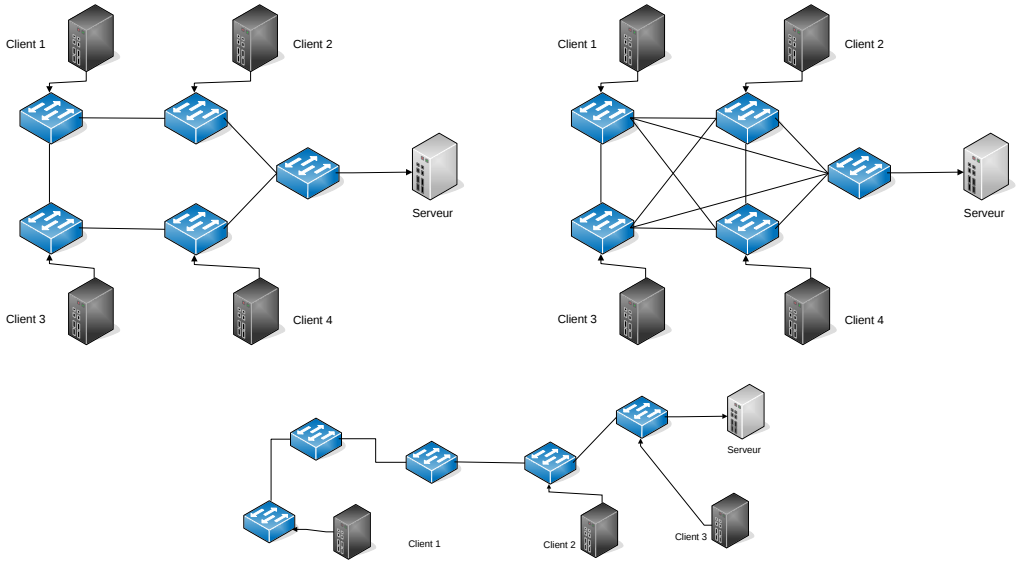


FIGURE 6.1 – Topologies de réseau prises en compte lors de la phase d'entraînement (le nombre de clients varie)

Dans cette série d'expériences, deux classes de trafic sont considérées : le trafic critique (TSN) avec une priorité élevée et le trafic « normal » ou BE, avec une priorité basse. On ne considère donc que deux files d'attente et portes. Cette réduction du nombre de classes de trafic permet de rationaliser l'espace des actions. Cela permet aussi de simplifier l'ordonnancement des flux : il n'y a en effet que deux séquences de temps. La première est utilisée par le trafic critique tandis que la seconde l'est par le reste du trafic.

6.2 Implémentation

L'environnement est modélisé grâce à une simulation d'un réseau TSN. Utiliser des simulations afin d'entrainer une IA par renforcement dont le but est d'agir sur le monde « réel » a déjà été fait, par exemple dans (HWANGBO et al. 2019).

Un simulateur reconnu lorsqu'il s'agit d'effectuer des simulations de réseaux informatiques est OMNeT++ [2]et ses extensions. OMNeT++ est à la fois une librairie et un cadre permettant de construire des simulations de réseaux informatiques. Les différents protocoles ne sont pas implémentés directement dans OMNeT++, pour cela, il faut ajouter INET [3]. INET peut être vu comme une extension d'OMNeT++ qui fourni les différents modèles de réseau informatique (protocoles, applications). OMNeT++ et INET sont développés conjointement par la même équipe, le code est open-source. En résumé, OMNeT++ est la base sur laquelle on construit les

1. Disponible à l'adresse `https://1.ieee802.org/tsn/iec-ieee-60802/`

simulations et INET fournit les modèles réseaux pour ces mêmes simulations. D'ailleurs, INET est lui-même une librairie et un cadre pouvant être étendu par d'autres extensions. Le groupe de travail IEEE 802.1 TSN a recommandé l'utilisation d'OMNeT++ pour effectuer des simulations de réseau TSN [4].

Une fois la simulation terminée, les résultats sont stockés au format SQLite. L'agent peut récupérer la latence de bout en bout par paquet et le nombre de paquets envoyés et reçus par flux à partir des résultats obtenus. En outre, la base de données SQLite contient un vecteur d'états des portes du TAS, le nombre de paquets en attente dans chaque file d'attente et d'autres informations pertinentes qui aident à évaluer l'ordonnancement de l'agent. Ces fonctionnalités permettent la gestion de l'environnement à partir d'un script Python, y compris la modification de la configuration, l'initiation de la simulation et l'analyse des résultats.

L'agent RL utilise la bibliothèque Stable-Baselines3 (Raffin et al. 2021) pour l'implémentation des algorithmes SAC et PPO, et les scripts d'entraînement et d'évaluation de RL Baselines3 Zoo (Raffin 2020). Il s'agit d'une collection d'implémentations RL fiables et open source en Python. Stable-Baselines3 s'appuie sur PyTorch (Paszke et al. 2019) pour les réseaux de neurones.

L'un des aspects les plus difficiles de RL est la mise en œuvre de la communication entre l'agent et l'environnement. Cependant, OMNeT++ n'est pas inclus dans la liste des environnements pris en charge. En conséquence, un module a été développé qui gère la communication entre l'agent et le simulateur OMNeT++. Son rôle est essentiellement de faire l'interface entre OMNeT++ et Stable-Baselines3. Ce module a deux responsabilités principales :

1. interpréter les actions fournies par l'agent et les convertir en instructions compréhensibles par OMNeT++,
2. traduire les résultats de simulation de la base de données SQLite en une récompense et un nouvel état pouvant être compris par l'agent.

Au début de chaque épisode, il génère un environnement qui peut être utilisé par l'agent RL. Pour cela, il génère la simulation OMNeT++/INET et il configure la topologie, après quoi l'agent peut la lancer.

4. Disponible à l'adresse `https://omnetpp.org/`
4. Disponible à l'adresse `https://inet.omnetpp.org/`
4. Voir `https://1.ieee802.org/protocol-simulations/`

Bibliographie du présent chapitre

Hwangbo, Jemin et al. (2019). « Learning agile and dynamic motor skills for legged robots ». In : *Science Robotics* 4.26, eaau5872.

Paszke, Adam et al. (2019). « PyTorch : An Imperative Style, High-Performance Deep Learning Library ». In : *Advances in Neural Information Processing Systems 32*. Sous la dir. de H. Wallach et al. Curran Associates, Inc., p. 8024-8035. url : http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Raffin, Antonin (2020). *RL Baselines3 Zoo*. https://github.com/DLR-RM/rl-baselines3-zoo.

Raffin, Antonin et al. (2021). « Stable-Baselines3 : Reliable Reinforcement Learning Implementations ». In : *Journal of Machine Learning Research* 22.268, p. 1-8. url : http://jmlr.org/papers/v22/20-1364.html.

Chapitre 7

Configurer le TAS de manière identique sur chaque commutateur

Sommaire du présent chapitre

Ce chapitre présente un premier modèle d'un agent RL permettant de configurer l'ordonnancement dans le TAS. Cette première série d'expérimentations a consisté à évaluer la possibilité de calculer un ordonnancement avec du RL dans le cadre de TSN sur un environnement très simplifié.

7.1 Modélisation

Toutes les expériences reposent sur une base commune présentée dans le chapitre précédent. Dans cette section, les éléments spécifiques à cette série d'expériences sont décrits.

7.1.1 Modélisation du réseau TSN

Une hypothèse supplémentaire est formulée sur le réseau TSN : l'ordonnancement est le même sur chaque commutateur. Cette hypothèse mène à ne considérer que des topologies

linéaires. Le nombre de commutateurs, quant à lui, peut varier de 1 à 10. Les liens ont une capacité de 100 ou 1 000 Mbit/s. Ce chiffre varie selon les simulations, cependant, au sein d'une même simulation, il est identique sur chaque lien.

7.1.2 Modélisation du trafic

À chaque nouvel épisode, un nouvel intervalle d'émission est généré aléatoirement pour chaque flux. Ces valeurs sont comprises dans l'intervalle : [240μs, 1 000 μs]. Une fois ces valeurs choisies, l'émission des paquets se fait à intervalle constant.

7.1.3 Paramètres du TAS à configurer

La durée de la séquence de temps réservée au trafic critique est égale à deux fois la durée pour envoyer une trame en nanosecondes. On choisit cette durée afin de s'assurer que le trafic critique a bien le temps d'arriver à destination. De plus, la période du trafic critique est en début de cycle.

La configuration de l'ordonnancement se résume donc à un seul paramètre : la durée du cycle d_{cycle} en nanosecondes.

La Figure 7.1 montre graphiquement les paramètres de configuration TAS que l'agent doit configurer. On remarque que l'agent peut décider d'ouvrir toutes les portes en même temps.

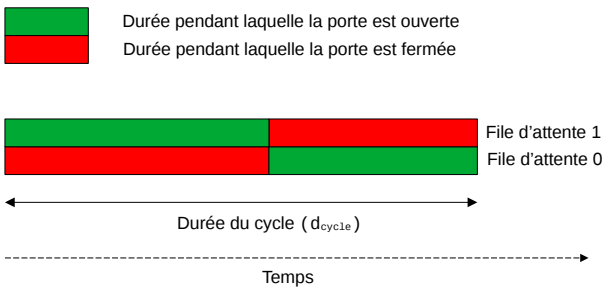


Figure 7.1 – Paramètres du TAS à configurer

7.2 Formulation du problème en RL

La définition du MDP est spécifique à chaque expérience. Il s'agit de bien définir les composantes état, action, récompense.

7.2.1 Définition du MDP

État

La composition détaillée de l'état est donnée dans le Tableau 7.1. L'état est un vecteur comportant onze entrées.

Deux types d'états terminaux sont identifiés :

Catégorie	Nom	Espace des états
Topologie	Nombre de stations émettrices	2
	Nombre de stations réceptrices	2
	Nombre de commutateurs	$\in [1;10]$
	Capacité des liens (Mbit/s)	100 ou 1000
Flux	Longueur des trames (octets)	$\in [94;1\,500]$
	Intervalle d'émission (μs)	$\in [240;1\,000]$
Contraintes	Latence maximale du trafic critique (μs)	$\in [100;20\,000]$

Tableau 7.1 – L'état et l'espace des états

1. l'agent réussit, la latence des paquets du flux critique est inférieure à celle fixée comme limite et l'on n'a pas de perte de paquets trop importante pour chaque flux (arbitrairement, 95% des paquets critiques et 50% des paquets BE envoyés sont arrivés), ce qui implique que le problème est résolu et qu'une configuration du TAS permettant de respecter les délais de latence est trouvée,
2. l'agent échoue, ce qui implique que le problème est insoluble ; on considère que c'est le cas lorsque l'agent nécessite plus de 100 tentatives ; dans de tels cas, on considère que l'agent a atteint un état terminal (sinon, l'entraînement prendrait un temps excessif), et l'agent reçoit une très mauvaise récompense.

L'environnement est alors réinitialisé. Dans notre cas, le nouvel environnement est une topologie réseau différente. Certains paramètres, comme la taille des paquets ou le délai entre l'envoi de deux paquets, peuvent aussi varier. Dans un cas idéal, la gigue serait également prise en compte (elle doit idéalement tendre vers 0). Cependant, nous considérerons que si toutes les latences du flux critique sont inférieures à son échéance, cela sera suffisant. Dans un contexte industriel, c'est généralement le cas.

Au tout début de l'entraînement et au début de chaque épisode, afin de pouvoir commencer, les variables de la table 7.1 sont initialisées aléatoirement, à l'exception du nombre de stations émettrices/réceptrices et des contraintes, qui sont pour leur part initialisées avec des valeurs très élevées qui ne risquent pas d'être atteintes.

Action

L'action consiste à modifier la valeur du cycle du TAS, comme expliqué dans la Sous-Section7.1.3. On considère que ne pas le modifier revient à additionner (ou soustraire) 0 à la durée du cycle. Dans l'absolu, on pourrait additionner ou soustraire n'importe quelle valeur dans $\mathbb{R}$. Nous allons cependant nous limiter à toutes les valeurs entières incluses dans $[-100\,000;100\,000]$. On rappelle que la durée du cycle est en nanosecondes, ce qui justifie les valeurs entières.

Récompense

Il y a deux critères particulièrement importants à respecter dans un réseau déterministe, liés à la qualité de service pour chaque flux critique :

1. le délai de bout en bout (latence) maximum ne doit pas être dépassé,
2. la variation de délai de bout en bout entre des paquets d'un même flux (gigue) doit tendre vers 0.

C'est sur cette base que sera déterminée la récompense de l'agent. De plus, si la taille maximum des files d'attentes est atteinte, les nouveaux paquets sont jetés. On souhaite que cela n'arrive pas. Par conséquent, on considère aussi que 95% des paquets du flux critique doivent arriver à destination, ainsi que 50% des paquets des autres flux, pourcentages fixés arbitrairement. Idéalement ces pourcentages devraient être à 100% tous les deux. Ils sont abaissés afin d'accélérer l'entraînement.

La manière dont est attribuée la récompense est simple :

— l'épisode est terminé avec succès : la récompense est de 1,
— la latence a baissé depuis l'épisode précédent et suffisamment de paquet sont arrivés : la récompense est de 0,
— certains éléments (latences, pertes ou gigue) se sont améliorés et d'autres, dégradés : la récompense est de −1,
— la tendance est à la dégradation des performances : la récompense est de −10,
— l'épisode est un échec : la récompense est de −100.

7.2.2 Algorithme d'apprentissage

On l'a vu au chapitre 4, les algorithmes tabulaires ne conviennent pas lorsque l'espace des états est grand, ce qui est notre cas : on choisira donc un algorithme approximatif. En outre, nous devons explorer l'espace des états correctement ; sinon, l'agent sera incapable de résoudre toutes les situations rencontrées. Pour toutes ces raisons, l'agent s'appuie sur l'utilisation de l'algorithme d'apprentissage SAC. Cet algorithme encourage l'agent à explorer l'espace des états, en utilisant l'entropie de la politique (c.-à-d., il rend l'agent très imprévisible pendant les premières phases de la formation).

