Apprentissage profond pour le trafic réseau : classification, génération et compression

Fabien Meslet-Millet

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Institut National Polytechnique de Toulouse (Toulouse INP)

Discipline ou spécialité :

Informatique et Télécommunication

Présentée et soutenue par :

M. FABIEN MESLET

le vendredi 10 novembre 2023

Titre :

Apprentissage profond pour le trafic réseau : classification, génération et compression.

Ecoles doctorale :

Mathématiques, Informatique, Télécommunications de Toulouse (MITT)

Unité de recherche :

Institut de Recherche en Informatique de Toulouse (IRIT)

Directeur(s) de Thèse :

M. EMMANUEL CHAPUT

MME SANDRINE MOUYSSET

Rapporteurs :

M. KAMAL SINGH, UNIVERSITE DE SAINT-ETIENNE

MME SELMA BOUMERDASSI, CNAM PARIS

Membre(s) du jury :

MME ANNE FLADENMULLER, UNIVERSITE SORBONNE, Présidente

M. EMMANUEL CHAPUT, TOULOUSE INP, Membre

M. GENTIAN JAKLLARI, TOULOUSE INP, Membre

MME GÉRALDINE TEXIER, IMT ATLANTIQUE, Membre

MME SANDRINE MOUYSSET, UNIVERSITE TOULOUSE 3, Membre

M. RAZVAN STANICA, INSA LYON, Membre

Ne priez pas pour une vie facile, priez pour avoir la force d'endurer la difficulté.
Bruce Lee

Lorsque tout le monde pense pareil, personne ne pense vraiment.
Walter Lippman

Remerciements

La rédaction de cette thèse a été une aventure de vie inoubliable. J'ai exploré de nouveaux horizons, repoussé mes limites et découvert ce que je souhaitais vraiment. Je veux exprimer ma profonde gratitude envers tous ceux qui ont contribué à cette aventure, leur soutien constant et leurs encouragements ont été essentiels à ma réussite.

Je tiens à exprimer ma profonde gratitude envers mon directeur de thèse, le Professeur Emmanuel Chaput. Son expertise, son soutien constant et ses conseils avisés ont été d'une valeur inestimable pour moi. Il a été un mentor bienveillant, qui m'a poussé à repousser mes limites intellectuelles et à viser l'excellence dans mes travaux. Son engagement envers ma réussite et sa passion pour le partage des connaissances m'ont inspiré à donner le meilleur de moi-même.

Je tiens également à exprimer ma profonde gratitude envers ma co-encadrante de thèse, Sandrine Mouysset. Son expertise complémentaire et ses conseils éclairés ont été d'une grande valeur pour mon projet de recherche. Son soutien constant, sa disponibilité et son approche collaborative ont grandement contribué à l'aboutissement de ma thèse. Je suis reconnaissant de pouvoir bénéficier de ses connaissances et de son expérience, ainsi que de son solide encadrement tout au long de ce processus.

Je les remercie tout les deux pour la liberté totale et la confiance qu'ils m'ont accordé dans mes recherches.

Mes remerciements vont ensuite à l'ensemble des membres du jury pour le temps passé à l'analyse de cette thèse et pour avoir été présents à ma soutenance. La qualité et la pertinence de leurs remarques et questions ont démontré leur intérêt pour ce travail et y ont apporté un éclairage fort enrichissant. Rien ne pouvait me faire plus plaisir. En particulier, je remercie Selma Boumerdassi et Kamal Singh d'avoir relu avec attention mon manuscrit. Merci à Géraldine Texier, Razvan Stanica, Anne Flademuller et Gentian Jakllari d'avoir apporté leur expertise à mes travaux. Ce fut un honneur.

Je voudrais remercier mes collègues du laboratoire de l'IRIT à l'ENSEEIHT, que j'ai pu cotoyer, de près ou de loin, au cours de ma thèse : Kevin, Chaima, Romain, Justin, Mohamed, Firmin, Quentin, Antonin, Benoit, Raphael, Lucas, Louis, Evelyne et

toutes/tous celles/ceux que j'oublie. Il y a aussi les doctorants, bientôt docteurs, avec qui j'ai partagé mon bureau Antonin, Benoit, Quentin et Raphael, merci pour votre soutien et bon courage pour la fin.

J'ai une pensée pour Edouard, une étoile partie trop vite, pour m'avoir montré qu'à tout moment un autre chemin était possible. J'ai également une pensé pour Charly, une personne éthique, avec qui j'ai pu avoir des discussions profondes sur certains sujets de la vie.

Merci également aux permanents pour vos conseils : Julien, Riadh, Gentian, Katia, André-Luc, Kathia, Jérome et Béatrice.

Je tiens à remercier les étudiants qui ont contribué à cette thèse au travers du projet long effectué en dernière année qui sont : Gwendhal, Kévin, Charles, Rémi.

Je voudrais remercier particulièrement le secrétariat de l'équipe RMESS : Sylvie Armengaud Metche (plus connue sous le nom de SAM), Vanessa Adjeroud, leur accueil chaleureux et leur aide tout le long de ces années de thèse.

Pour finir, je souhaite remercier mes parents qui ont su me soutenir et m'offrir une vie personnelle, de la maternelle jusqu'à aujourd'hui, aussi épanouie. Je suis également reconnaissant envers eux pour leur soutien moral, leur encouragement et leur compréhension pendant cette période exigeante.

Bien évidemment, merci à ma copine qui m'a supporté à chaque instant tout au long de la thèse, malgré mes sauts d'humeurs râleuses. Je tiens à te dire combien je suis reconnaissant d'avoir une personne aussi merveilleuse et aimante à mes côtés. Tes pancakes, faits avec amour, ont su me remonter le moral dans les moments difficiles.

Jérémy, mon meilleur ami d'enfance et Arnaud, respectivement depuis la CM2 et le lycée jusqu'à aujourd'hui. La thèse est finie, je n'ai plus d'excuses. Merci d'avoir été là malgré ma réactivité.

Enfin, je souhaite exprimer ma gratitude envers l'Institut National Polytechnique pour m'avoir offert cette opportunité et pour m'avoir fourni les ressources nécessaires pour mener à bien ma recherche.

Merci.

Résumé

La thèse proposée se concentre sur l'application de l'apprentissage profond aux infrastructures réseaux des villes intelligentes. Avec l'évolution constante de l'Internet des Objets (IoT), les villes intelligentes ont émergé comme un domaine de recherche important. Les réseaux IoT sont au cœur de ces villes intelligentes, fournissant un moyen de collecter des données à partir d'un grand nombre de capteurs répartis dans la ville. Cependant, les infrastructures de réseau actuelles ne sont pas adaptées à la quantité et à la variété des données générées par l'IoT, ce qui rend l'analyse de ces données difficile. Dans cette thèse, nous explorons trois axes d'amélioration pour les infrastructures réseaux des villes intelligentes : la classification de trafic réseau chiffré, la génération de trafic réseau et la compression d'en-têtes de paquet réseau IoT.

Dans le premier axe, nous proposons une nouvelle architecture d'apprentissage profond, nommée Servername Protocol Packet Network (SPPNet), pour la classification de trafic réseau chiffré. Les données de trafic réseau chiffré sont aujourd'hui massivement utilisées dans les communications au sein des réseaux. La classification de ces données peut améliorer grandement la qualité de service et donc le fonctionnement des réseaux. Après avoir montré les limites des approches actuelles, nous proposons une preuve de concept et nous étudions ses avantages et les limites de cette architecture face au trafic chiffré.

Dans le deuxième axe, nous proposons une architecture d'apprentissage profond, nommée Network Clustering Sequential Traffic Generation (NeCSTGen), pour la génération de trafics réseaux. La génération de trafics est importante pour le dimensionnement des infrastructures réseaux des villes intelligentes. NeCSTGen peut générer du trafic réseau de manière cohérente et réaliste. De plus, contrairement aux approches actuelles, NeCSTGen est adaptatif et peut fonctionner sur tous les types de protocoles.

Dans le troisième axe, nous proposons une architecture d'apprentissage profond, nommée Deep Compression Header (DCH), pour la compression d'en-têtes de paquets réseau IoT. Les en-têtes des paquets réseau IoT sont souvent volumineux par rapport aux données véhiculées. Cette limite entraîne une perte de la bande passante et une augmentation de la consommation de la batterie. De plus, la variété des protocoles actuels

rend l'adaptation des techniques de compression complexe. Pour remédier à ces limites, nous proposons DCH une approche par apprentissage profond permettant la compression universelle d'en-têtes de paquets réseau IoT. DCH a été implémentée sur des équipements IoT pour permettre une compression directement sur ceux-ci.