7.3 Implémentation

La mise en application de la solution proposée précédemment est illustrée sur la figure 7.2. La boucle d'entraînement comprend les trois composantes suivantes :

1. un environnement basé sur l'utilisation d'un outil de simulation permettant de modéliser et de simuler des réseaux TSN,
2. un agent RL, qui est une implémentation python de l'algorithme SAC,
3. une interface de l'environnement qui facilite les interactions entre l'agent RL et l'environnement, et définit le MDP.

Les standards TSN n'ont été implémentés dans INET que très récemment. Avant cela, il existait deux extensions : CoRE4INET (Steinbach et al. 2011) et NeSTiNg (Falk et al. 2019). Aussi bien l'un que l'autre ont déjà été utilisés (par exemple pour CoRE4INET voir (Jiang et al. 2018)). Cependant, NeSTiNg a été préféré, car recommandé par le groupe de travail IEEE 802.1 [1].

NeSTiNg est un outil intéressant en cela que la configuration de TSN est facilement modifiable, d'une part, et que les résultats obtenus sont facilement analysables en Python, d'autre part. Ceci permet de gérer l'environnement depuis un script Python qui modifie la configuration, lance la simulation et analyse les résultats. De plus, la combinaison d'OMNeT++ et de NeSTiNg est reconnue lorsqu'il s'agit d'effectuer des simulations de réseaux TSN, en particulier par l'IEEE (voir la page `https://1.ieee802.org/protocol-simulations/`). Le principal inconvénient de NeSTiNg (mais aussi de CoRE4INET) est l'absence de développement actif. La conséquence est que NeSTiNg n'est compatible qu'avec de vieilles versions d'OMNeT++/INET (tout comme

1. Voir `https://1.ieee802.org/protocol-simulations/`

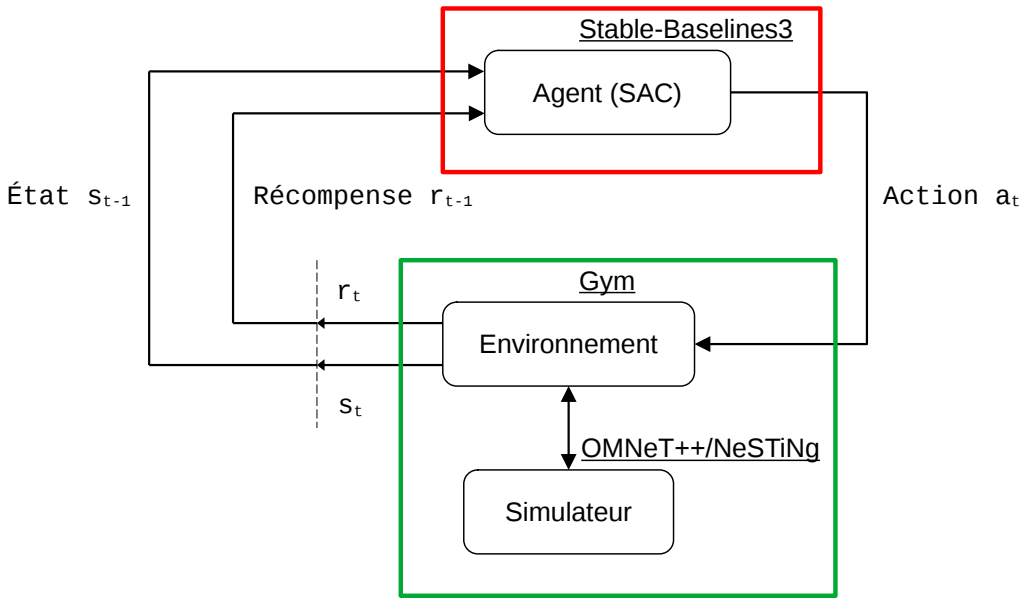


FIGURE 7.2 – La première architecture proposée

CoRE4INET). C'est pourquoi les simulations ont été basculées vers INET dès les modèles TSN intégrés (voir chapitre suivant).

La simulation prend en paramètre un fichier XML qui contient l'ordonnancement, tel que décidé par l'agent. Lorsqu'elle est terminée, la simulation génère des résultats sous format SQLite, qui contiennent entre autres informations les latences de chaque paquet, ce qui permet de calculer la latence maximum et la gigue. Ces deux informations sont celles utilisées par l'agent pour calculer la récompense et le nouvel état.

La simulation permet d'entrainer et d'évaluer l'agent. On simule une durée de 0,1 secondes. La raison est que simuler une durée plus longue rend l'entraînement beaucoup trop long pour une phase de test. Malheureusement, la raison de cette lenteur est liée à la version d'OMNeT++ que l'on est obligé d'utiliser avec NeSTiNg et à NeSTiNg lui-même, et ne peut que difficilement être réduite.

L'agent est connecté à l'environnement via l'Application Programming Interface (API) de OpenAI Gym (BROCKMAN et al. 2016). Cette API permet à l'agent d'être mis en contact avec le simulateur en implémentant un ensemble d'interactions pour la prise de décision et la récupération d'observations.

7.4 Expérience

Le but de cette expérience est de démontrer qu'un agent RL est capable de configurer l'ordonnancement du TAS de manière identique sur chaque commutateur dans un délai raisonnable.

L'agent est évalué de deux manières différentes :

1. pendant l'entraînement : cela permet de suivre les progrès de l'entraînement de l'agent,

2. à la fin de l'entraînement sur un nouvel environnement (c.-à-d., non vu par l'agent durant l'entraînement) pour voir si l'agent est effectivement en mesure d'atteindre ses objectifs.

Pendant l'entraînement, l'agent est évalué à chaque 10 000 pas. En d'autres termes, l'entraînement est arrêté tous les 10 000 pas afin d'évaluer la progression de l'agent. Cette évaluation consiste à exécuter l'agent sur dix épisodes aléatoires (c.-à-d., dix problèmes différents) et à évaluer deux métriques :

1. La durée moyenne des épisodes sur les dix épisodes. Elle représente le nombre d'essais (en moyenne) dont l'agent a besoin avant de trouver une solution. La durée moyenne des épisodes est calculée de la manière suivante :

$$\frac{\text{nombre total d'essais durant l'évaluation}}{10}$$

2. La récompense moyenne sur les dix épisodes. Cette mesure est utilisée pour montrer la progression de l'agent, car sa récompense est censée s'améliorer pendant l'entraînement. La moyenne est calculée de la manière suivante :

$$\frac{\text{somme des récompenses gagnées durant l'évaluation}}{10}$$

Au total, cinq évaluations ont été réalisées sur l'ensemble de l'entraînement. La Figure 7.3 montre la longueur moyenne des épisodes à chaque évaluation. On note que le nombre de pas de temps nécessaires à l'agent pour trouver la configuration adéquate diminue avec le temps. Par exemple, dans la 3e évaluation, l'agent avait besoin en moyenne de 27 essais avant de trouver une bonne configuration alors que dans la 5e évaluation, l'agent avait besoin de 24 essais en moyenne. Ceci montre une progression d'apprentissage de l'agent.

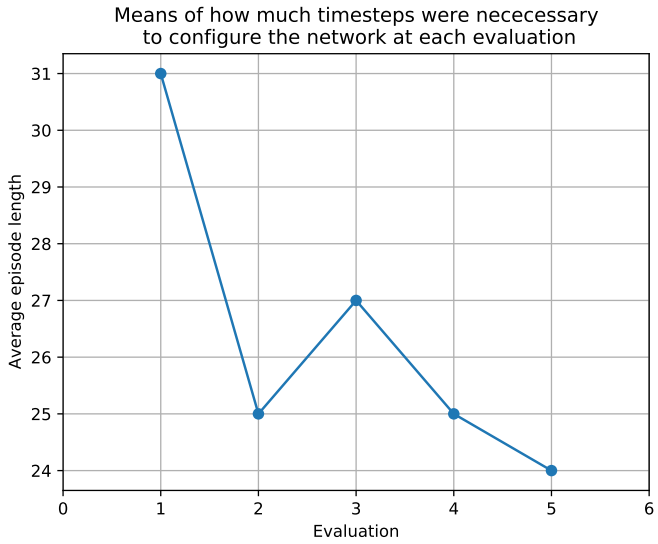


FIGURE 7.3 – Moyenne des nombres de pas nécessaires pour configurer à chaque évaluation

De la même manière, la Figure 7.4 montre la récompense moyenne que l'agent a gagnée à chaque évaluation. On peut voir que la moyenne des récompenses obtenues par l'agent s'améliore

à chaque itération. Dans la 3ème évaluation, l'agent a gagné une récompense moyenne de −23 alors que dans la 5ème évaluation, l'agent a gagné une récompense moyenne de −20. Cela montre que l'agent reçoit moins de récompenses négatives au fil du temps et fait donc moins de mauvaises actions ou trouve une solution plus rapidement.

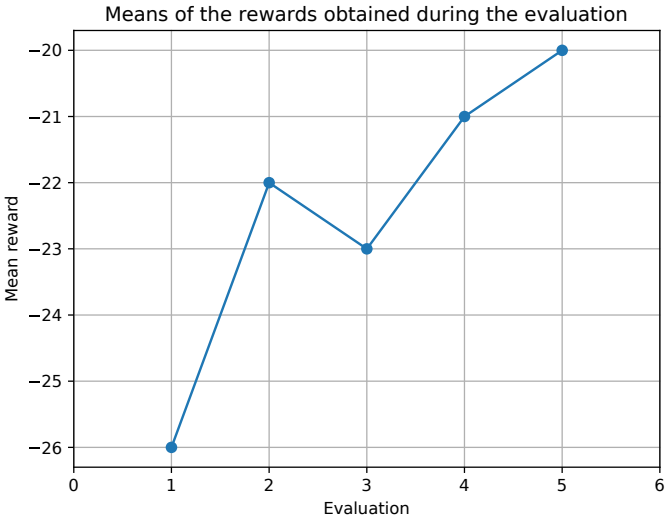


FIGURE 7.4 – Moyenne des récompenses obtenues à chaque évaluation

Ces deux figures montrent que l'agent s'améliore pendant l'entraînement, car il devient capable de trouver une meilleure solution plus rapidement.

À la fin de l'entraînement (c'est-à-dire après 50 000 pas de temps), l'agent est exécuté sur un environnement de test. Ce dernier doit être différent de l'environnement vu par l'agent pendant l'entraînement. Le Tableau 7.2 fournit les paramètres de l'environnement de test.

TABLEAU 7.2 – Paramètres de l'environnement de test

Paramètre	Valeur
Nombre de commutateurs	5
Capacité des liens	1 Gbps
Taille des données utiles critiques	1 000 octet
Taille des données utiles BE	500 octet
Intervalle d'envoi de paquets critiques	500 μs
Intervalle d'envoi de paquets BE	200 μs
Délai à respecter pour les flux critiques	2.5 ms

À travers cette évaluation, on cherche à vérifier si l'agent conçu est capable de trouver un bon ordonnancement pour le TAS qui permet de respecter le délai imposé par le trafic critique. L'évaluation est fondée sur deux observations :

1. si l'échéance du flux critique a été respectée (c.-à-d., que la latence de bout en bout de chaque paquet critique doit être inférieure à l'échéance),

2. combien de temps l'agent a eu besoin pour établir une bonne configuration (c.-à-d., le temps de calcul).

L'agent a été testé sur un ordinateur portable avec un processeur Intel®Core™i5-8265U fonctionnant à 1,60 GHz avec 15,5 Gio RAM. Pour ce test, on simule 10 secondes. Une plus longue durée simulée permet de valider l'hypothèse selon laquelle il n'y a pas une grosse différence entre simuler 0,1 et 10 secondes, en particulier en termes de remplissage des files d'attentes.

L'ordonnancement décidé par l'agent est donné dans le Tableau 7.3. On note que l'agent propose une longueur de cycle de 490 000 nanosecondes. Les portes sont ouvertes pour TSN pour 16 000 nanosecondes, puis les portes pour tout autre trafic sont ouvertes pendant le reste du temps.

Tableau 7.3 – Ordonnancement décidé par l'agent

Paramètre	Valeur
Durée du cycle	490 000 ns
Durée de la séquence du trafic critique	16 000 ns
Durée de la séquence du trafic BE	474 000 ns

La Figure 7.5 montre la latence de bout en bout mesurée pour le trafic critique pendant les premières 1000 ms de la simulation. On note que l'ordonnancement décidé par l'agent permet de respecter le délai imposé pour le trafic critique.

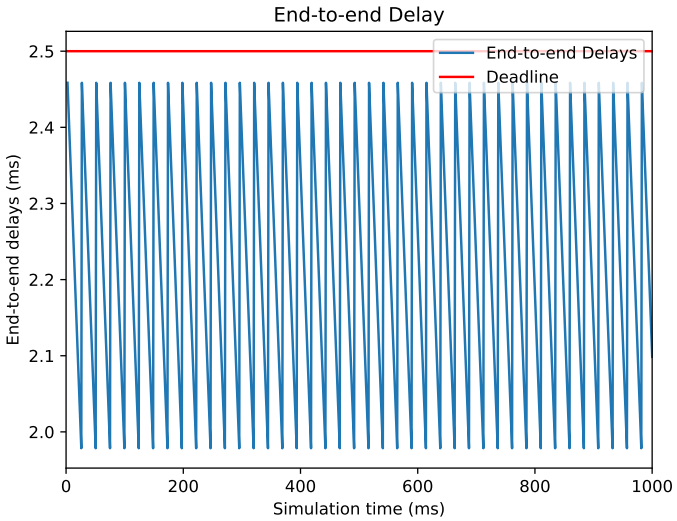


Figure 7.5 – Délais de bout en bout du flux critique pendant les premières 1 000 ms de la simulation

La Figure 7.6 montre la variation du délai de paquets (c.-à-d., la gigue) pour le trafic critique pendant les premières 1000ms de la simulation. On remarque qu'un nombre de trames critiques subissent une certaine gigue (moins de 0,5 ms cependant). En fait, l'agent cherche une configura-

tion acceptable et prend la première qu'il trouve. Il ne cherche pas la configuration optimale. Cette gigue est toujours acceptable, car le délai limite du trafic critique est respecté.