Les approches proposées ont été évaluées expérimentalement, montrant des performances supérieures à celles des approches actuelles. Ces résultats sont prometteurs pour l'application de l'apprentissage profond aux infrastructures réseaux des villes intelligentes.

Mots clés : Apprentissage profond, Internet des Objets, Trafic réseau, Classification, Génération, Compression

Abstract

The proposed thesis focuses on the application of deep learning to the network infrastructures of smart cities. With the constant evolution of the Internet of Things (IoT), smart cities have emerged as an important research domain. IoT networks are at the heart of these smart cities, providing a means of collecting data from a large number of sensors distributed throughout the city. However, current network infrastructures are not adapted to the quantity and variety of data generated by the IoT, making the analysis of this data difficult. In this thesis, we explore three axes of improvement for the network infrastructures of smart cities : encrypted network traffic classification, network traffic generation, and IoT packet header compression.

In the first axis, we proposed a new deep learning architecture, called Servername Protocol Packet Network (SPPNet), for the classification of encrypted network traffic. Encrypted network traffic data is nowadays massively used in network communications. The classification of this data can greatly improve the quality of service and thus the operation of networks. After showing the limitations of current approaches, we propose a proof of concept and study its advantages and limitations with respect to encrypted traffic.

In the second axis, we proposed a deep learning architecture, called Network Clustering Sequential Traffic Generation (NeCSTGen), for network traffic generation. Realistic network traffic generation is important for sizing network infrastructures for smart cities. NeCSTGen can generate network traffic in a consistent and realistic manner. Moreover, unlike current approaches, NeCSTGen is adaptive and can run on all types of protocols.

In the third axis, we proposed a deep learning architecture, called Deep Compression Header (DCH), for IoT network packet header compression. IoT network packet headers are often large compared to the data carried. This limitation leads to loss of bandwidth and increased battery consumption. In addition, the variety of current protocols makes the adaptation of compression techniques complex. To address these limitations, we propose DCH a deep learning approach for universal IoT network packet header compression. DCH has been implemented on IoT devices to enable compression directly on them.

The proposed approaches have been experimentally evaluated, showing superior performance compared to current approaches. These results are promising for the application of deep learning to smart city network infrastructures.

Keywords : Deep Learning, Internet of Things, Network traffic, Classification, Generation, Compression

Table des matières

Table des figures

Liste des tableaux

Acronymes

6LoWPAN IPv6 over Low-power Wireless Personal Area Networks. , 114, 116, 132, 133, 135, 136

AC Arithmetic Coding. xviii, 115, 116, 117, 118, 119, 120, 122, 123, 124, 128, 132, 134, 136, 149

AE Autoencoders. xv, 14, 15, 16

AES Advanced Encryption Standard. xxiii, 48

ARP Address Resolution Protocol. , 47

CDF Cumulative Distribution Function. xvii, xviii, xxiv, 96, 97, 98, 100, 102, 104, 106

CNAME Canonical NAME. , 43

CNN Convolutional Neural Network. xv, 12, 13, 28, 52, 71

CoAP Constrained Application Protocol. xx, xxi, 109, 123, 127, 132, 133, 135, 146, 156, 157, 196, 198

DCH Deep Compression Header. xix, xx, 9, 109, 110, 117, 118, 121, 122, 124, 125, 126, 129, 132, 133, 134, 135, 136, 137, 146, 147, 149, 154, 155, 156, 157, 158, 159, 160, 161, 163, 165, 166, 167, 168, 169, 171, 195

DES Data Encryption Standard.

DNS Domain Name System. xx, xxi, 43, 47, 71, 134, 146, 196, 198

DPI Deep Packet Inspection. , 28

ESP Encapsulating Security Payload. , 112, 113

FTP File Transfer Protocol. , 26

GAN Generative Adversarial Network. , 68, 70, 71, 72

GMM Gaussian Mixture Model. , 16, 82, 83, 84, 85, 94

GP Gaussian Process. , 5

GradCAM Gradient Class Activation Map. xv, xvi, 20, 38, 39

GRU Gated Recurrent Network. , 17, 18, 38, 42, 82, 83, 124, 125

HTML HyperText Markup Language.

HTTP Hypertext Transfer Protocol. xvi, xvii, xx, xxi, 26, 33, 48, 49, 52, 64, 65, 71, 76, 77, 79, 86, 87, 88, 89, 90, 91, 99, 102, 104, 113, 114, 128, 146, 196, 197, 198

HTTPS Hypertext Transfer Protocol Secure. , 65

IA Intelligence Artificielle. xv, 2, 3, 4, 6, 7, 8, 173

IANA Internet Assigned Numbers Authority. , 28

ICMP Internet Control Message Protocol. , 60, 71

IoT Internet of Things. , 6, 7, 8, 9, 57, 70, 72, 109, 116, 118, 121, 122, 128, 131, 135, 136, 149, 150, 161, 163, 164, 166

IoU Intersection Over Union. , 102

IP Internet Protocol. xvi, xxiii, 27, 28, 37, 38, 39, 40, 41, 42, 43, 46, 60, 62, 65, 67, 70, 77, 79, 110, 112, 113, 132, 135, 146, 147, 149

IPv4 Internet Protocol Version 4. , 113, 122, 146

IPv6 Internet Protocol Version 6. , 109, 113, 114, 116, 123, 132, 134, 135, 146

IPX Internetwork Packet Exchange. , 113

IRC Internet Relay Chat. , 33

IRIT Institut en Recherche Informatique de Toulouse. , 12, 36, 45, 47, 92, 126

KL Kullback-Leibler. , 15, 16

kNN K-Nearest Neighbors. , 29

LDE Logscale Diagram Estimate. xviii, 95, 103, 104, 105

LSTM Long Short Term Memory. , 17, 28, 70, 115

MQTT Message Queuing Telemetry Transport. xx, xxi, 109, 132, 133, 135, 146, 156, 157, 196, 198

NB Naives Bayes. , 29

NCP Network Control Protocol. , 113

NeCSTGen Network Clustering Sequential Traffic Generation. xvi, xvii, xxiv, 8, 9, 57, 58, 73, 74, 75, 78, 81, 82, 92, 93, 94, 95, 101, 102, 104, 106, 171, 172, 173

NIDS Network Intrusion Detection System. , 25, 70, 71, 72

OSI Open Systems Interconnection. , 27, 79, 110

P2P Peer-To-Peer. , 26, 32, 33, 41

POP3S Post Office Protocol over SSL. , 41

QoE Quality of Experience. , 95

QoS Quality of Service. , 25, 27, 95, 171

QUIC Quick UDP Internet Connections. , 34, 43

ResNet Residual Network. xv, 13, 14

RNN Recurrent Neural Network. xv, 16, 17, 18, 47, 57, 83, 84, 87, 90, 94, 115, 124, 125, 169

ROCCO Robust Checksum-Based Compression. , 114

RoHC Robust Header Compression. , 117, 125

RoHCv1 Robust Header Compression Version 1. , 114

RoHCv2 Robust Header Compression Version 2.

RSSI Received Signal Strength Indication. xvii, 100, 101

RTP Real-time Transport Protocol. , 112, 113, 114

RTT Round Trip Time. xviii, 95, 104, 106

SCSH Static Context Header Compression. , 114, 116, 117, 132

SMTP Simple Mail Transfer Protocol. xx, xxi, 33, 77, 79, 90, 97, 102, 104, 134, 146, 196, 198

SMTPS Simple Mail Transfer Protocol Secure.

SNMP Simple Network Management Protocol. xvii, 90, 96, 97, 99

SNR Signal-to-Noise Ratio. xvii, xviii, 57, 72, 100, 101, 102

SPPNet Servername Protocol Packet Network. xvi, xxiii, 8, 26, 36, 41, 42, 43, 45, 47, 52, 171

SSL Secure Socket Layer. , 65

SVM Support Vector Machine. , 5

t-SNE t-distribution Stochastic Neighbor Embedding. , 195

TCN Temporal Convolutional Networks. , 169

TCP Transmission Control Protocol. xvi, xvii, xxiii, 26, 28, 34, 36, 37, 38, 39, 40, 41, 42, 43, 46, 60, 62, 65, 67, 76, 79, 84, 96, 99, 101, 102, 110, 112, 113, 132, 146, 147

TLS Transport Layer Security. xv, xvi, xxiii, 31, 32, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 48

TOR The Onion Router. xxiii, 31, 32, 36, 37, 38, 49

TTL Time to Live. , 132

UDP User Datagram Protocol. xvi, xxiii, 28, 33, 36, 37, 38, 39, 40, 41, 42, 43, 60, 65, 76, 79, 102, 110, 112, 113, 132, 146, 147

VAE Variational Autoencoders. xv, xvii, xxiii, xxiv, 15, 16, 50, 51, 52, 57, 82, 83, 85, 87, 90, 91, 92, 94

VoIP Voice Over IP. xvi, 32, 41, 48, 49, 51, 52, 53, 64, 65

Introduction

Contexte et problématique de la thèse

Les centres urbains ne cessent de croître, aujourd'hui nous observons l'émergence de mégas cités. Les mégas cités sont des villes qui comprennent plus de 10 millions d'habitants, voire 30 millions pour certaines d'entres-elles. Aujourd'hui, elles sont plus de trente dans le monde et devraient être cinquante en 2050 [15]. D'après la Banque Mondiale, d'ici 2050, 7 habitants sur 10 habiteront dans des centres urbains [16], il est donc essentiel de faire face à cette croissance. L'accroissement des centres urbains s'accompagne d'une activité grandissante comme la croissance du trafic. Cela se fait au détriment de l'environnement avec une pollution de l'air et sonore, mais aussi une dégradation de la sécurité. Le citoyen est directement impacté par cette pollution qui a un effet nocif sur sa santé. Il est donc essentiel d'intégrer le développement durable au sein de l'évolution des centres urbains. La technologie et le numérique peuvent être un élément de solution.