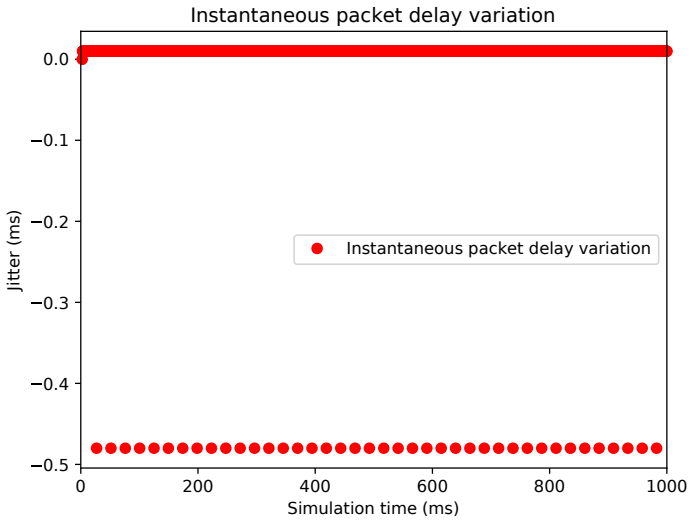


FIGURE 7.6 – Variation instantanée du délai des paquets du flux critique pendant les premières 1 000 ms de la simulation

Afin d'évaluer le temps nécessaire à l'agent pour sortir avec une bonne configuration Qbv, on mesure le temps d'initiation (c.-à-d., le temps de démarrage de l'agent) et le temps de fin (c.-à-d., le temps où l'agent a trouvé la bonne configuration du TAS et l'a validée sur l'environnement de test). La différence entre ces deux paramètres permettra d'avoir une idée du temps de calcul nécessaire à l'agent. L'agent a eu besoin d'environ 34 secondes pour décider la configuration du TAS, la tester et la valider dans l'environnement de test. Cependant, la partie principale de ce temps d'exécution est liée au temps nécessaire pour exécuter la simulation de test, qui dure approximativement 33 secondes. De fait, OMNeT++/INET et NeSTiNg fournissent une modélisation de réseau détaillée et précise qui conduit à un temps d'exécution élevé.

7.5 Conclusion

L'expérience menée dans ce chapitre indique que l'agent conçu peut présenter une configuration acceptable, soulignant le potentiel du RL pour gérer l'ordonnancement du TAS dans des réseaux TSN. Néanmoins, cette première série d'expériences n'est pas complètement satisfaisante pour plusieurs raisons. Tout d'abord, d'un point de vue IA, l'agent n'est pas efficace. Cela tient en particulier à la formulation de la récompense. En effet, l'agent fait face à un problème connu dit des « récompenses clairsemées ». Ce problème arrive lorsque la récompense n'est pas suffisamment diversifiée, spécialement lorsque l'espace des états est très grand. Dit autrement, il y a trop d'états qui retournent la même récompense. La conséquence est que l'agent n'est alors pas en mesure d'apprendre quelle séquence a mené au résultat, car il ne peut distinguer ces séquences entre elles. De plus, l'agent possède une certaine connaissance sur les flux, ce que l'on souhaite éviter. Enfin, plusieurs hypothèses, visant à simplifier le problème d'ordonnancement, ont été formulés. Cela a permis d'examiner la capacité de l'agent RL à configurer le TAS dans

des scénarios simples. Cependant, l'hypothèse disant que l'ordonnancement est identique sur chaque commutateur est très forte. Cette hypothèse est levée dans le chapitre suivant.

Bibliographie du présent chapitre

Brockman, Greg et al. (2016). « Openai gym ». In : *arXiv :1606.01540*.

Falk, Jonathan et al. (mars 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.

Jiang, Junhui et al. (août 2018). « A Time-sensitive Networking (TSN) Simulation Model Based on OMNET++ ». In : *2018 IEEE International Conference on Mechatronics and Automation (ICMA)*, p. 643-648. doi : 10.1109/ICMA.2018.8484302.

Steinbach, Till et al. (2011). « An extension of the OMNeT++ INET framework for simulating real-time ethernet with high accuracy ». In : *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, p. 375-382.

Chapitre 8

Configurer le TAS de manière individuelle sur chaque commutateur

Sommaire du présent chapitre

Dans ce chapitre, nous allons lever l'hypothèse faite auparavant selon laquelle les configurations du TAS sont identiques dans chaque commutateur. Nous présentons les différences dans la modélisation par rapport au chapitre précédent, ainsi que dans la formulation du problème en RL, puis nous montrons le résultat de ces changements sur plusieurs expériences un peu plus complexes à chaque itération.

8.1 Modélisation

Comme dans le chapitre précédent, les éléments spécifiques à cette série d'expériences sont décrits dans cette section.

8.1.1 Modélisation du réseau TSN

En RL, l'espace des actions ne peut varier au cours de l'entraînement. Dans cette série d'expériences, le but est de configurer chaque commutateur individuellement. L'espace des actions dépend donc du nombre de commutateurs. Par conséquent, on ne considère un réseau qu'avec un nombre constant de commutateurs. Il convient de noter que le nombre de commutateurs dans une usine n'est généralement pas soumis à des changements fréquents. Ainsi, cette hypothèse est crédible.

8.1.2 Modélisation du trafic

On considère que le trafic critique est périodique et trafic BE sporadique. La période des flux critiques suit une distribution uniforme dans la fourchette de $]0\mu s, 400\mu s[$. À l'inverse, la période des flux BE est supposée suivre une distribution exponentielle avec un paramètre λ de $400\mu s$. Pour entraîner l'agent sur des configurations de réseau variées (c.-à-d., varier le nombre de flux dans le réseau), plusieurs clients distribués sont utilisés qui génèrent du trafic critique et BE. Il n'y a qu'un seul flux critique.

Le flux critique a une échéance (c.-à-d., la latence maximale acceptée de bout en bout) qui doit être respectée. L'objectif de l'agent sera alors de trouver un ordonnancement TAS qui permettra à la latence du flux critique d'être en dessous de cette échéance.

8.1.3 Paramètres du TAS à configurer

Lors de la configuration du TAS sur un seul port de sortie du commutateur TSN, le processus implique de définir les paramètres suivants pour chaque file d'attente :

— d_{cycle} : la durée du cycle du TAS.
— d_{offset} : la durée du décalage depuis le début du cycle jusqu'à l'ouverture de la porte (l'offset) ; correspond à une porte fermée.
— d_{ouvert} : les durées de l'ouverture des portes.

La Figure 8.1 montre graphiquement les paramètres de configuration TAS que l'agent doit configurer pour un port ayant deux files d'attentes. La file d'attente 1 est associée au trafic critique tandis que la file 0 est associée au trafic BE.

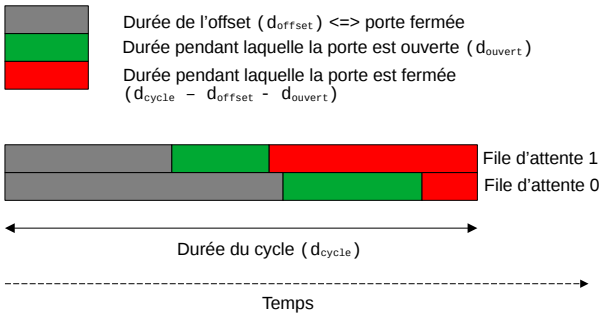


FIGURE 8.1 – Paramètres du TAS à configurer

8.2 Formulation du problème en RL

Dans cette section, le MDP est défini de manière à modéliser correctement un environnement dans lequel les configurations de chaque commutateur sont individuelles.

8.2.1 Définition du MDP

État

L'état du modèle comprend tous les paramètres de configuration TAS (c.-à-d., la durée du cycle TAS, la durée de l'ouverture des portes pour le trafic TSN et pour le trafic BE, le moment où ouvrir les portes dans le cycle) ainsi que le paramètre de latence (l'échéance) du flux critique. L'agent passe d'un état à l'autre en proposant, à chaque fois, une nouvelle configuration TAS pour le réseau qui n'a pas été explorée/testée. L'agent mesure la qualité de chaque transition (c.-à-d., d'un état s_t à s_{t+1}) par une observation de l'environnement et plus précisément de la latence de bout en bout du flux critique.

L'agent peut passer d'un état avec une mauvaise configuration TAS à un état avec une bonne configuration TAS en un seul pas et vice versa. Le défi réside dans la capacité du modèle à apprendre quelle est une bonne configuration pour son environnement. Pour surmonter ce défi, les règles suivantes sont mises en place afin d'aider l'agent :

- La latence mesurée de bout en bout du flux TSN à l'état actuel est inférieure à l'échéance (c.-à-d., la latence maximale de bout en bout acceptée par le flux TSN), la récompense est donc positive et l'épisode s'arrête.
- La latence mesurée de bout en bout du flux TSN à l'état actuel est supérieure à l'échéance. L'agent dispose de deux options :
  1. Le nombre de pas est inférieur au nombre d'étapes temporelles maximales configurées par épisode : l'agent tente d'explorer d'autres possibilités.
  2. Le nombre de pas est égal au nombre de pas maximum configuré par épisode (c.-à-d., cela indique la fin de l'épisode) : une récompense négative est donnée, l'agent met à jour l'état et passe à un nouvel épisode.

Ce faisant, on donne à l'agent la possibilité d'explorer un ensemble de configurations dans chaque épisode. Ceux-ci sont délibérément courts pour accélérer l'apprentissage. L'agent est mis à jour régulièrement jusqu'à ce qu'un sous-espace de configurations soit découvert qui permette à l'agent de converger.

Action

Les actions consistent à modifier les paramètres de configuration du TAS, comme expliqué à la Sous-Section8.1.3. À la différence de l'expérience précédente, où l'agent ajoutait ou retranchait une valeur, cette fois, l'agent choisit directement chaque valeur. L'espace des actions inclut, pour chaque commutateur dans le réseau, cinq actions :

1. déterminer la durée du cycle,
2. déterminer la durée de l'offset pour le trafic critique,
3. déterminer la durée de l'offset pour le trafic BE,
4. déterminer la durée durant laquelle le trafic critique passe,
5. déterminer la durée durant laquelle le trafic BE passe.

Récompense

La récompense est utilisée pour évaluer le nouvel ordonnancement décidé par l'agent. On utilise la formule suivante :

$$recompense = \frac{echeance - nouvelle\_latence}{10} \tag{8.1}$$

où *nouvelle_latence* représente le délai de bout en bout mesuré à chaque pas du flux critique et *echeance* représente une limite au-delà de laquelle *nouvelle_latence* est considérée comme inacceptable pour le flux critique. Les deux paramètres sont donnés en millisecondes. Tant que la nouvelle latence est inférieure à l'échéance, la récompense est négative. Cela permet d'encourager l'agent à trouver plus rapidement une solution acceptable, puis à améliorer ses décisions. Si la latence obtenue avec la configuration TAS décidée est inférieure à l'échéance, l'épisode prend fin.

8.2.2 Algorithme d'apprentissage

L'agent est le cerveau de notre environnement d'apprentissage. Il est régi par un algorithme d'apprentissage tel que SAC ou PPO, selon le problème (c.-à-d., comment l'environnement est modélisé, car chaque algorithme ne peut pas être appliqué à chaque environnement).

8.3 Implémentation

La configuration de l'entraînement est divisée en trois parties comme montré sur la Figure 8.2 :

1. L'agent RL (régi par l'algorithme PPO ou SAC).
2. Le simulateur de réseau OMNeT++/INET.
3. L'environnement intermédiaire permettant la connexion entre l'agent et le simulateur.

La durée de la simulation est d'une seconde par itération.
L'agent est connecté à l'environnement via l'API Gymnasium (Towers et al. 2023), héritière et mise à jour de Openai Gym.

8.4 Expériences

Six ensembles d'expériences ont été menées, chacun permettant d'affiner un peu plus le modèle.