Parmi les solutions que peuvent apporter le numérique pour le développement durable des centres urbains, nous trouvons la planification urbaine (pour le transport par exemple) qui s'appuie sur l'utilisation de modèles mathématiques avancés couplés avec de l'analyse de données. L'économie du partage est un autre exemple qui se fonde sur la croissance et l'émergence d'Internet permettant de mettre en relation des personnes dans l'objectif de partager des biens comme des véhicules. Cela permet de réduire l'usage du véhicule dans nos villes et par conséquence la pollution.

Ville intelligente

Présentation

L'introduction du numérique au sein des centres urbains a fait naître le terme de « ville numérique », ou « ville intelligente » (« Smart Cities »). Ce terme souligne un nouveau concept de développement urbain. Il s'agit d'améliorer la qualité de vie des

citadins en rendant la ville plus adaptative et efficace, à l'aide de nouvelles technologies qui s'appuient sur un écosystème d'objets et de services. Le périmètre couvrant ce nouveau mode de gestion des villes inclut notamment : infrastructures publiques (bâtiments, mobiliers urbains, domotique, etc.), réseaux (eau, électricité, gaz, télécoms) ; transports (transports publics, routes et voitures intelligentes, covoiturage, mobilités dites douces - à vélo, à pied, etc.) ; les e-services et e-administrations [17]. Le mot « intelligence » fait référence à l'intelligence des machines, et plus précisément des algorithmes. Les « villes intelligentes » forment donc des systèmes qui sont contrôlés automatiquement, dans lesquels les algorithmes jouent un rôle dans l'automatisation.

Les infrastructures

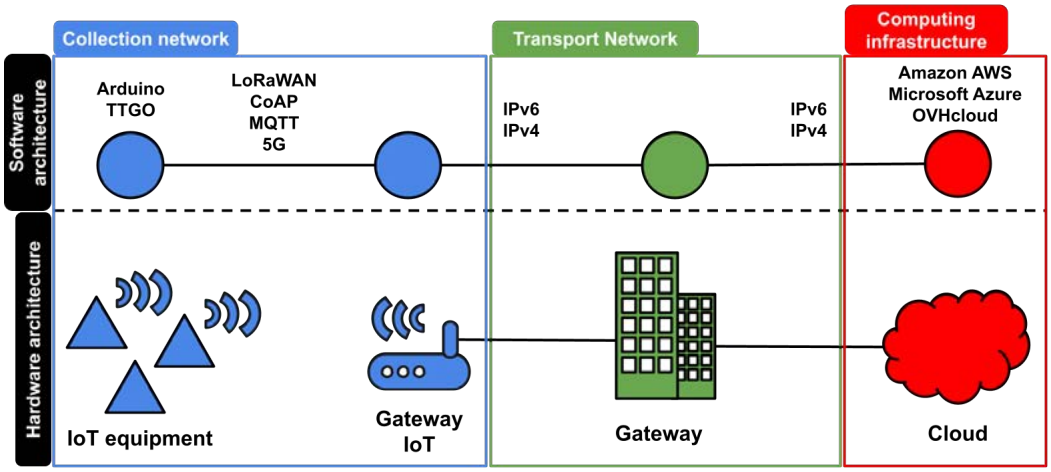

FIGURE 1 Architecture logicielle et matérielle d'une ville intelligente, appelée « Smart Citie ».

La Figure 1 illustre l'architecture logicielle et matérielle d'une ville intelligente. Nous retrouvons les différents éléments liés à l'architecture logicielle et matérielle. Nous pouvons distinguer trois parties : la collecte des données grâce aux capteurs et une ou plusieurs passerelles, l'acheminement des données grâce à une infrastructure réseau et l'exploitation de ces données par des algorithmes le plus souvent mis dans le nuage. La collecte des données est effectuée via des capteurs reliés à une passerelle. Les données collectées sont ensuite acheminées depuis la passerelle au sein d'une infrastructure réseau appartenant à un opérateur. Enfin, les données acheminées sont stockées et exploitées au sein de serveur de stockage et de calcul publique ou privée. C'est au sein de l'infrastructure de calcul que nous retrouvons les outils d'Intelligence Artificielle (IA) qui serviront d'aide à la décision

et permettront de prendre des décisions sur l'environnement.

L'intelligence Artificielle (IA)

Présentation

Face à la quantité importante de données générées les solutions d'algorithmes basées sur IA se sont développées au sein des « Smart Cities » [18]. Ces solutions nécessitent l'ingestion d'une grande volumétrie de données pour pouvoir fonctionner. La volumétrie et la variété des données présentes dans les « Smart Cities » font de l'IA une solution pertinente pour répondre aux problématiques variées qui existent. Nous retrouvons des cas d'application à destination des citadins, mais également des cas d'usages à destination de l'infrastructure réseaux au cœur des « Smart Cities ».

Définition

L'intelligence, d'après le dictionnaire français le Larousse, se définit comme l' « aptitude d'un être humain à s'adapter à une situation, à choisir des moyens d'action en fonction des circonstances. ». L'intelligence est décrite comme une faculté d'adaptation. Dans ce sens général, les animaux, les plantes, les outils informatiques, font preuve d'une intelligence. Donc, l'IA est l'ensemble des théories et des techniques mises en œuvre en vue de réaliser des machines capables de simuler l'intelligence. Elle correspond à un ensemble de concepts et de technologies plus qu'à une discipline autonome constituée. Dans le monde de l'informatique, la signification la plus probante de ce terme vient sans doute de Marvin Lee Minsky, co-fondateur du groupe IA au MIT, la décrivant comme : « La construction de programmes informatiques qui s'adonneraient à des tâches qui sont, pour l'instant accomplies de façon plus satisfaisante par des êtres humains car elles demandent des processus mentaux de haut niveau (...) ». De nombreuses techniques entrent dans ce cadre général, notamment l'apprentissage automatique, l'apprentissage profond, les systèmes-experts et les algorithmes évolutionnaires. La Figure 2 représente l'imbrication de ces différents concepts.

Le « Machine Learning », littéralement « apprentissage automatique » est un champ d'étude qui concerne l'application de méthodes d'optimisation à de la modélisation statistique. Le modèle (ou fonction de prédiction) établi permettra de remplir des tâches difficiles ou problématiques telles que de la modélisation mathématique basée sur les données, de la classification, régression (prédiction), partitionnement/classification ou décision/comportement de manière automatique à partir de données.