8.4.1 Affiner l'espace des états

Le premier ensemble d'expériences se concentre sur l'évaluation de la limite de l'espace des actions défini dans l'environnement. La Figure 8.3 représente la courbe d'apprentissage pour les deux algorithmes RL SAC et PPO sur une topologie de réseau linéaire de 5 commutateurs connectant un client à un serveur. On note que le taux de convergence de PPO est meilleur que celui de SAC. Comme on peut le constater, SAC a besoin de 16 000 pas d'entraînement supplémentaires pour atteindre le même taux de convergence que PPO. Cette première expérience avec les algorithmes SAC et PPO, qui ont respectivement un espace des actions continu et discret, montre que, en raison de la précision excessive qu'un espace continu peut apporter,

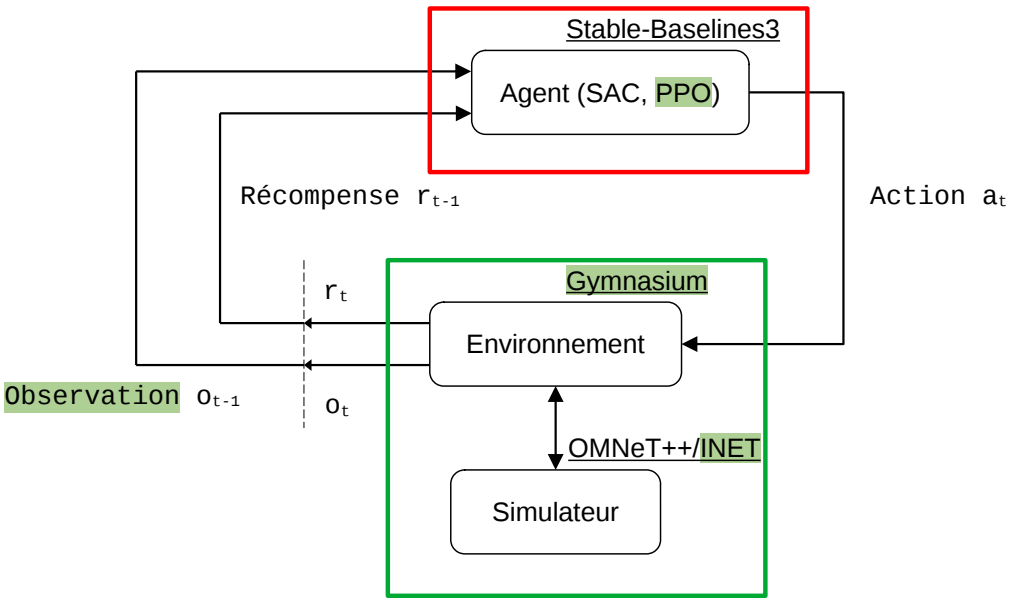


Figure 8.2 – Configuration de l'entraînement

l'apprentissage avec SAC a été extrêmement ralenti. Par conséquent, dans la suite et à des fins expérimentales, la qualité de précision des actions a été réduite en faveur d'un apprentissage plus rapide. La plupart des informations comprises dans l'état étant des durées, cela ne porte pas vraiment à conséquence d'un point de vue logique. Ainsi, pour les expériences suivantes, on s'appuiera sur l'algorithme PPO.

8.4.2 Évaluer la capacité à configurer différentes topologies séparément

La Figure 8.4 montre le taux d'apprentissage de notre agent RL pour les trois types de topologie de réseau (c.-à-d., linéaire, anneau et maillée) avec 5 commutateurs et 3 clients. Nous notons que le taux d'apprentissage de l'agent appliqué à un réseau maillé est meilleur que lorsque la topologie du réseau est linéaire ou circulaire.

Le cas du réseau linéaire présente le pire cas où le flux doit alors rencontrer tous les commutateurs du réseau, se trouvant ainsi considérablement ralenti. Il est intéressant de comparer le taux d'apprentissage de notre agent appliqué à une topologie linéaire avec la présence de trois clients (c.-à-d., un client générant le flux critique et deux clients générant plusieurs flux BE) et le taux d'apprentissage de l'agent appliqué à une topologie linéaire avec un seul client (c.-à-d., un seul client générant le trafic critique et BE en même temps) et en utilisant l'algorithme PPO (ce qui était le cas dans l'expérience précédente, voir Figure 8.3). On note que la convergence de l'agent commence à partir de 5 000 pas si un seul client est considéré, alors que la convergence commence à partir de 17 000 pas avec trois clients. Par conséquent, plus on augmente le nombre de flux dans le réseau et plus on fait varier l'emplacement des clients générant ces flux, plus le taux d'apprentissage de l'agent est lent.

FIGURE 8.3 – Entraînement sur une topologie linéaire avec SAC et PPO

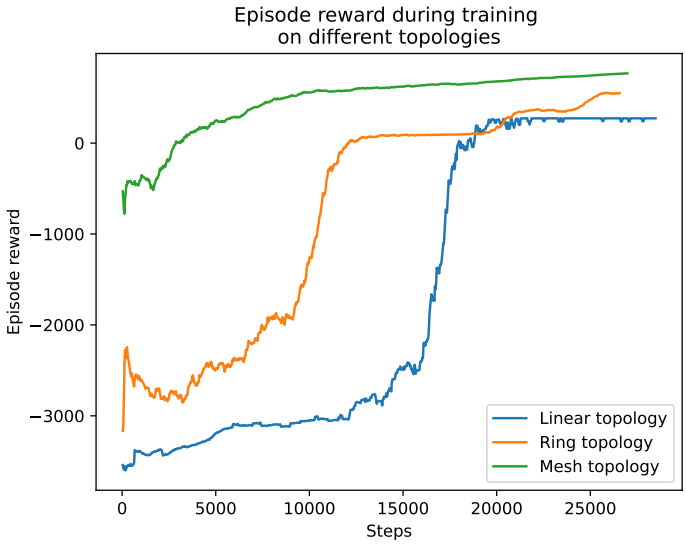


FIGURE 8.4 – Entraînement sur trois types de topologie de réseau (maille, anneau et linéaire)

8.4.3 Évaluer la capacité à s'adapter à un changement applicatif

L'expérience suivante consiste à introduire de la complexité. En effet, cette expérience consiste à faire varier le nombre de clients pendant l'entraînement sur un réseau linéaire classique avec 5 commutateurs. L'objectif est d'observer le comportement du modèle dans le cas d'une variation entre 1 à 5 clients après chaque épisode. Comme on peut le constater sur la Figure 8.5, lors des tests effectués sur l'évaluation du modèle, l'agent parvient à trouver des configurations

acceptables à partir d'environ 30 000 pas d'apprentissage. Cependant, la convergence nécessite encore plus du double du temps.

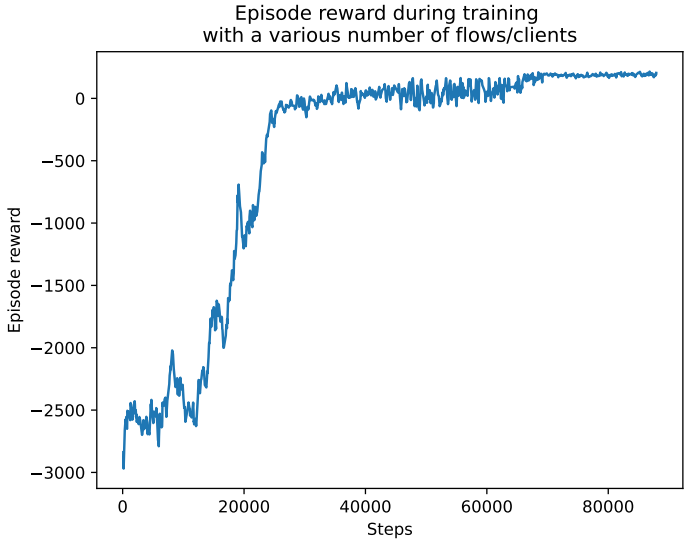


FIGURE 8.5 – Entraînement avec un nombre de clients aléatoire (de 1 à 5) à chaque épisode sur topologie linéaire

8.4.4 Évaluer la capacité à s'adapter à un changement de topologie

Parallèlement à l'expérience sur la variation du nombre de clients pendant l'apprentissage, on essaye de faire varier la topologie avec un nombre fixe de clients (3 clients) afin d'observer la réaction de l'agent à ces changements majeurs (Figure 8.6). La première observation est évidemment l'augmentation drastique du temps d'entraînement par rapport à l'entraînement sur une topologie simple (Figure 8.4). Au-delà d'une augmentation du temps d'apprentissage nécessaire, on constate que le modèle parvient toujours à trouver la convergence vers des configurations intéressantes. Il y a cependant une limite à cette convergence, car elle stagne vers une configuration moyenne adaptée à différentes topologies sans être excellente dans chaque cas. Les nouvelles configurations produites sont bien meilleures qu'une configuration générée aléatoirement, mais elles sont moins efficaces qu'une configuration générée avec un modèle entraîné uniquement sur une topologie dédiée. Bien que cela ait été anticipé, il est toujours intéressant d'observer le temps et les performances maximales qu'une telle configuration peut produire en moyenne à travers une topologie.

8.4.5 Évaluer la capacité à configurer un réseau TSN dynamique

Pour pousser au plus loin la complexité d'un modèle mono-agent, on met en place une expérience représentant la fusion des deux expériences précédentes en modifiant légèrement la récompense afin de viser la minimisation d'une récompense négative. Celle-ci peut prendre des valeurs dans]-∞, 0] et doit tendre vers 0 lorsque la configuration minimise la latence du réseau :

$$recompense = \frac{echeance - nouvelle\_latence}{10} - punition \tag{8.2}$$

FIGURE 8.6 – Entraînement sur une topologie aléatoire (maille, anneau et linéaire) à chaque épisode avec 3 clients

La variable *punition* est importante pour punir l'agent lorsqu'il génère une erreur lors de la simulation en raison de valeurs trop extrêmes qui ont été décidées par l'agent, ou pour accentuer le fait qu'il n'a pas pu décider des configurations TAS permettant d'atteindre une bonne latence. De plus, cela permet de n'avoir que des récompenses négatives, lesquelles tendent vers 0 quand l'agent devient performant.

Le but de cette expérience est d'observer le niveau d'optimisation qu'un modèle peut atteindre avec une configuration sur différentes topologies et avec un nombre variable de clients. Le résultat est présenté sur la Figure 8.7, qui décrit le comportement de l'apprentissage dans un environnement très instable. En dépit d'être plus long à entraîner, on peut remarquer une belle amélioration de la récompense qui converge vers son maximum réalisable comme défini dans la modélisation. Comme on peut le voir, l'apprentissage se fait dans des étapes consécutives de recherche puis de stabilisation jusqu'à trouver un nouvel ensemble de paramètres à étudier plus en détail. On peut identifier une amélioration initiale avec des paramètres choisis plus efficacement que par hasard et ensuite une stabilisation avec une (très) légère amélioration dans le temps. Nous pouvons également observer, à travers la convergence, la robustesse du modèle lorsqu'il fait face à des environnements difficiles comme le nôtre. Cela suggère de nombreuses possibilités concernant sa capacité à apprendre sur des environnements beaucoup plus grands avec notamment beaucoup plus de clients.

Lors de cette dernière expérience, on s'intéresse également à la durée moyenne des épisodes (Figure 8.8). Cela montre de combien d'essais (en moyenne) l'agent a besoin avant de trouver une solution. Nous pouvons voir que le modèle pousse l'agent à trouver plus rapidement une solution acceptable. À la fin de l'entraînement, l'agent a besoin en moyenne d'un seul essai avant de trouver une solution. Si l'on considère qu'une simulation dure environ 20 secondes, cela signifie que l'agent est effectivement capable de trouver rapidement une configuration acceptable.

Une fois l'entraînement terminé, on teste l'agent sur la topologie montré sur la Figure 8.9. Le client 1 envoie un flux critique et un flux BE, les autres clients n'envoient que des flux BE. Les paquets font 1 500 octets. L'agent n'a pas rencontré cette topologie durant l'entraînement : il y a

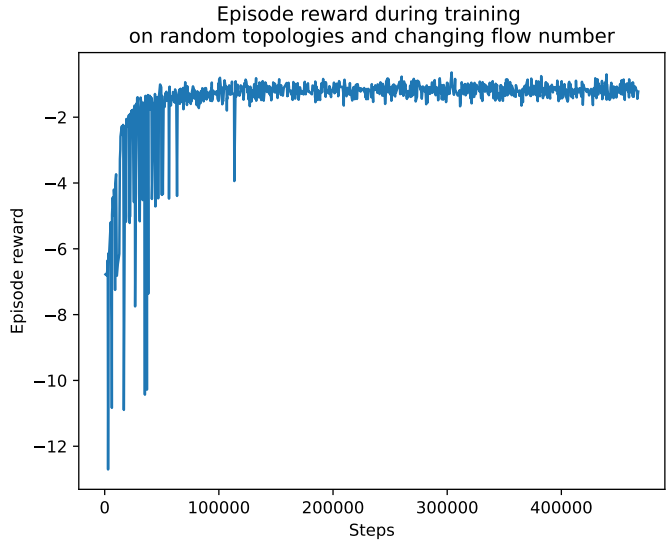


Figure 8.7 – Entraînement sur des topologies aléatoires (maille, anneau et linéaire) et avec un nombre de clients aléatoires (de 1 à 5)

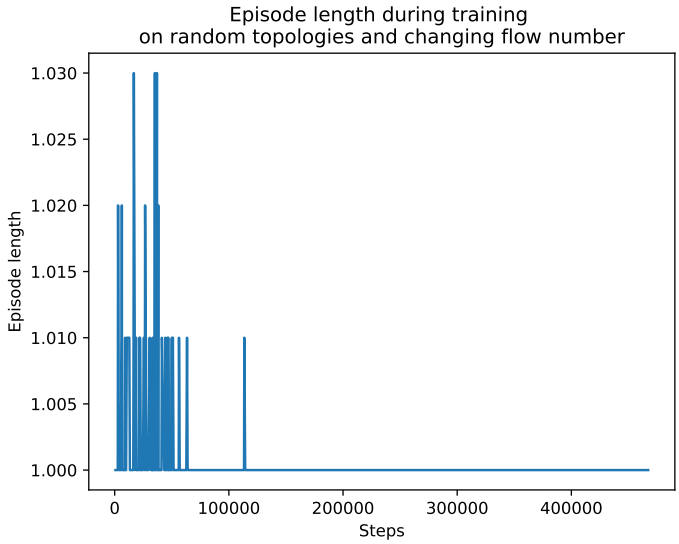


Figure 8.8 – Durée moyenne des épisodes lors du dernier entraînement

en effet un client supplémentaire. Cela permet de vérifier que l'agent est capable de s'adapter à de nouvelles situations.