Le « Deep Learning » ou « apprentissage profond » est un ensemble de méthodes

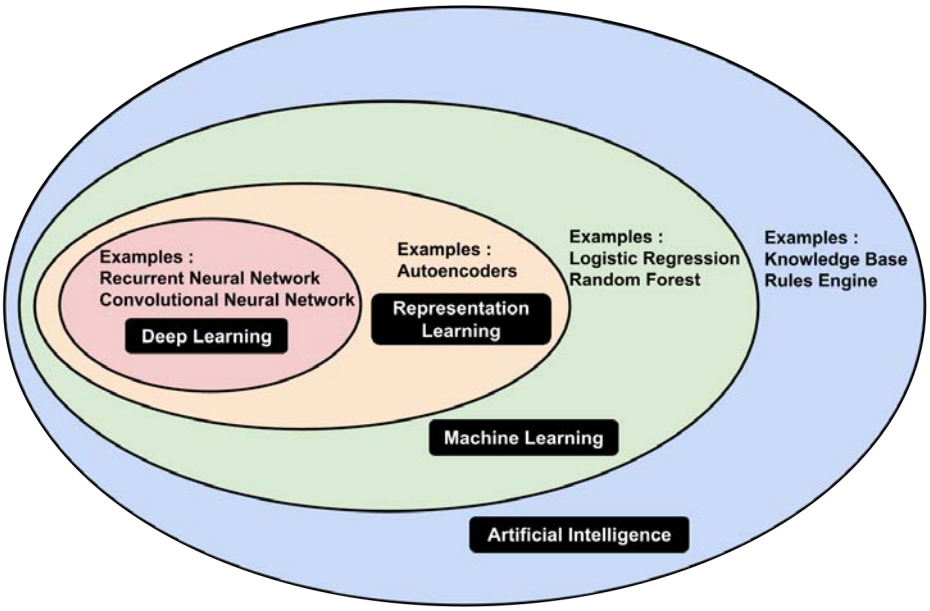

FIGURE 2 Diagramme de Venn, extrait de [1], montrant comment l'apprentissage profond est un type d'apprentissage par représentation, qui est à son tour un type d'apprentissage automatique, utilisé pour de nombreuses approches de l'IA, mais pas toutes. Chaque section du diagramme de Venn comprend un exemple de technologie d'IA.

d'apprentissage automatique tentant de modéliser avec un haut niveau d'abstraction des données. Pour cela, l'apprentissage se fait au travers d'une hiérarchie de représentations de plus en plus abstraites de façon hiérarchique. Le plus souvent des architectures de types « réseaux de neurones » sont utilisées.

Le « Reinforcement Learning » ou « apprentissage par renforcement » a pour but d'apprendre à un système, à partir d'expériences, ce qu'il convient de faire en différentes situations, de façon à optimiser une récompense quantitative au cours du temps. Le système, appelé « agent intelligent », va : observer les effets de ses actions, déduire de ses observations la qualité de ses actions et améliorer ses actions futures. C'est un domaine issu de l'intersection entre l'apprentissage automatique et la robotique.

Avantages de l'apprentissage profond

En reprenant l'analyse effectuée par [2] nous pouvons reconnaître de nombreux bénéfices à l'emploi des méthodes d'apprentissage profond pour l'amélioration du fonctionnement de l'infrastructure « Smart Cities ».

L'étape d'ingénierie des caractéristiques consiste à extraire de nouvelles informations ou à transformer les données existantes en entrée des algorithmes d'apprentissage automatique. Cependant, l'ingénierie des caractéristiques est coûteuse en temps [19]. L'un des avantages de l'apprentissage profond est de pouvoir extraire de façon automatique et hiérarchique les caractéristiques [20]. Dans le cas des réseaux, cela permet de plus facilement travailler avec des données hétérogènes, souvent bruitées et présentant des schémas spatiaux temporels non triviaux [21].

Un autre avantage est la capacité des modèles d'apprentissage profond à traiter un large volume de données. Les approches traditionnelles telles que : Support Vector Machine (SVM) [22] and Gaussian Process (GP) [23] nécessitent de pouvoir sauvegarder les données en mémoire pour pouvoir apprendre. De plus, leurs performances ne croissent pas avec l'augmentation du volume de données [1]. Les approches par apprentissage profond ont, grâce à l'algorithme Stochastic Gradient Descent (SGD) utilisé, la possibilité d'apprendre sur une partie des données de façon itérative et leurs performances continuent de croître avec l'augmentation du volume de données.

Les représentations apprises par les méthodes d'apprentissage profond peuvent être utilisées pour plusieurs tâches. Cela ne peut pas s'effectuer par les méthodes d'apprentissage automatique classiques telles que : régression linéaire, forêt aléatoire, etc. Un seul réseau de neurones peut donc servir à plusieurs objectifs sans avoir besoin d'être re-entraîné pour être appris. Dans un cadre réseau, cela peut permettre de réduire la mémoire et la puissance de calcul utilisée [24].

Les méthodes d'apprentissage profond sont pertinentes pour exploiter les données géométriques [25] très présentes dans le contexte des réseaux. Ces données sont difficilement exploitables par les approches d'apprentissage automatique traditionnelles. Les données géométriques font référence aux données multivariées représentées par des coordonnées, la topologie, la métrique et l'ordre [26]. Les données mobiles, telles que la localisation de l'utilisateur mobile et la connectivité du réseau, peuvent être naturellement représentées par des nuages de points et des graphes, qui présentent d'importantes propriétés géométriques. Parmi les architectures utilisées, nous retrouvons PointNet++ [27] and Graph CNN [28].

Inconvénients de l'apprentissage profond

Néanmoins, les approches par apprentissage profond se définissent comme des « boîtes noires » dont leur capacité d'interprétation semble limitée. Beaucoup de chercheurs reconnaissent et font des investigations afin de remédier à ces limites [29–31].

Les approches par apprentissage profond reposent sur les données. Malgré leur quantité importante disponible au sein des réseaux mobiles, leur collecte peut s'avérer coûteuse.

En effet, nous pouvons faire face à des problèmes de vie privée, mais également par des problèmes de qualité (données trop bruitées, peu adaptées au problème visé, ...). Dans cette situation, le bénéfice de l'apprentissage profond peut être inférieur au coût engendré.

La puissance de calcul nécessaire pour apprendre un modèle par apprentissage profond est souvent importante. Cela nécessite des techniques de parallélisation avancée d'un point de matériel (GPUs, TPUs) et logiciel. Dans un cadre réseau, envisager leur déploiement sur des téléphones mobiles ou sur des capteurs Internet of Things (IoT) peut s'avérer compliqué, voire impossible. Les contraintes en matière de ressources de calcul et d'énergie, surtout des capteurs IoT, sont souvent trop importantes pour permettre le déploiement de solution d'apprentissage profond.

Applications de l'apprentissage profond au sein des réseaux

Les outils d'IA peuvent être utilisés pour une large variété de cadres applicatifs. Dans le cas des « Smart Cities » leur mise en œuvre peut avoir lieu pour une application directe auprès des citadins ou l'amélioration du fonctionnement de l'infrastructure réseau à tous les niveaux. Plus spécifiquement, l'apprentissage profond a permis de nombreuses avancées par rapport aux approches traditionnelles dans le domaine des réseaux. En se fondant sur l'analyse de [2] nous pouvons résumer les différentes avancées de l'apprentissage profond dans les domaines applicatifs des réseaux avec la liste suivante :

1. **L'analyse des données mobiles au niveau du réseau** se concentre sur les données mobiles collectées au sein du réseau, y compris la prédiction du réseau, la classification du trafic et la fouille de données d'enregistrements des appels [32–36].
2. **L'analyse des données mobiles au niveau des applications** s'intéresse à l'analyse des données mobiles sur les appareils périphériques [32, 37–40].
3. **L'analyse de la mobilité des utilisateurs** met en lumière les schémas de déplacement des utilisateurs mobiles, que ce soit au niveau du groupe ou de l'individu [41–47].
4. **La localisation des utilisateurs** dans des environnements intérieurs ou extérieurs, sur la base de différents signaux reçus de dispositifs mobiles ou de canaux sans fil [48–54].
5. **Les réseaux de capteurs sans fil** sont abordés sous quatre angles différents, à savoir la détection centralisée ou décentralisée, l'analyse des données, la localisation et d'autres applications [55–59].
6. **Le contrôle des réseaux** étudie l'utilisation de l'apprentissage par renforcement et par imitation pour l'optimisation des réseaux, le routage, l'ordonnancement,

l'allocation des ressources et le contrôle radio [43, 60–63].

7. **La sécurité des réseaux** s'intéresse à l'amélioration de la sécurité des réseaux. Ces travaux sont regroupés par thème : infrastructure, logiciels et confidentialité [64–68].
8. **Le traitement du signal** étudie qui bénéficient de l'apprentissage profond pour les aspects de la couche physique [69–73].
9. **D'autres applications variées émergentes** dans le domaine des réseaux mobiles tel que, par exemple, la « blockchain » [74–78].

Chacun de ces domaines exploitent une variété d'architectures d'apprentissage profond. Ces architectures sont traverses aux différents domaines applicatifs. Le choix de l'architecture est empirique et nécessite de longues phases de tests pour trouver la bonne configuration. Dans le cadre de cette thèse nous nous intéresserons à deux domaines applicatifs : **L'analyse des données mobiles au niveau du réseau** au travers de la classification et de la génération de trafic réseau et à **d'autres applications variées émergentes** tels que la compression d'en-têtes pour l'IoT.

Déploiements

Généralement, les solutions IA sont déployées sur des infrastructures type « Cloud », là où la puissance de calcul concentrée est la plus importante. Cette approche fait face à deux grandes problématiques. La première est l'acheminement des données. Les données collectées via les capteurs doivent être acheminées jusqu'au centre de données, en entrée de l'algorithme pour pouvoir prendre une décision. Véhiculer une grande volumétrie de données peut entraîner de la congestion réseau et des coûts d'infrastructure pour pouvoir supporter cette charge. La seconde problématique c'est la réactivité. L'action effectuée sur l'environnement ne peut être prise qu'après que l'algorithme est pris sa décision. La réaction n'est pas immédiate et un délai est introduit par l'acheminement des données sur le réseau.

Contributions

Dans nos travaux, nous nous intéressons à l'architecture logicielle et à l'exploitation des données dans l'objectif d'améliorer l'exploitation de l'infrastructure matérielle. Notre objectif est donc de proposer un ensemble de solutions visant à aider la collecte et l'acheminement des données au sein des infrastructures « Smart Cities ». Pour cela, nous

avons pour chaque chapitre abordé une solution technique permettant de remédier à une ou plusieurs problématiques présentes dans ces infrastructures.

Pour commencer, nous nous sommes intéressés à l'acheminement et au traitement des données. Cela nécessite de comprendre les mécanismes de chiffrement, la structure ainsi que la dynamique des données véhiculées. En nous intéressant à la dynamique nous avons constaté que celle-ci variait fortement en fonction du cadre applicatif. Face à cette diversité, nous avons cherché à simuler au mieux la dynamique afin de permettre un dimensionnement plus fin de l'infrastructure matérielle des « Smart Cities ». Le dimensionnement de ces infrastructures passe par des choix protocolaires qui nécessitent de répondre à des contraintes d'énergie, de calcul et de données. Nous avons cherché à remédier à ces limites en proposant un moyen de compression universelle par IA permettant de réduire la consommation d'énergie des équipements IoT en compressant les données véhiculées en utilisant une approche « Fog Computing » [79, 80].

Plan du document

Pour l'ensemble de nos solutions, nous utilisons l'IA et notamment l'apprentissage profond pour la majorité en non supervisé. En effet, la variété et la volumétrie des données nous offrent la possibilité d'utiliser ces méthodes. De plus, elles nous permettent d'offrir un cadre générique qui s'adapte à la variété des données et de leur cadre applicatif. Dans un premier temps, nous présenterons les différentes architectures d'apprentissage profond exploitées au cours de cette thèse. Par la suite, nous aborderons nos contributions, elles peuvent être résumées ainsi :

— **Classification de trafic réseau chiffré « en direct »**. Nous avons développé Servername Protocol Packet Network (SPPNet) une approche modulaire, permettant la classification de flux réseau chiffrée, par apprentissage profond, « en direct » (Chapitre 2). Ce modèle a été implanté sur une machine accompagnée d'une interface Java Script afin d'effectuer de la classification chiffrée dans des contextes variés (section 2.5.5). Dans le même temps, nous avons montré les limites des approches de classification de trafics réseau chiffrés de la littérature actuelle et les traitements à effectuer pour une classification pertinente.

— **Génération de trafic réseau**. Nous avons proposé Network Clustering Sequential Traffic Generation (NeCSTGen) une approche composée d'un ensemble de modèles d'apprentissage profond, permettant la génération de trafic réseau sans a priori sur la structure du trafic à générer (Chapitre 3). Le trafic généré par notre approche peut ensuite être exporté dans un fichier « .pcap » pour effectuer tout type

d'analyses. Nous avons montré l'adaptabilité de NeCSTGen face à différentes sources de trafic provenant de différents réseaux.

— **Compression d'en-tête**. Nous avons développé Deep Compression Header (DCH) une approche, par apprentissage profond, permettant la compression sans perte d'en-têtes de tout type de protocoles réseau sans connaissance de la structure des données à compresser (Chapitre 4). DCH a été implanté sur un capteur IoT sous forme de tables compressées par des techniques que nous avons élaborée (section 4.5). La performance de compression effectuée sur capteurs a également été évaluée.

— **Étude de sensibilité pour la compression d'en-tête**. Nous avons effectué une étude de sensibilité de DCH afin de comprendre l'impact de la structure du protocole à compresser sur les performances de compression (Chapitre 5). Un jeu de données artificiel a été développé afin d'appréhender les différents type de structures pouvant exister au sein des protocoles de communication pour IoT.

Chapitre 1

Architectures d'apprentissage profond

Sommaire

1.1 Introduction

L'objectif de ce chapitre est de présenter les différents types d'architectures par apprentissage profond utilisées dans la suite de cette thèse. Ainsi, nous aborderons les architectures que nous utilisons pour effectuer de la classification, de la génération et de la compression de trafic réseau. La modélisation et l'exploitation de ces architectures d'apprentissage profond sont permises par la plateforme de calcul Osirim [81] de l'Institut en Recherche Informatique de Toulouse (IRIT). De plus, l'ensemble de nos approches et architectures développées sont reproductibles et les liens des dépôts GitHub seront mentionnés au cours des chapitres.

Le « Deep Learning » ou « apprentissage profond » est un ensemble de méthodes d'apprentissage automatique tentant de modéliser avec un haut niveau d'abstraction des données. Une des caractéristiques de l'apprentissage profond réside dans l'utilisation d'architectures variées pour la réalisation de tâches diverses. Ces architectures font la spécificité de l'apprentissage profond et leur conception est un élément important pour leur performance. De plus, les choix effectués durant leur définition dépendent de l'objectif visé. C'est ce qui explique, entre autres, les différences d'architectures qui peuvent exister entre les applications en vision par ordinateur et en traitement du langage naturel.

Dans une première partie, nous présentons les architectures convolutives utilisées pour la classification de trafic réseau. Dans une seconde partie, nous parlerons des autoencodeurs utilisés pour la classification et la génération de trafic réseau. Dans une troisième partie, nous exposons les architectures récurrentes utilisées pour la génération et la compression de trafic réseau. Dans une quatrième partie, nous abordons les paramètres particuliers utilisés pour la configuration de ces architectures. Enfin, en dernière partie, nous synthétisons les outils spécifiques utilisés sur ces architectures.

1.2 Convolutional Neural Network (CNN)

L'architecture Convolutional Neural Network (CNN) (ou ConvNets) se compose d'une succession de couches formant des filtres convolutifs. Ces filtres, appris durant la phase d'apprentissage, permettent de repérer des motifs dans les données en entrée du modèle. Ils permettent l'apprentissage de motifs invariants en translations et, plus généralement, d'être robustes aux transformations affines des objets.

Nous illustrons le fonctionnement d'une couche convolutive à deux dimensions dans la Figure 1.1. $\mathbf{p}_G$ représente toutes les positions dans le champ réceptif G du filtre convolutif W. W représente la plage réceptive de chaque neurone en entrée de la couche convolutive.

Ici, les poids W sont partagés entre les différentes positions des données en entrée.

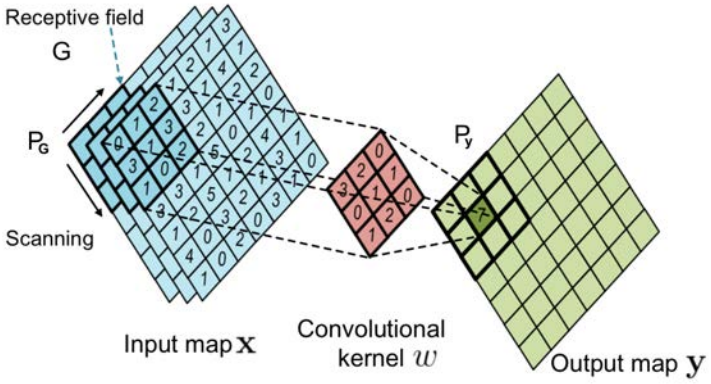

FIGURE 1.1 Fonctionnement d'une couche convolutive à deux dimensions au sein d'une architecture CNN extrait de [2].

L'architecture CNN est majoritairement utilisée en imagerie, mais nous la retrouvons aussi dans le domaine du traitement du langage naturel [82] et des séries temporelles [83]. Compte tenu de la grande similitude entre les images et les données mobiles spatiales (par exemple, les instantanés du trafic mobile, la mobilité des utilisateurs, etc.), les modèles basés sur des CNNs présentent un énorme potentiel pour l'analyse des données mobiles à l'échelle du réseau. Lors du chapitre 2, dans le cadre de la classification de trafic réseau, nous considérons les paquets réseau comme des images. En effet, ils possèdent tous deux des caractéristiques qui peuvent être détectées à l'aide de filtres convolutifs. Un paquet de réseau transportant des données liées à de la navigation web contiendra des termes tels que « <http/> » détectables par des filtres convolutifs.