La Figure 8.10 montre l'ordonnancement décidé par l'agent pour le commutateur relié au serveur. On constate que l'ordonnancement est cohérent (les portes ne sont pas fermées en même temps), même si les flux critiques semblent privilégiés au détriment des autres (la porte de la file d'attente des flux critiques est ouverte pendant beaucoup plus longtemps que celle de la file

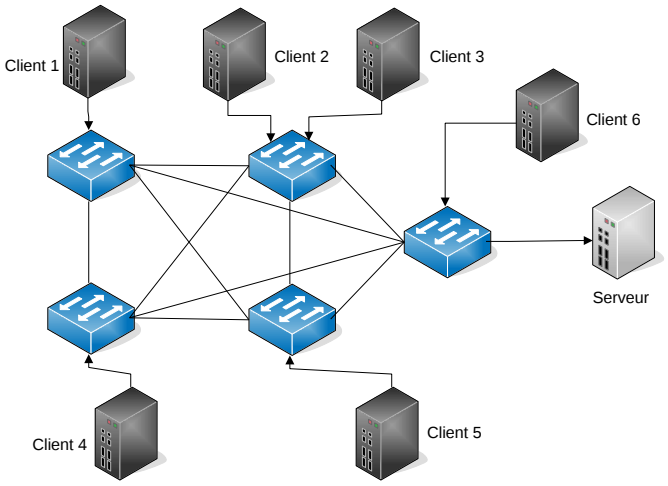


FIGURE 8.9 – Topologie utilisée pour le test

d'attente des flux BE).

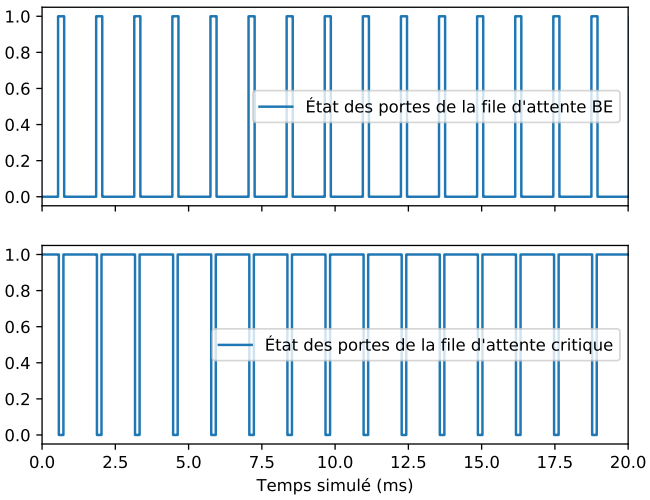


FIGURE 8.10 – Ordonnancement décidé par l'agent

8.5 Conclusion

L'agent décrit dans ce chapitre permet de trouver un ordonnancement acceptable dans un temps très court (moins d'une minute) pour chaque commutateur du réseau. Cet agent n'a pas de connaissances préalables sur les flux, et il est capable de s'adapter à un changement applicatif.

Cependant, cette expérience montre aussi la limite de l'apprentissage, tel que modélisé actuellement, en raison du temps qu'il faut pour atteindre de tels résultats. L'accélération du simulateur par parallélisation n'est pas possible aujourd'hui, mais l'un des axes d'amélioration à travailler pour un modèle plus puissant serait de répartir l'exécution des lots d'entraînement sur plusieurs simulations en parallèle. Malheureusement, plus la modélisation est complexe, plus l'entraînement prend du temps.

Dans le chapitre suivant, nous revenons sur les résultats et donnons les suites possibles à ces travaux.

Bibliographie du présent chapitre

Towers, Mark et al. (mars 2023). *Gymnasium*. DOI : 10.5281/zenodo.8127026. URL : https://zenodo.org/record/8127025 (visité le 08/07/2023).

Conclusion générale et perspectives

Sommaire du présent chapitre

Conclusion

Cette thèse de doctorat a évalué la possibilité de configurer de l'ordonnancement du trafic dans les réseaux TSN à l'aide de l'apprentissage par renforcement. De plus, cette configuration se devait d'être dynamique et surtout, sans connaissance préalable des différents flux. Le dernier critère étant que le processus de décision devait être rapide, de l'ordre de la minute.

La première contribution de cette thèse (Chapitre 7) a été de montrer qu'un agent pouvait, dynamiquement, configurer l'ordonnancement dans un réseau déterministe de manière identique sur chaque commutateur grâce au RL. Il s'agissait principalement de voir ce qu'il était possible de faire en utilisant l'apprentissage par renforcement pour décider un ordonnancement dans le contexte d'un réseau filaire TSN. La clé pour atteindre cet objectif consiste à traduire le problème en un MDP adéquat. La bonne formulation de la récompense, en particulier, est critique. Le problème étant par nature compliqué, des simplifications ont été formulées. Ainsi, l'agent souffre de quelques défauts, par exemple, il possède une connaissance minime des flux. Cette étude a cependant permit de démontrer la pertinence et la faisabilité d'une telle approche et donc, l'intérêt qu'il y avait à poursuivre dans cette voie en complexifiant le modèle.

La seconde contribution (Chapitre 8) permet d'aller plus loin dans cette étude en rendant les configurations sur chaque commutateur indépendantes les unes des autres. Le but principal ici a été d'améliorer l'agent précédent en :

1. supprimant la totalité des connaissances de l'agent sur les flux,
2. ajoutant des topologies différentes (maillée, en anneau),
3. faisant varier le nombre d'applications génératrices de trafic.

Ce qui a conduit à refaire la totalité de la formulation du problème RL. En particulier, la récompense a été reformulée afin de résoudre le problème des trop maigres récompenses rencontré lors de la précédente contribution. De plus, l'agent à désormais un plus grand contrôle sur l'ordonnancement, car il décide directement des valeurs. Cette étude confirme l'intérêt d'utiliser le RL afin de trouver l'ordonnancement du TAS dans les réseaux TSN. En effet, l'agent

de configuration est capable de configurer TSN sur des topologies réseaux différentes, dans un temps raisonnable, sans avoir de connaissances préalables sur les flux et sans être affecté par le nombre ni la position dans la topologie des clients/applications.

En conclusion, cette thèse de doctorat a apporté des contributions significatives à la configuration de l'ordonnancement du trafic dans les réseaux TSN. Les résultats et les pistes d'améliorations présentés ici ouvrent la voie à de futures recherches et à des améliorations dans le domaine de la gestion du trafic dans les réseaux TSN, contribuant ainsi à l'avancement des technologies de communication déterministes pour une variété d'applications industrielles critiques.

Les contributions de cette thèse ont permis de montrer qu'un agent entraîné à l'aide du paradigme RL est capable de décider, dynamiquement, un ordonnancement pour les réseaux TSN dans le cadre du mécanisme TAS, et ce, sans avoir de connaissances préalables sur les flux. Cependant, de nombreux travaux restent à mener. Il existe plusieurs pistes à explorer pour des travaux futurs. La plupart portent sur l'amélioration de l'agent décrit dans le chapitre précédent. Les autres portent sur la manière d'étendre ses capacités à d'autres scénarios.

Améliorer les capacités de l'agent

Tout d'abord, comme on l'a vu au chapitre précédent, l'agent privilégie très clairement les flux critiques au détriment des flux BE. Cela vient de ce que la récompense ne prend en compte que les flux critiques. Ainsi, afin d'y remédier, la récompense devrait inclure aussi bien des informations sur les latences que sur les pertes de paquets. Elle pourrait être de la forme :

$$\text{différence de latence} - \text{paquets critique perdus} - \text{paquets BE perdus}$$

La principale problématique liée à la modélisation de la récompense est que l'on cherche une fonction qui est à valeur strictement négative, et tend vers 0 quand l'agent est entraîné. Plus on rajoute de critères, plus cela est difficile à atteindre, car moins on contrôle les variables.

L'une des principales limites de l'agent est qu'il n'est efficace que sur un nombre de commutateurs fixe. Dit autrement, le cœur du réseau ne peut être dynamique. La cause de ce problème est que l'espace des actions ne peut varier au cours de l'entraînement, il est fonction du nombre d'actions à effectuer sur chaque commutateur et du nombre de commutateurs. Par conséquent, faire varier le nombre de commutateurs revient à faire varier l'espace des actions. Afin de résoudre ce problème, deux solutions peuvent être envisagées.

La plus simple consiste à entraîner l'agent sur un nombre élevé de commutateurs et à « occulter » ceux d'entre eux qui tomberaient en panne. L'idée est que l'agent croit que le commutateur existe toujours, alors que non. L'inconvénient de cette méthode est qu'en plus de perdre en efficacité, on est obligé d'entraîner l'agent sur un nombre élevé de commutateurs, car, si l'on peut en enlever, on ne pourra pas en ajouter.

La seconde solution, plus élégante et compliquée, consiste à utiliser l'apprentissage par renforcement multi-agent (Multi-Agent Reinforcement Learning (MARL)) Busoniu, Babuska et De Schutter 2008. Le MARL est un sous-domaine du RL où plusieurs agents interagissent entre eux et apprennent de leurs actions pour atteindre un objectif commun (coopération) ou opposé (compétition). Il étend le concept de RL traditionnel pour gérer des scénarios complexes impliquant plusieurs entités décisionnelles (Figure 8.11). Pendant l'entraînement, les agents interagissent avec l'environnement et apprennent de leurs propres expériences ainsi que des interactions avec d'autres agents. Grâce à l'apprentissage itératif, ils développent progressivement des stratégies efficaces et des techniques de coordination qui mènent à des résultats positifs. Tout au long du processus d'apprentissage, les agents observent, prennent des mesures, reçoivent

des récompenses et mettent à jour leurs politiques afin d'élaborer des stratégies optimales pour maximiser leur performance collective.

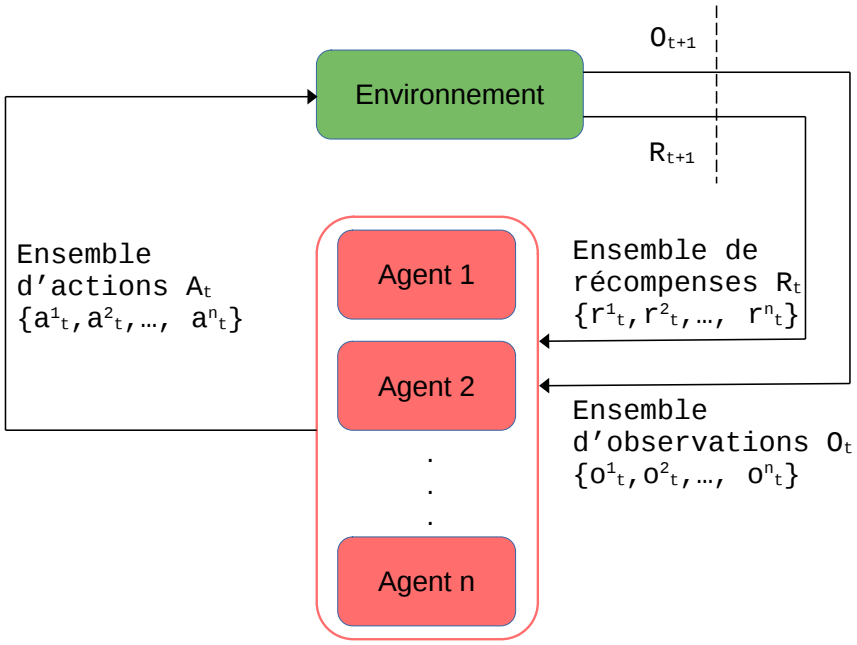


FIGURE 8.11 – Les interactions entre l'agent et l'environnement dans le cadre du MARL

Dans notre cas, l'idée serait de dire que chaque commutateur est représenté par un agent. Leur but serait alors de coopérer afin de trouver leurs ordonnancements respectifs permettant d'atteindre des latences globales basses. Cette solution implique cependant de nombreux changements. Tout d'abord, les algorithmes inclus dans Stable-Baselines3 ne sont plus valides. Basculer l'agent vers du MARL nécessite l'utilisation d'environnements de RL plus complexes. La bibliothèque Ray (MORITZ et al. 2018) et son extension RLlib (LIANG et al. 2018) sont des candidats intéressants. RLlib fournit une API complète de programmation d'algorithmes RL, des outils permettant l'optimisation d'hyperparamètres ainsi que son propre environnement d'apprentissage, ce qui rend la bibliothèque RLlib très complète, mais également très complexe. De plus, l'environnement doit être (au moins partiellement) réécrit. En effet, Gymnasium n'est pas directement adapté au MARL. Trois solutions sont possibles. La première consiste à réécrire l'environnement en utilisant l'API de Ray, la seconde à utiliser PettingZoo (TERRY et al. 2021), une API dédiée au MARL, et la dernière à envelopper l'environnement Gymnasium afin de le rendre compatible.