Residual Network (ResNet) [3] est un exemple d'architecture CNN connue. La particularité de l'architecture ResNet [3] est la présence de connexions dites « résiduelles » entre chaque bloc. La Figure 1.2 représente une connexion résiduelle.

Ces connexions permettent, lors de la rétro-propagation du gradient, d'acheminer l'information jusqu'aux couches les plus hautes. Sans ces connexions, lorsque le nombre de couches du réseau de neurones augmente, l'information n'arrive pas jusqu'au couche les plus hautes. C'est le problème de « évanouissement du gradient ». Ce phénomène s'accroît avec l'augmentation du nombre de couches. Ces connexions ont également un effet régularisateur et permettent d'éviter, en partie, le sur-apprentissage. Sa particularité fait cette architecture une référence parmi les CNN et elle sera utilisée dans la thèse au cours du chapitre 2.

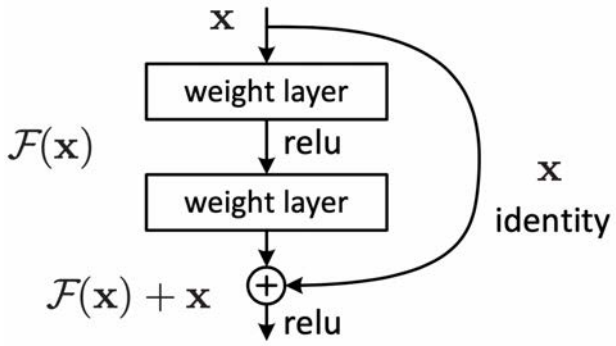

FIGURE 1.2 Représentation d'une connexion résiduelle de l'architecture ResNet [3].

1.3 Autoencodeur (AE)

Le point de vue pris dans cette thèse est de considérer des approches non supervisées. Pour cela, nous utiliserons l'architecture Autoencoders (AE), l'objectif est d'apprendre une représentation compacte des données par la réduction de dimensions [84]. La Figure 1.3 illustre l'architecture d'un AE.

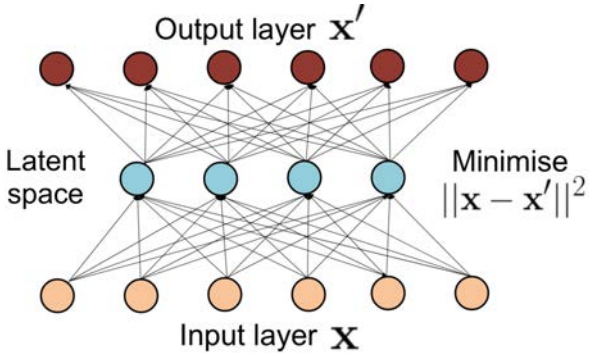

FIGURE 1.3 Architecture d'un AE extrait de [2].

Un AE se compose de deux parties, un « encodeur » et un « décodeur ». Ils sont appris simultanément et peuvent être assimilés à un compresseur et décompresseur.

L'entrée de l'encodeur est un point de données x (un paquet réseau représenté sous la forme d'une image par exemple), son objectif est d'encoder ce point de données dans un espace latent, noté Z avec z un échantillon de l'espace latent. Notons $f_\theta(z|x)$ la représentation de l'encodeur paramétré par θ et z un échantillon d'une distribution apprise par l'encodeur. Le décodeur, noté $g_\omega(x|z)$, paramétrés par ω, prend en entrée

un échantillon de l'espace latent, et donne en sortie x' aussi semblable que possible à x. L'objectif d'un AE est de minimiser l'erreur de reconstruction entre x et x'.

L'AE est uniquement entraîné à coder et décoder avec le moins de pertes possible, quelle que soit la façon dont l'espace latent est organisé. La continuité (deux points proches dans l'espace latent ne doivent pas donner deux contenus complètement différents une fois décodés) et l'exhaustivité (pour une distribution choisie, un point échantillonné dans l'espace latent doit donner un contenu « significatif » une fois décodé) n'est pas garantie dans un AE.

Pour remédier à ces limites le VAE [4] a été introduit. Celui-ci permet de passer par des densités de probabilité pour une génération plus réaliste avec de la variabilité. La Figure 1.4 illustre le fonctionnement d'un Variational Autoencoders (VAE) [4].

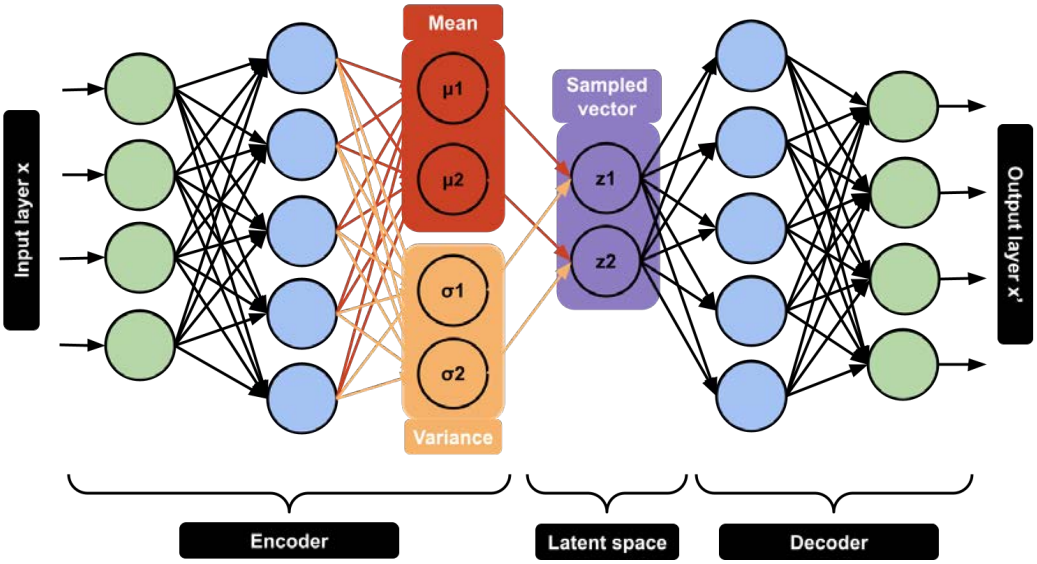

FIGURE 1.4 Architecture d'un VAE [4].

L'objectif d'un VAE [4] est de minimiser l'erreur de reconstruction entre x et x' et la divergence de Kullback-Leibler (KL) entre $\mathcal{N}(0,1)$ et $f_\theta(z|x)$. x peut être un paquet réseau ou un vecteur contenant des statistiques de flux, par exemple. Une fois entraîné, le VAE [4] peut générer de nouveaux échantillons de données en échantillonnant l'espace latent $z_i \sim P(z)$ et en utilisant le décodeur avec $x'_i = g_\omega(x|z_i)$. Le VAE [4] possède une variation dans sa conception appelée β-VAE [10]. Son objectif est de contraindre l'encodeur à projeter les données dans un espace latent dont les facteurs sont indépendants entre eux. Concrètement, la partie de la fonction de coût optimisant la divergence de KL est pondérée à l'aide d'un facteur $\beta > 1$. À l'inverse pondérer la fonction de coût avec

$\beta < 1$ reviendrait à s'approcher d'un AE classique.

Les points de données x similaires seront, le plus souvent, regroupés sous la forme de clusters au sein de l'espace latent du VAE [4]. Par exemple, deux vecteurs de statistiques ayant la même valeur de débit en octets/s. La divergence de KL introduit dans la fonction de coût permettra à ces clusters d'approcher une forme de gaussienne. Ces clusters peuvent ensuite être identifiés à l'aide de l'algorithme Gaussian Mixture Model (GMM) pour pouvoir être échantillonnés. L'algorithme GMM est un modèle probabiliste de mélange gaussien qui suppose que tous les points de données sont générés à partir d'un mélange d'un nombre fini de distributions gaussiennes avec des paramètres inconnus. Ces paramètres sont appris à l'aide de l'algorithme Expectation-Maximization (EM) [85]. En revanche, le nombre de gaussiennes est un paramètre à déterminer par expertise humaine.

L'utilisation du VAE [4] accompagné du GMM sera exploitée durant le chapitre 4 de la thèse pour effectuer de la génération de trafic réseau. Cette architecture nous permettra de synthétiser l'information en regroupant les paquets avec des caractéristiques similaires dans un espace $\mathbb{R}^2$. Générer un paquet consistera à prendre un échantillon de l'espace latent Z, à le faire passer dans le décodeur du VAE [4] pour pouvoir recréer un paquet. Le VAE [4] seul sera également utilisé pour la classification de trafic réseau basé sur des statistiques. Il offrira un moyen de visualisation simple et ouvrira la voie à l'augmentation de jeu de données réseau (en permettant la génération de nouveaux échantillons).