Une autre problématique à explorer est liée à la définition de la fin d'un épisode. Dit autrement, comment définir une échéance en deçà de laquelle on estime que l'agent a trouvé un bon résultat ? Cette échéance doit répondre à trois critères : elle doit être atteignable et elle devrait varier d'un épisode à un autre afin de modéliser des applications différentes. De plus, elle doit être différente pour chaque flux critique. Pour résoudre ce problème, une solution serait de réécrire la récompense dans le but de se passer d'échéances et de se diriger vers un

problème d'optimisation des latences. Algorithmiquement, cela reviendrait à quelque-chose comme montré dans l'Algorithme 9, le principal problème résidant dans le critère d'arrêt :

Algorithme 9 : Proposition de base pour une récompense

```
1 Enregistrer ancienne_latence;
2 Récupérer nouvelle_latence;
3 tant que nouvelle_latence < ancienne_latence faire
4 |   ancienne_latence = nouvelle_latence;
5 |   récompense = ancienne_latence – nouvelle_latence;
6 fin
```

Enfin, un dernier problème est lié au passage à l'échelle et, encore, à l'espace des actions. En effet, en l'état actuel des choses, l'agent ne cherche que les durées pendant lesquelles les deux portes sont ouvertes, ce qui donne systématiquement une GCL à deux entrées (une porte ouverte, l'autre fermée, puis l'inverse). Cependant, d'un point de vue théorique, cette GCL pourrait très bien être à trois entrées ou plus, et ce, de manière indépendante sur chaque commutateur. Malheureusement, comme précédemment, le fait que l'espace des actions soit statique empêche de rendre les différentes configurations du TAS complètement indépendantes les unes des autres. Une approche possible pour gérer de telles situations serait d'utiliser des actions basées sur la séquence, où un agent peut sélectionner des actions de longueur variable. Ceci est souvent fait en utilisant des réseaux de neurones récurrents (RNNs) ou d'autres modèles de séquence. L'agent peut générer des actions en produisant une séquence d'actions discrètes ou en prédisant la longueur et les composants de l'action de manière dynamique. Par exemple, dans un scénario dans lequel un agent joue à un jeu de cartes et l'action est de jouer un ensemble de cartes, l'agent peut choisir de jouer une carte ou plusieurs cartes dans une seule action. En considérant une représentation d'action basée sur une séquence, l'agent peut sélectionner des actions de longueur variable. Une autre approche pourrait consister à utiliser des techniques de masquage pour gérer l'espace d'action variable. On peut coder les actions légales à l'aide d'un masque binaire, indiquant quelles actions sont valides à chaque point. Au cours de l'entraînement, on peut masquer les actions non valides pour un état donné, ce qui permet au réseau de se concentrer uniquement sur les actions valides. Enfin, au lieu d'utiliser un seul réseau neuronal, on pourrait concevoir des réseaux distincts pour les différents espaces d'action. Chaque réseau serait chargé de gérer une gamme spécifique d'actions en fonction de l'état de l'environnement. Cette approche offre une plus grande souplesse d'adaptation aux différents espaces d'action. On le voit, il s'agit pour l'heure d'un problème complètement ouvert.

Étendre les capacités de l'agent

D'un point de vue plus réseau, une suite intéressante serait de considérer des réseaux sans-fil. En effet, de nombreux travaux sont en cours pour intégrer TSN dans la 5G (Larrañaga et al. 2020) ou dans la future 6G (Thi et al. 2023). Considérer des réseaux sans-fil aura plusieurs impacts :

1. Il y aura des interférences extérieures sur le réseau.
2. Les délais de transmission et de propagation pourront varier.

Ceci rendra la configuration du TAS un peu plus difficile pour l'agent. Par exemple, l'environnement extérieur devra être pris en compte dans le modèle, ce qui aura pour conséquence de le complexifier.

Une autre perspective intéressante serait de pouvoir configurer plusieurs mécanismes TSN. Par exemple, le bon fonctionnement du TAS reposant sur le bon fonctionnement de gPTP, il serait intéressant que l'agent puisse configurer de bout en bout tous ces mécanismes afin d'assurer le bon fonctionnement du réseau.

Comment utiliser l'agent

L'agent de configuration décrit dans les deux chapitres précédents peut s'insérer au sein d'une architecture basée sur le modèle entièrement centralisé décrit dans le standard IEEE 802.1Q (Sous-section 2.2.2). En combinant ce modèle avec le paradigme SDN, on obtient une architecture permettant la configuration d'un réseau TSN, de manière centralisée et dynamique. Le fonctionnement de cette architecture, dépeint sur la Figure 8.12, est le suivant :

0. L'agent s'entraîne sur des simulations.
1. La CNC détecte ou est notifié d'un changement dans le réseau.
2. Elle demande à l'agent une nouvelle configuration pour le TAS en lui fournissant la nouvelle topologie et les contraintes de temps.
3. L'agent fournit une nouvelle configuration pour le TAS.
4. La CNC demande à un module de validation de valider/vérifier que la configuration proposée est faisable.
5. La configuration est validée et on passe au point suivant ou la configuration n'est pas validée et il faut revenir au point 2.
6. La CNC pousse les configurations dans les commutateurs.

Le module de vérification nécessite quelques explications. Une simulation est une représentation ou une imitation d'un phénomène réel, d'une situation ou d'un processus à l'aide d'un modèle ou d'un système de modèles. Elle permet de reproduire les comportements, les interactions et les résultats possibles de manière virtuelle, en simulant les variables, les paramètres et les conditions de la réalité. Elles induisent par conséquent des approximations. L'agent apprenant sur des simulations, il est possible qu'il y ait des écarts avec la réalité. Appliquer une configuration qui ne serait pas faisable peut avoir des conséquences négatives (dans le cadre de l'industrie, cela peut aller jusqu'à l'arrêt de l'usine). Il peut donc être judicieux d'évaluer cette configuration avant de l'appliquer.

Pour résoudre ce problème, un module de validation des configurations est nécessaire. Afin d'atteindre ce but, ce module utilise des analyses pire-cas des délais de traversée ou du calcul réseau (« Network Calculus » en anglais) (Maile, Hielscher et German 2020 ; Boyer 2021 ; Garreau et al. 2021 ; Deng et al. 2022). L'analyse pire-cas des délais de traversée cherche à déterminer le délai maximum qu'un paquet de données peut prendre pour traverser l'ensemble du réseau TSN, en tenant compte de tous les facteurs de retard possibles. Ces facteurs peuvent inclure les temps d'accès au canal, les temps de traitement des commutateurs, les temps de transmission des liaisons et les éventuels conflits de ressources. L'objectif de cette analyse est de garantir que les délais de transmission des données dans le réseau demeurent dans les limites acceptables pour les applications en temps réel. Cela permet d'éviter les retards excessifs qui pourraient compromettre les performances du système. L'analyse pire-cas des délais de traversée prend en compte divers paramètres tels que les temps de commutation, les contraintes de bande passante, les temps de transmission des trames, les temps de réinitialisation des commutateurs, etc. Elle combine ces informations pour évaluer le pire scénario possible et s'assurer que les délais restent dans les limites spécifiées. En général, le délai pire-cas calculé est pessimiste en cela qu'il est supérieur au délai pire-cas réel.

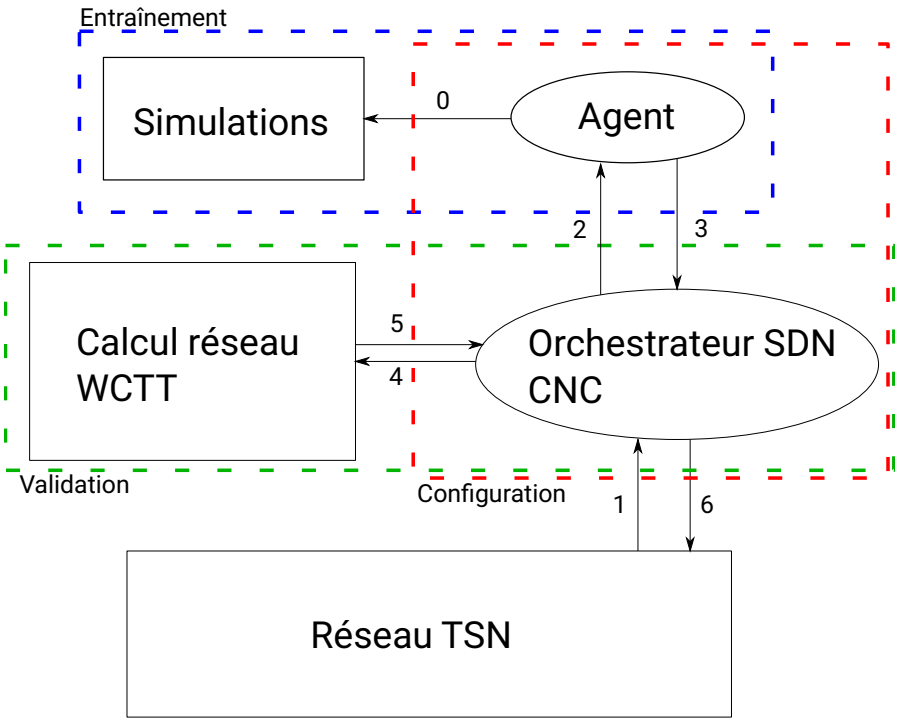


FIGURE 8.12 – Utilisation de l'agent au sein d'une architecture entièrement centralisée

Deux cas sont possibles :

1. Les latences pires cas sont inférieures aux échéances et tout va bien.
2. Les latences pires cas sont supérieures aux échéances.

Dans le second cas, il est possible que le délai pire cas réel soit inférieur à celui calculé (à cause du pessimisme induit par ce type de calcul) et inférieur aux échéances voulues. Malheureusement, on n'a pas la possibilité de distinguer ce cas de celui où la configuration n'est vraiment pas bonne. Par conséquent, dans ce cas comme dans l'autre, la configuration du TAS doit être à nouveau calculée. Afin d'avoir une configuration qui ait plus de chances d'être acceptée, la CNC va demander à l'agent une nouvelle configuration pour le TAS en lui fournissant la même topologie, mais avec des contraintes de temps plus exigeantes.

Bibliographie du présent chapitre

Boyer, Marc (mars 2021). « Garantir les temps de réponse des réseaux embarqués à l'aide du calcul réseau ». Habilitation à diriger des recherches. Institut National Polytechnique de Toulouse. URL : https://hal.science/tel-03220790.

Busoniu, Lucian, Robert Babuska et Bart De Schutter (mars 2008). « A Comprehensive Survey of Multiagent Reinforcement Learning ». In : *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38.2, p. 156-172. ISSN : 1558-2442. DOI : 10.1109/TSMCC.2007.913919.

Deng, Libing et al. (jan. 2022). « A Survey of Real-Time Ethernet Modeling and Design Methodologies : From AVB to TSN ». In : *ACM Comput. Surv.* 55.2. ISSN : 0360-0300. DOI : 10.1145/3487330. URL : https://doi.org/10.1145/3487330.

Garreau, Richard et al. (2021). « Analyse pire-cas des délais de traversée et des tailles de files d'attente dans les réseaux embarqués (Ethernet commuté) ». In.

Larrañaga, Ana et al. (sept. 2020). « Analysis of 5G-TSN Integration to Support Industry 4.0 ». In : *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. T. 1, p. 1111-1114. DOI : 10.1109/ETFA46521.2020.9212141.

Liang, Eric et al. (2018). *RLlib : Abstractions for Distributed Reinforcement Learning*. arXiv : 1712.09381 [cs.AI].

Maile, Lisa, Kai-Steffen Hielscher et Reinhard German (mai 2020). « Network Calculus Results for TSN : An Introduction ». In : *2020 Information Communication Technologies Conference (ICTC)*, p. 131-140. DOI : 10.1109/ICTC49638.2020.9123308.

Moritz, Philipp et al. (2018). *Ray : A Distributed Framework for Emerging AI Applications*. arXiv : 1712.05889 [cs.DC].

Terry, J et al. (2021). « Pettingzoo : Gym for multi-agent reinforcement learning ». In : *Advances in Neural Information Processing Systems* 34, p. 15032-15043.

Thi, Minh-Thuyen et al. (2023). « Enabling Programmable Deterministic Communications in 6G ». In : *Proceedings of the Twenty-Fourth International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*. MobiHoc '23. Washington, DC, USA : Association for Computing Machinery, p. 322-327. ISBN : 9781450399265. DOI : 10.1145/3565287.3617607. URL : https://doi.org/10.1145/3565287.3617607.

Bibliographie

Sommaire du présent chapitre

Livres

Goodfellow, Ian, Yoshua Bengio et Aaron Courville (2016). *Deep Learning*. https://www.deeplearningbook.org. MIT Press.

Sutton, Richard S et Andrew G Barto (2018). *Reinforcement learning : An introduction*. MIT press.

Articles de journaux

Ahmad, Ijaz et al. (2020). « Machine Learning Meets Communication Networks : Current Trends and Future Challenges ». In : *IEEE Access* 8, p. 223418-223460. ISSN : 2169-3536. DOI : 10.1109/ACCESS.2020.3041765.

Arulkumaran, Kai et al. (nov. 2017). « Deep Reinforcement Learning : A Brief Survey ». In : *IEEE Signal Processing Magazine* 34.6, p. 26-38. ISSN : 1558-0792. DOI : 10.1109/MSP.2017.2743240.

Ashjaei, Mohammad et al. (2017). « Schedulability analysis of Ethernet Audio Video Bridging networks with scheduled traffic support ». In : *Real-Time Systems* 53.4, p. 526-577.

Bellman, Richard (1957). « A Markovian decision process ». In : *Journal of mathematics and mechanics*, p. 679-684.

Bezerra, Daniel et al. (2022). « A machine learning-based optimization for end-to-end latency in TSN networks ». In : *Computer Communications* 195, p. 424-440. ISSN : 0140-3664. DOI : 10.1016/j.comcom.2022.09.011. URL : https://www.sciencedirect.com/science/article/pii/S0140366422003504.

Boutaba, Raouf et al. (2018). « A comprehensive survey on machine learning for networking : evolution, applications and research opportunities ». In : *Journal of Internet Services and Applications* 9.1, p. 1-99.

Brockman, Greg et al. (2016). « Openai gym ». In : *arXiv :1606.01540*.

Burns, Alan (1991). « Scheduling hard real-time systems : a review ». In : *Software Engineering Journal* 6.3, p. 116-128.

Busoniu, Lucian, Robert Babuska et Bart De Schutter (mars 2008). « A Comprehensive Survey of Multiagent Reinforcement Learning ». In : *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38.2, p. 156-172. issn : 1558-2442. doi : 10.1109/TSMCC.2007.913919.

Craciunas, Silviu S, R Serna Oliver et T Ag (2017). « An overview of scheduling mechanisms for time-sensitive networks ». In : *Proceedings of the Real-time summer school LÉcole dÉté Temps Réel (ETR)*, p. 1551-3203.

Deng, Libing et al. (jan. 2022). « A Survey of Real-Time Ethernet Modeling and Design Methodologies : From AVB to TSN ». In : *ACM Comput. Surv.* 55.2. issn : 0360-0300. doi : 10.1145/3487330. url : https://doi.org/10.1145/3487330.