1.4 Recurrent Neural Network (RNN)

A l'échelle du réseau de nombreuses caractéristiques séquentielles peuvent être exploitées. Par exemple, un flux possèdent une succession de paquets dans un certain ordre, les champs d'un en-tête sont organisées de façon séquentielle, etc. L'architecture Recurrent Neural Network (RNN) permet d'apprendre des motifs séquentiels au sein des données en entrée.

Nous utiliserons ce type d'architecture pour extraire des informations issues de noms de domaine dans le cadre de la classification de trafic réseau dans le chapitre 2. En effet, les noms de domaine possèdent une structure séquentielle qui peut être intéressante à exploiter pour la classification de trafic réseau. Cette architecture sera également utilisée pour effectuer de la génération de trafic réseau car un trafic réseau possède une structure séquentielle à plusieurs niveaux. Tout d'abord, il est composé d'une succession de paquets, inclus dans une succession de flux, eux-mêmes inclus dans une succession d'échanges entre des applications, etc. Nous l'utilisons également dans le cadre de la compression afin de réaliser une compression séquentielle d'en-tête réseau. L'architecture prendra en entrée

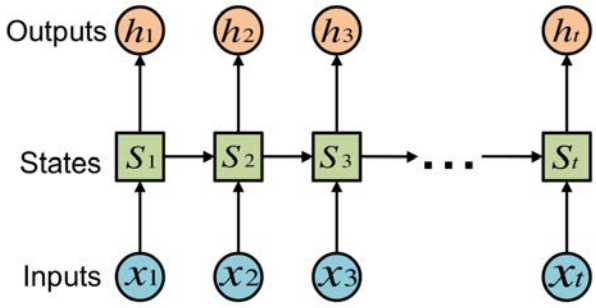

FIGURE 1.5 Architecture RNN extrait de [2].

les bits contenus dans l'en-tête à compresser ainsi que les bits de l'en-tête des paquets précédent. Cette architecture nous permet de prendre en compte la séquentialité des bits de l'en-tête ainsi que celle des paquets précédents.

Les problèmes d'évanouissement et d'explosion du gradient rendent les RNN traditionnels difficiles à entrainer [86]. L'architecture Long Short Term Memory (LSTM) [87], atténue ces problèmes en introduisant un ensemble de « passerelles ». Ces passerelles sont des opérations matricielles effectuées sur l'entrée qui permettent de réduire les problèmes de gradient et améliorent les performances des RNN traditionnels.

Cependant, l'ajout de ces passerelles augmente le nombre de paramètres. Les LSTM [87] deviennent donc très long à apprendre. Pour pallier ces limites, les Gated Recurrent Network (GRU) [88] ont été introduits. Leur objectif est d'offrir les mêmes avantages qu'un LSTM [87] avec l'utilisation des passerelles mais en utilisant un nombre de paramètres inférieur. C'est donc pour ces avantages que l'architecture GRU [88] sera celle privilégiée tout au long de cette thèse.

Le défaut majeur des RNN est de ne prendre en compte la séquentialité des données uniquement dans un sens. L'architecture peut lire les données séquentielles de gauche à droite ou de droite à gauche. Ainsi, les données lues en dernier par le modèle auront naturellement plus d'importance. Pour contrer ce bais, les RNN bidirectionnels [5] ont été développés car ils peuvent fonctionner en utilisant des cellules LSTM [87] (LSTM bidirectionnels) ou GRU [88] (GRU bidirectionnels). La Figure 1.6 illustre ce type de mécanisme.

Ainsi, l'architecture GRU bidirectionnelle devient une extension du GRU standard prenant en compte l'information contextuelle à la fois du passé et du futur d'une séquence. Le GRU bidirectionnel combine deux GRU indépendants, un fonctionnant dans la direction avant (de gauche à droite) et l'autre dans la direction arrière (de droite à gauche).

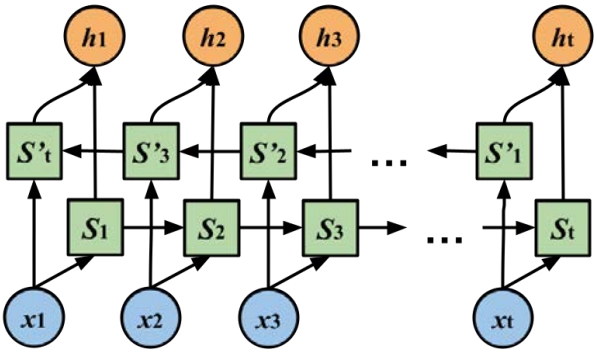

FIGURE 1.6 Architecture RNN bidirectionnel [5].

Les sorties des deux GRU sont ensuite concaténées pour obtenir une représentation bidirectionnelle. Cela permet au modèle d'apprendre à la fois des dépendances passées et futures dans la séquence. La version RNN bidirectionnelle avec des GRU [88] sera utilisée au cours du chapitre 3 et du chapitre 4 de cette thèse pour effectuer de la génération et de la compression de trafic réseau.

1.5 Hyper-paramètres particuliers

Les architectures présentées précédemment comportent un ensemble d'hyper-paramètres dont certains seront modifiés au cours de cette thèse. Les paramètres sont les suivants :

— **Fonction de coût :** elle représente l'objectif visé par l'architecture d'apprentissage profond.

— **Fonction d'activation :** elle permet d'introduire de la non linéarité. Sa structure a une influence sur l'expressivité du réseau de neurones.

Le Tableau 1.1 effectue une liste des différentes fonctions de coût et leur utilisation au cours de cette thèse.

Le Tableau 1.2 répertorie les fonctions d'activation utilisées au cours de cette thèse définie pour $x \in \mathbb{R}$. Leur choix est fait en fonction de l'état de l'art actuel dans le domaine de l'apprentissage profond.

Pour chaque domaine d'application, nous proposons des architectures complexes qui combinent toutes les architectures précédemment présentées avec les hypers-paramètres ci-dessus. De façon classique nous pouvons associer à chaque problème une architecture. Ici, nous avons cherché à assembler ces différentes architectures en modifiant leurs structures et leurs hyper-paramètres. Par exemple, dans le cadre de la génération de trafic réseau plusieurs architectures sont assemblées avec, pour chacune d'entre-elle, un sous objectif

TABLEAU 1.1 Liste des fonctions de coût utilisées abordées au cours de cette thèse.

Fonctions de coût	Formules	Utilisations
Mean Absolute Error (MAE)	$\sum_{i=1}^{D} \lvert x_i - y_i \rvert$ avec D le nombre d'individus utilisés.	Régression
Categorical Cross-Entropy (CCE)	$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$ avec M le nombre de classes, y l'indicateur binaire 0 ou 1 si le label de la classe c est la classification correct de l'observation o, p la probabilité prédite de l'observation o dans la classe c.	Classification
Mean Squared Error (MSE)	$\sum_{i=1}^{D} (x_i - y_i)^2$ avec D le nombre d'individus utilisés.	Régression
Kullback-Leibler divergence (KL)	$KL(\hat{y} \lvert\rvert y) = \sum_{c=1}^{M} \hat{y}_c \log \frac{\hat{y}_c}{y_c}$	Estimation, mesure la divergence entre deux distributions.

TABLEAU 1.2 Liste des fonctions d'activation les plus courantes d'un neurone formel avec $x \in \mathbb{R}$.

Type de fonctions	Formules
Rectified Linear Unit (ReLU) [89]	$\mathbf{ReLU(x)} = \max(\mathbf{x}, 0)$
Leaky Rectified Linear Unit (ReLU) [90]	$\mathbf{LeakyReLU(x)} = \max(\epsilon\mathbf{x}, \mathbf{x}), avec \epsilon \in]0, 1[$
Softmax	$\mathbf{softmax(x}_i\mathbf{)} = \frac{e^{\mathbf{x}_i}}{\sum_{j=0}^{k} e^{\mathbf{x}_k}}$ avec $i, k \in \mathbb{N}$ avec k le nombre de classes utilisées pour la classification.

associé. Cela nous permet d'introduire de la flexibilité mais également d'augmenter les performances car, comme vu précédemment, chaque architecture possède un rayon d'action défini. De même, plusieurs fonctions de coût sont additionnées afin de répondre au mieux à plusieurs sous objectifs visés.

1.6 Outils d'analyse

Les architectures d'apprentissage profond permettent de répondre à de nombreux enjeux dans le domaine des réseaux. Néanmoins, ces architectures peuvent être assimilées

à des « boîtes noires » où leur interprétation est limitée. Au cours de cette thèse, nous utiliserons différents outils supplémentaires afin d'interpréter et de comprendre la prise de décision des modèles développés vis à vis d'une entrée donnée. Parmi ces outils, nous avons utilisé les « Occlusion Maps » [8] et GradCAM [6].