Farkas, Janos, Lucia Lo Bello et Craig Gunther (juin 2018). « Time-Sensitive Networking Standards ». In : *IEEE Communications Standards Magazine* 2.2, p. 20-21. issn : 2471-2833. doi : 10.1109/MCOMSTD.2018.8412457.

Finn, Norman (juin 2018). « Introduction to Time-Sensitive Networking ». In : *IEEE Communications Standards Magazine* 2.2, p. 22-28. issn : 2471-2833. doi : 10.1109/MCOMSTD.2018.1700076.

Garreau, Richard et al. (2021). « Analyse pire-cas des délais de traversée et des tailles de files d'attente dans les réseaux embarqués (Ethernet commuté) ». In.

Gärtner, Christoph, Amr Rizk, Boris Koldehofe, René Guillaume et al. (2023). « Fast incremental reconfiguration of dynamic time-sensitive networks at runtime ». In : *Computer Networks* 224, p. 109606. issn : 1389-1286. doi : 10.1016/j.comnet.2023.109606. url : https://www.sciencedirect.com/science/article/pii/S1389128623000518.

Ghotra, Vishavjeet, Max Helm et Benedikt Jaeger (2023). « TSN Qbv and Schedule Generation Approaches ». In : *Network* 29.

Hornik, Kurt (1991). « Approximation capabilities of multilayer feedforward networks ». In : *Neural Networks* 4.2, p. 251-257. issn : 0893-6080. doi : 10.1016/0893-6080(91)90009-T. url : https://www.sciencedirect.com/science/article/pii/089360809190009T.

Hwangbo, Jemin et al. (2019). « Learning agile and dynamic motor skills for legged robots ». In : *Science Robotics* 4.26, eaau5872.

« IEC/IEEE International Standard - Precision Clock Synchronization Protocol for Networked Measurement and Control Systems » (juin 2021). In : *IEC/IEEE 61588-2021*, p. 1-504. doi : 10.1109/IEEESTD.2021.9456762.

« IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks - Amendment 25 : Enhancements for Scheduled Traffic » (mars 2016). In : *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)*, p. 1-57. doi : 10.1109/IEEESTD.2016.8613095.

« IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks » (déc. 2022). In : *IEEE Std 802.1Q-2022 (Revision of IEEE Std 802.1Q-2018)*, p. 1-2163. doi : 10.1109/IEEESTD.2022.10004498.

« IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks – Amendment 31 : Stream Reservation Protocol (SRP) Enhancements and Performance Impro-

vements » (oct. 2018). In : *IEEE Std 802.1Qcc-2018 (Amendment to IEEE Std 802.1Q-2018 as amended by IEEE Std 802.1Qcp-2018)*, p. 1-208. DOI : 10.1109/IEEESTD.2018.8514112.

« IEEE Standard for Local and Metropolitan Area Networks–Timing and Synchronization for Time-Sensitive Applications » (juin 2020). In : *IEEE Std 802.1AS-2020 (Revision of IEEE Std 802.1AS-2011)*, p. 1-421. DOI : 10.1109/IEEESTD.2020.9121845.

Kingma, Diederik P et Jimmy Ba (2014). « Adam : A method for stochastic optimization ». In : *arXiv preprint arXiv :1412.6980*.

Leung, Joseph Y.-T. et Jennifer Whitehead (1982). « On the complexity of fixed-priority scheduling of periodic, real-time tasks ». In : *Performance Evaluation* 2.4, p. 237-250. ISSN : 0166-5316. DOI : 10.1016/0166-5316(82)90024-4. URL : https://www.sciencedirect.com/science/article/pii/0166531682900244.

Li, Yuxi (2018). « Deep reinforcement learning ». In : *arXiv preprint arXiv :1810.06339*.

Lo Bello, Lucia et Wilfried Steiner (juin 2019). « A Perspective on IEEE Time-Sensitive Networking for Industrial Communication and Automation Systems ». In : *Proceedings of the IEEE* 107.6, p. 1094-1120. ISSN : 1558-2256. DOI : 10.1109/JPROC.2019.2905334.

Mammeri, Zoubir (2019). « Reinforcement Learning Based Routing in Networks : Review and Classification of Approaches ». In : *IEEE Access* 7, p. 55916-55950. ISSN : 2169-3536. DOI : 10.1109/ACCESS.2019.2913776.

Messenger, John L. (juin 2018). « Time-Sensitive Networking : An Introduction ». In : *IEEE Communications Standards Magazine* 2.2, p. 29-33. ISSN : 2471-2833. DOI : 10.1109/MCOMSTD.2018.1700047.

Migge, Jörn et al. (2018). « Insights on the Performance and Configuration of AVB and TSN in Automotive Ethernet Networks ». In : *Proc. Embedded Real-Time Software and Systems (ERTS 2018)*.

Min, Junhong et al. (2023). « Reinforcement learning based routing for time-aware shaper scheduling in time-sensitive networks ». In : *Computer Networks* 235, p. 109983. ISSN : 1389-1286. DOI : 10.1016/j.comnet.2023.109983. URL : https://www.sciencedirect.com/science/article/pii/S1389128623004280.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves et al. (2013). « Playing atari with deep reinforcement learning ». In : *arXiv preprint arXiv :1312.5602*.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A Rusu et al. (2015). « Human-level control through deep reinforcement learning ». In : *nature* 518.7540, p. 529-533.

Nasrallah, Ahmed, Akhilesh S. Thyagaturu et al. (2019). « Ultra-Low Latency (ULL) Networks : The IEEE TSN and IETF DetNet Standards and Related 5G ULL Research ». In : *IEEE Communications Surveys & Tutorials* 21.1, p. 88-145. ISSN : 1553-877X. DOI : 10.1109/COMST.2018.2869350.

Pouyanfar, Samira et al. (sept. 2018). « A Survey on Deep Learning : Algorithms, Techniques, and Applications ». In : *ACM Comput. Surv.* 51.5. ISSN : 0360-0300. DOI : 10.1145/3234150. URL : https://doi.org/10.1145/3234150.

Raffin, Antonin et al. (2021). « Stable-Baselines3 : Reliable Reinforcement Learning Implementations ». In : *Journal of Machine Learning Research* 22.268, p. 1-8. URL : http://jmlr.org/papers/v22/20-1364.html.

Schulman, John, Xi Chen et Pieter Abbeel (2017). « Equivalence between policy gradients and soft q-learning ». In : *arXiv preprint arXiv :1704.06440*.

Schulman, John, Filip Wolski et al. (2017). « Proximal policy optimization algorithms ». In : *arXiv preprint arXiv :1707.06347*.

Stüber, Thomas et al. (2023). « A Survey of Scheduling Algorithms for the Time-Aware Shaper in Time-Sensitive Networking (TSN) ». In : *IEEE Access* 11, p. 61192-61233. ISSN : 2169-3536. DOI : 10.1109/ACCESS.2023.3286370.

Terry, J et al. (2021). « Pettingzoo : Gym for multi-agent reinforcement learning ». In : *Advances in Neural Information Processing Systems* 34, p. 15032-15043.

Tesauro, Gerald (jan. 2007). « Reinforcement Learning in Autonomic Computing : A Manifesto and Case Studies ». In : *IEEE Internet Computing* 11.1, p. 22-30. issn : 1941-0131. doi : 10.1109/MIC.2007.21.

Tindell, Ken W, Alan Burns et Andy J. Wellings (1992). « Allocating hard real-time tasks : an NP-hard problem made easy ». In : *Real-Time Systems* 4.2, p. 145-165.

Wang, Mowei et al. (mars 2018). « Machine Learning for Networking : Workflow, Advances and Opportunities ». In : *IEEE Network* 32.2, p. 92-99. issn : 1558-156X. doi : 10.1109/MNET.2017.1700200.

Watkins, Christopher JCH et Peter Dayan (1992). « Q-learning ». In : *Machine learning* 8.3-4, p. 279-292.

Xu, Xun et al. (2021). « Industry 4.0 and Industry 5.0—Inception, conception and perception ». In : *Journal of Manufacturing Systems* 61, p. 530-535. issn : 0278-6125. doi : 10.1016/j.jmsy.2021.10.006. url : https://www.sciencedirect.com/science/article/pii/S0278612521002119.

Xue, Junli et al. (2023). « Scheduling Time-Critical Traffic with Virtual Queues in Software-Defined Time-Sensitive Networking ». In : *IEEE Transactions on Network and Service Management*, p. 1-1. issn : 1932-4537. doi : 10.1109/TNSM.2023.3287634.

Yang, Liu et al. (déc. 2022). « Joint Routing and Scheduling Optimization in Time-Sensitive Networks Using Graph-Convolutional-Network-Based Deep Reinforcement Learning ». In : *IEEE Internet of Things Journal* 9.23, p. 23981-23994. issn : 2327-4662. doi : 10.1109/JIOT.2022.3188826.

Yu, Hao, Tarik Taleb et Jiawei Zhang (août 2023). « Deep Reinforcement Learning-Based Deterministic Routing and Scheduling for Mixed-Criticality Flows ». In : *IEEE Transactions on Industrial Informatics* 19.8, p. 8806-8816. issn : 1941-0050. doi : 10.1109/TII.2022.3222314.

Zhu, Yuan et al. (2023). « Deep Reinforcement Learning-Based Joint Scheduling of 5G and TSN in Industrial Networks ». In : *Electronics* 12.12. issn : 2079-9292. doi : 10.3390/electronics12122686. url : https://www.mdpi.com/2079-9292/12/12/2686.

Articles d'actes de conférences

Atallah, Ayman A., Ghaith Bany Hamad et Otmane Ait Mohamed (juill. 2018). « Fault-Resilient Topology Planning and Traffic Configuration for IEEE 802.1Qbv TSN Networks ». In : *2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS)*, p. 151-156. doi : 10.1109/IOLTS.2018.8474201.

Craciunas, Silviu S. et al. (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*. RTNS '16. Brest, France : Association for Computing Machinery, p. 183-192. isbn : 9781450347877. doi : 10.1145/2997465.2997470. url : https://doi.org/10.1145/2997465.2997470.

Dürr, Frank et 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*. RTNS '16. Brest, France : Association for Computing Machinery, p. 203-212. isbn : 9781450347877. doi : 10.1145/2997465.2997494. url : https://doi.org/10.1145/2997465.2997494.

Falk, Jonathan et al. (mars 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.

Gärtner, Christoph, Amr Rizk, Boris Koldehofe, Rhaban Hark et al. (juin 2021). « Leveraging Flexibility of Time-Sensitive Networks for dynamic Reconfigurability ». In : *2021 IFIP Networking Conference (IFIP Networking)*, p. 1-6. doi : 10.23919/IFIPNetworking52078.2021.9472834.

Haarnoja, Tuomas et al. (2018). « Soft actor-critic : Off-policy maximum entropy deep reinforcement learning with a stochastic actor ». In : *International conference on machine learning*. PMLR, p. 1861-1870.

Henderson, Peter et al. (2018). « Deep reinforcement learning that matters ». In : *Proceedings of the AAAI conference on artificial intelligence*. T. 32. 1.

Jiang, Junhui et al. (août 2018). « A Time-sensitive Networking (TSN) Simulation Model Based on OMNET++ ». In : *2018 IEEE International Conference on Mechatronics and Automation (ICMA)*, p. 643-648. doi : 10.1109/ICMA.2018.8484302.

Larrañaga, Ana et al. (sept. 2020). « Analysis of 5G-TSN Integration to Support Industry 4.0 ». In : *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. T. 1, p. 1111-1114. doi : 10.1109/ETFA46521.2020.9212141.

Mai, Tieu Long, Nicolas Navet et Jörn Migge (mai 2019a). « A Hybrid Machine Learning and Schedulability Analysis Method for the Verification of TSN Networks ». In : *2019 15th IEEE International Workshop on Factory Communication Systems (WFCS)*, p. 1-8. doi : 10.1109/WFCS.2019.8757948.

— (2019b). « On the Use of Supervised Machine Learning for Assessing Schedulability : Application to Ethernet TSN ». In : *Proceedings of the 27th International Conference on Real-Time Networks and Systems*. RTNS '19. Toulouse, France : Association for Computing Machinery, p. 143-153. isbn : 9781450372237. doi : 10.1145/3356401.3356409. url : https://doi.org/10.1145/3356401.3356409.

Maile, Lisa, Kai-Steffen Hielscher et Reinhard German (mai 2020). « Network Calculus Results for TSN : An Introduction ». In : *2020 Information Communication Technologies Conference (ICTC)*, p. 131-140. doi : 10.1109/ICTC49638.2020.9123308.

Mao, Hongzi et al. (2016). « Resource Management with Deep Reinforcement Learning ». In : *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. HotNets '16. Atlanta, GA, USA : Association for Computing Machinery, p. 50-56. isbn : 9781450346610. doi : 10.1145/3005745.3005750. url : https://doi.org/10.1145/3005745.3005750.

Nasrallah, Ahmed, Venkatraman Balasubramanian et al. (déc. 2019). « Reconfiguration Algorithms for High Precision Communications in Time Sensitive Networks ». In : *2019 IEEE Globecom Workshops (GC Wkshps)*, p. 1-6. doi : 10.1109/GCWkshps45667.2019.9024705.

Nayak, Naresh Ganesh, Frank Dürr et 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*. RTNS '16. Brest, France : Association for Computing Machinery, p. 193-202. isbn : 9781450347877. doi : 10.1145/2997465.2997487. url : https://doi.org/10.1145/2997465.2997487.

Pahlevan, Maryam et Roman Obermaisser (sept. 2018). « Genetic Algorithm for Scheduling Time-Triggered Traffic in Time-Sensitive Networks ». In : *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*. T. 1, p. 337-344. doi : 10.1109/ETFA.2018.8502515.

Raagaard, Michael Lander et al. (oct. 2017). « Runtime reconfiguration of time-sensitive networking (TSN) schedules for Fog Computing ». In : *2017 IEEE Fog World Congress (FWC)*, p. 1-6. doi : 10.1109/FWC.2017.8368523.

Samson, Maxime et al. (sept. 2023). « Computing Data Streams in Real-Time Networks from Component-Based Software Engineering ». In : *2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA)*, p. 1-8. doi : 10.1109/ETFA54631.2023.10275469.

Santos, Aellison Cassimiro T. dos, Ben Schneider et Vivek Nigam (oct. 2019). « TSNSCHED : Automated Schedule Generation for Time Sensitive Networking ». In : *2019 Formal Methods in Computer Aided Design (FMCAD)*, p. 69-77. doi : 10.23919/FMCAD.2019.8894249.

Serna Oliver, Ramon, Silviu S. Craciunas et Wilfried Steiner (avr. 2018). « IEEE 802.1Qbv Gate Control List Synthesis Using Array Theory Encoding ». In : *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, p. 13-24. doi : 10.1109/RTAS.2018.00008.

Shih, Yuan-Yao et al. (sept. 2023). « Scheduling of Integrated 5G and Time Sensitive Network for Deterministic Communication ». In : *2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA)*, p. 1-8. doi : 10.1109/ETFA54631.2023.10275610.

Silva, Luis et al. (mai 2019). « On the adequacy of SDN and TSN for Industry 4.0 ». In : *2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC)*, p. 43-51. doi : 10.1109/ISORC.2019.00017.

Steinbach, Till et al. (2011). « An extension of the OMNeT++ INET framework for simulating real-time ethernet with high accuracy ». In : *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, p. 375-382.

Thi, Minh-Thuyen et al. (2023). « Enabling Programmable Deterministic Communications in 6G ». In : *Proceedings of the Twenty-Fourth International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*. MobiHoc '23. Washington, DC, USA : Association for Computing Machinery, p. 322-327. isbn : 9781450399265. doi : 10.1145/3565287.3617607. url : https://doi.org/10.1145/3565287.3617607.

Wang, Xiaolong et al. (avr. 2022). « Deep Reinforcement Learning aided No-wait Flow Scheduling in Time-Sensitive Networks ». In : *2022 IEEE Wireless Communications and Networking Conference (WCNC)*, p. 812-817. doi : 10.1109/WCNC51071.2022.9771665.

Zhou, Boyang et Liang Cheng (2021). « Mitigation of Scheduling Violations in Time-Sensitive Networking Using Deep Deterministic Policy Gradient ». In : *Proceedings of the 4th FlexNets Workshop on Flexible Networks Artificial Intelligence Supported Network Flexibility and Agility*. FlexNets '21. Virtual Event, USA : Association for Computing Machinery, p. 32-37. isbn : 9781450386340. doi : 10.1145/3472735.3473385. url : https://doi.org/10.1145/3472735.3473385.

Thèses

Bailleul, Quentin (nov. 2023). « Dimensioning TSN network synchronization in different embedded contexts ». Theses. Institut National Polytechnique de Toulouse - INPT. url : https://theses.hal.science/tel-04400599.

Boyer, Marc (mars 2021). « Garantir les temps de réponse des réseaux embarqués à l'aide du calcul réseau ». Habilitation à diriger des recherches. Institut National Polytechnique de Toulouse. url : https://hal.science/tel-03220790.

Watkins, Christopher John Cornish Hellaby (1989). « Learning from delayed rewards ». Thèse de doct.

Rapports

Mai, Tieu Long et Nicolas Navet (2020). *Deep Learning to Predict the Feasibility of Priority-Based Ethernet Network Configurations*. Rapp. tech. University of Luxembourg.
— (2021). *Improvements to Deep-Learning-based Feasibility Prediction of Switched Ethernet Network Configurations*. Rapp. tech. University of Luxembourg.
Navet, Nicolas, Tieu Long Mai et Jörn Migge (2019). *Using machine learning to speed up the design space exploration of Ethernet TSN networks*. Rapp. tech. University of Luxembourg.

Autres sources

Irpan, Alex (2018). *Deep Reinforcement Learning Doesn't Work Yet*. https://www.alexirpan.com/2018/02/14/rl-hard.html.
Liang, Eric et al. (2018). *RLlib : Abstractions for Distributed Reinforcement Learning*. arXiv : 1712.09381 [cs.AI].
Moritz, Philipp et al. (2018). *Ray : A Distributed Framework for Emerging AI Applications*. arXiv : 1712.05889 [cs.DC].
Paszke, Adam et al. (2019). « PyTorch : An Imperative Style, High-Performance Deep Learning Library ». In : *Advances in Neural Information Processing Systems 32*. Sous la dir. de H. Wallach et al. Curran Associates, Inc., p. 8024-8035. url : http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.
Raffin, Antonin (2020). *RL Baselines3 Zoo*. https://github.com/DLR-RM/rl-baselines3-zoo.
Towers, Mark et al. (mars 2023). *Gymnasium*. doi : 10.5281/zenodo.8127026. url : https://zenodo.org/record/8127025 (visité le 08/07/2023).

Glossaire

Nombres | A | B | C | D | E | F | H | I | M | O | S | T

Nombres

5G cinquième génération de technologie sans fil, apportant des améliorations significatives en termes de vitesse, de latence et de capacité par rapport aux générations précédentes. 52, 54, 92

A

algorithmes DSE algorithmes utilisés pour explorer l'espace des solutions lors de la conception de systèmes informatiques ou de systèmes embarqués et optimiser plusieurs objectifs (tels que les performances, la consommation d'énergie, la latence, etc.) en prenant en compte les contraintes spécifiques du système. xvii, 51

ANSI organisation américaine qui se consacre à l'élaboration de normes dans divers domaines, tels que la technologie, l'industrie, le commerce, l'environnement et la sécurité dans le but d'améliorer l'efficacité et la compétitivité de l'industrie américaine. xvii, 11

API ensembles de protocoles, de définitions et d'outils logiciels qui permettent aux applications de communiquer entre elles. xvii, 69

ATS défini dans le standard IEEE 802.1Qcr, ce mécanisme priorise et ordonnance le trafic en utilisant la file d'attente par classe et le remodelage par flux; il ne repose pas sur une communication synchrone, offrant ainsi une indépendance par rapport aux mécanismes de synchronisation d'horloge comme gPTP et une utilisation plus élevée des liens que les mécanismes synchrones comme le TAS. xvii

B

Best Master Clock Algorithm algorithme de sélection d'horloge principale parmi plusieurs horloges candidates pour garantir une synchronisation précise et cohérente dans un système distribué. xvii, 20

C

CBS défini dans le standard IEEE 802.1Qav, ce mécanisme utilise des files d'attentes associées à un compteur de crédit; ce compteur accumule des crédits lorsque la file d'attente est inactive et consomme des crédits lorsque des trames sont transmises. xvii, 19

D

DetNet cadre développé par l'Internet Engineering Task Force (IETF) pour fournir des services déterministes au sein des réseaux IP. 53

E

Ethernet technologie de réseau local qui permet la transmission de données entre différents appareils, tels que des ordinateurs, des routeurs et des commutateurs réseau, basée sur un système de communication par câble, où les données sont transmises sous forme de signaux électriques sur des fils de cuivre ou des fibres optiques. 11

F

Fog computing aussi appelée informatique géodistribuée, informatique en brouillard ou encore infonébulisation, modèle de traitement des données décentralisé qui permet de rapprocher les ressources informatiques du bord du réseau, offrant ainsi une meilleure performance, une réduction de la latence et une sécurité accrue des données. 53

H

hyperparamètre variable dont la valeur peut être ajustée par le programmeur afin d'optimiser l'algorithme, par exemple, pour optimiser le temps nécessaire à l'apprentissage. 29, 30, 39

I

industrie 4.0 quatrième révolution industrielle, celle des usines intelligentes, où les machines, les équipements et les systèmes interagissent et se connectent pour optimiser les flux de production, faciliter la prise de décision, réduire les coûts et accélérer le temps de mise sur le marché. 11

Integer Linear Programming méthode d'optimisation mathématique utilisée pour résoudre des problèmes d'optimisation où les variables de décision sont restreintes à prendre des valeurs entières. xvii, 51

M

modèle OSI modèle de référence qui divise le processus de communication en plusieurs couches distinctes, chacune responsable de fonctions spécifiques, facilitant l'interconnexion et l'interopérabilité entre différents systèmes informatiques. xviii, 11

MTU taille maximale d'un paquet pouvant être transmis en une seule fois sans fragmentation. xviii, 62

O

OPC Unified Architecture standard d'échange de données ouvert et indépendant des différentes plateformes et systèmes d'exploitations. xviii, 18

S

Satisfiability Modulo Theories permet de déterminer si une formule mathématique est satisfaisable; généralisation du problème de satisfiabilité booléenne (SAT) à des formules plus complexes impliquant des nombres réels, des entiers et / ou diverses structures de données telles que des listes, des tableaux, des vecteurs de bits et des chaînes. xviii, 51

Software-defined networking modèle d'architecture réseau qui a comme caractéristique de remplacer le matériel spécialisé par des abstractions logicielles en séparant la partie de l'infrastructure de mise en réseau qui décide où l'information est envoyée (plan

de contrôle) de la partie où les données se déplacent réellement (plan de données), et permet la prise de décision dans l'application logicielle. xviii, 19

T

Time-Division Multiple Access méthode de transmission utilisée dans les réseaux de communication pour permettre à plusieurs utilisateurs de partager une même fréquence radio; le temps est divisé en intervalles courts appelés slots et chaque utilisateur se voit attribuer un ou plusieurs slots dans lesquels il peut transmettre ou recevoir des données. xviii, 14

Table des matières

Ordonnancer le trafic dans des réseaux déterministes grâce à l'apprentissage par renforcement

Résumé

L'un des changements les plus perturbateurs apportés par l'industrie 4.0 est la mise en réseau des installations de production. De plus, les discussions portant sur l'Industrie 5.0 montrent la nécessité d'un écosystème industriel intégré, combinant IA et jumeau numérique. Dans cet environnement, les équipements industriels fonctionneront de manière transparente avec les travailleurs humains, nécessitant une latence minimale et une connectivité haut débit pour la surveillance en temps réel. Afin de répondre à ces exigences, l'ensemble de standard Time-Sensitive Networking (TSN) a été introduit. Cependant, la configuration de TSN dans un réseau industriel complexe pose de nouveaux défis. Par exemple, les standards TSN permettent une certaine flexibilité et modularité dans le plan de données, néanmoins, les mécanismes définis dans ces standards dépendent de nombreux paramètres (tels que la topologie du réseau, le routage, etc.) ce qui rend le travail de conception difficile. IEEE 802.1Q est l'un des principaux standards TSN qui fournit plusieurs mécanismes pour atteindre une latence déterministe. L'un d'eux s'appelle le Time-Aware Shaper (TAS). Un commutateur avec une fonction TAS divise le trafic de données, à travers plusieurs priorités, en plusieurs files d'attente organisées selon un ordonnancement régulier. Les principales manières d'organiser ce processus sont basées sur des méthodes exactes ou heuristiques. Ceux-ci sont bons pour les réseaux fermés (lorsque tous les flux sont identifiés à l'avance et que la topologie du réseau est fixe). Cependant, dans un réseau ouvert (où plus de flux sont ajoutés au réseau et la topologie du réseau est dynamique), l'ordonnancement dans TSN peut entraîner des problèmes NP-difficile. L'objectif de cette thèse est de proposer une solution pour traiter l'ordonnancement dans TSN en utilisant l'apprentissage par renforcement profond avec l'utilisation de simulations pour entraîner et évaluer l'agent de configuration.

Mots clés : apprentissage par renforcement (intelligence artificielle), apprentissage profond, industrie 4.0, intelligence artificielle, ordonnancement (informatique), temps réel (informatique), industrie 5.0, time-sensitive networking (tsn), time-aware shaper (tas)

Scheduling the traffic in deterministic networks through reinforcement learning

Abstract

One of the most disruptive changes brought by Industry 4.0 is the networking of production facilities. Furthermore, the discussions on Industry 5.0 show the need for an integrated industrial ecosystem, combining AI and the digital twin. In this environment, industrial equipment will work seamlessly with human workers, requiring minimal latency, high-speed connectivity for real-time monitoring. In order to meet this requirement, the Time-Sensitive Networking (TSN) set of standards was introduced. However, configuring TSN in a complex industrial network poses new challenges. For example, the TSN standard allow some flexibility and modularity in the data plane, however, the mechanisms defined by these standards depends on many parameters (such as network topology, routing, etc.) which makes the design work difficult. IEEE 802.1Q is one of the main TSN standards that provides several mechanisms to achieve deterministic latency. One of them is called Time-Aware Shaper (TAS). A switch with a TAS function divides the data traffic, through multiple priorities, into multiple queues arranged in a regular schedule. The main way to organize this process is based on exact or heuristic methods. These are good for closed networks (when all streams are identified in advance and the network topology is fixed). However, in an open network (where more streams are added to the network and the network topology is dynamic), scheduling in TSN can lead to NP-hard problems. The goal of this thesis is to propose a solution to process the scheduling in TSN using Deep Reinforcement Learning with the use of simulations to train and evaluate the configuration agent.

Keywords: reinforcement learning, deep learning, industry 4.0, artificial intelligence, computer scheduling, real-time data processing, industry 5.0, time-sensitive networking (tsn), time-aware shaper (tas)

Laboratoire des Systèmes Communicants (LSC) – Centre d'intégration Nano-INNOV – 2 Boulevard Thomas Gobert – 91120 Palaiseau – France

Laboratoire d'Informatique et d'Automatique pour les Systèmes (LIAS) – Laboratoire LIAS - ISAE-ENSMA – Téléport 2 - 1 avenue Clément Ader – BP 40109 – 86961 Chasseneuil – France