FIGURE 1.7 Illsutration du fonction de Gradient Class Activation Map (GradCAM) [6] extrait de [6].

Ces deux outils visent au même objectif qui est l'interprétation d'une architecture d'apprentissage profond mais avec de légères différences. GradCAM [6] va chercher a détecter les zones de l'image qui vont faire le plus réagir les filtres convolutifs. Le résultat de GradCAM [6] sera une carte de couleur illustrée sur la Figure 1.7 dont les couleurs chaudes indiqueront une réaction importante des filtres collectifs. Une réaction importante de ces filtres impliquera un impact important sur la sortie du modèle dans le cas d'une architecture convolutive. GradCAM [6] est donc limité à ce type d'architecture et ne permet pas d'interpréter le modèle à une faible granularité. Par exemple, l'impact individuel d'un pixel sur la sortie. Nous exploitons ce type d'outils lors de chapitre 2 pour interpréter le modèle de classification de trafic réseau, basé sur des architectures convolutives, que nous avons développé.

Pour remédier à ces limites, nous utilisons un autre outil appellé « Occlusion Maps » [8]. L'objectif est d'appliquer un masque sur plusieurs zones à l'entrée du modèle d'apprentissage profond et de voir comment évolue la sortie. Si la sortie du modèle évolue de façon importante après application du masque, alors, la zone de l'entrée du modèle est une zone d'intérêt importante pour la prise de décision du modèle. L'avantage de ce type d'approche est de pouvoir l'utiliser indépendamment du type d'architecture utilisée. De plus, suivant la taille du masque appliqué sur l'entrée une granularité plus ou moins fine dans l'interprétation peut être obtenue. C'est pourquoi, nous l'utiliserons au cours du

chapitre 2 pour comprendre finement les zones d'impacts du modèle lors de classification de trafic réseau et lors du chapitre 4 et 5 pour interpréter et transformer le modèle de compression d'en-tête pour le rendre déployable sur un équipement.

1.7 Conclusion

En général, l'avantage des réseaux neuronaux, omniprésents en apprentissage profond, est d'être des approximateurs universels parcimonieux et donc des classifieurs ou régresseurs universels. De plus, il bénéficie d'une facilité de développement et d'une forte rapidité d'exécution en particulier lors de la présence de plusieurs processeurs. Enfin, la robustesse de la plupart des solutions permet une forte résistance au bruit des données (aux « erreurs » apprises) qui peuvent être très présentes dans le domaine des réseaux.

Cependant, le choix de l'architecture est empirique et nécessite de longues phases de tests et d'apprentissage pour trouver la bonne configuration. Il n'existe aucune règle qui permettrait de déterminer le nombre de neurones à placer dans la couche cachée pour avoir un réseau de neurones optimal. La méthode pour s'en rapprocher est d'essayer « au hasard » plusieurs nombres de neurones en couche cachée jusqu'à avoir des résultats les plus probants possibles après l'apprentissage. De plus, le temps d'apprentissage peut être long et coûteux en puissance de calcul. Les résultats restent également difficiles à interpréter en termes de connaissance. Enfin, la présence de minima locaux dans la fonction de coût (qui n'est pas forcement convexe) rend difficile sa minimisation.

Dans le prochain chapitre, nous aborderons un premier cas d'application de l'apprentissage profond au sein des réseaux : la classification de trafic réseau chiffré. Nous proposerons une architecture permettant de pallier les limites des approches traditionnelles et celles par apprentissage profond actuelles.

Chapitre 2

Classification « en direct » de trafic réseau chiffré

Sommaire

2.1 Introduction

L'objectif de ce chapitre est d'étudier l'utilisation des approches de classification de trafic réseau chiffré. Nous montrons les limites des approches actuelles et nous proposons une alternative afin de permettre la classification de trafic réseau chiffré « en direct » par apprentissage profond.

La mise en place d'un réseau dépend fortement d'une bonne prédiction du trafic conduisant à un déploiement adéquat et à la mise en place de mécanismes de Quality of Service (QoS). L'ingénierie du trafic pourrait être remise en cause par une faible connaissance des flux réellement transportés. Les systèmes de détection d'intrusion Network Intrusion Detection System (NIDS) peuvent être fondés sur des profils de trafic pour détecter différents types d'attaques. Les opérateurs de réseaux doivent être en mesure de classer leur trafic afin de mettre en œuvre des mesures d'atténuations et de priorisations du trafic, entre autres. La gestion des flux de données est devenue une activité essentielle des réseaux, ce qui renforce le besoin d'outils efficaces de classification des flux de données. Cette classification de trafic réseau va consister, par exemple, à attribuer aux ensembles de paquets qui traversent le réseau un type de trafic tel que : voix, vidéo, données, texte et trafic web. Le plus souvent, chaque classe de trafic est associée à des contraintes différentes : la voix nécessite un délai constant tandis que la donnée nécessite d'éviter les pertes. Cependant, le chiffrage généralisé des communications a considérablement compromis les outils existants.

Les avancées récentes en apprentissage profond offrent une grande variété d'architectures qui semblent pertinentes pour la classification de trafic. Ces architectures, présentées dans le chapitre 1, sont basées sur différentes représentations de données en entrée du processus de classification. Dans ce chapitre, nous montrons la nécessité d'approfondir la compréhension des caractéristiques utilisées par les modèles d'apprentissage profond pour effectuer une telle classification. Nous pensons que les en-têtes et la charge utile doivent être traitées séparément car ils contiennent différents types d'informations. Nous montrons, dans ce chapitre, que la suppression des en-têtes augmente les performances de classification. Nous montrons également l'intérêt qui existe à exploiter les statistiques de réseau.

Dans une première partie, nous faisons un état de l'art des techniques de classification de flux réseau. Dans une seconde partie, nous présentons le jeu de données utilisé et les pré-traitements appliqués. Notre objectif est de définir un meilleur traitement des données afin que l'architecture choisie améliore significativement le processus de classification. Dans une troisième partie, nous évaluons les approches actuelles et nous montrons leurs

limites. Dans une quatrième partie, nous proposons une nouvelle architecture modulaire d'apprentissage profond appelée SPPNet pour surmonter les inconvénients des approches traditionnelles. Nous montrons par une preuve de concept que SPPNet permet d'effectuer une classification de flux réseau au niveau des paquets. Enfin, en dernière partie, nous essayons de comprendre les facteurs qui influent sur les performances de classification et nous proposons une approche basée sur les statistiques réseaux.

2.2 État de l'art

Les classifieurs de trafic réseau sont des outils qui analysent le trafic réseau pour le trier en fonction de ses caractéristiques et de son origine, afin d'appliquer des traitements spécifiques. Ils ont pour objectifs d'améliorer les performances du réseau, de renforcer la sécurité et de gérer les ressources de manière optimale.

Les classifieurs peuvent être classés en trois catégories, les classifieurs traditionnels, les classifieurs par apprentissage automatique et les classifieurs par apprentissage profond. Face à certaines limites des générateurs traditionnels, les classifieurs par apprentissage automatique puis par apprentissage profond ont émergé.

Dans un premier temps, nous présentons le fonctionnement de la classification de trafic réseau tel que les catégories de classifieurs et des caractéristiques utilisées pour effectuer la classification. Dans un second temps, nous présentons et classons les approches par apprentissage profond. Pour finir, nous parlerons des limites des approches traditionnelles, par apprentissage automatique et par apprentissage profond.

2.2.1 Classification de trafic réseau

Définition des classes

La classification est la toute première et la plus importante partie de la caractérisation du trafic. Nous pouvons définir les classes de différentes manières en prenant comme base l'applicatif, (Skype, Google, Mèl) [91], le type de trafic (VoIP, vidéo, navigation web, Peer-To-Peer (P2P)) ou les protocoles (File Transfer Protocol (FTP), Transmission Control Protocol (TCP), Hypertext Transfer Protocol (HTTP)) [9], entre autres. Ce choix dépend de l'utilisation que l'on souhaite donner à notre classifieur, il n'y a donc pas de raison objective pour préférer une représentation par classe plutôt qu'une autre. Cependant, l'utilisation d'un modèle ou d'un autre entraînera des différences lors de l'étape de labelisation. La collecte des données sera conditionnée par ce choix.

Classification « avec état » ou « sans état »

La classification peut être mise en œuvre sur la base d'un flux ou au niveau des paquets. Dans le contexte de classification de trafic Internet Protocol (IP), un flux se définit comme l'ensemble des paquets appartenant au tuple : addresse IP source, adresse IP destination, port source et port destination.

Une classification basée sur le flux peut utiliser des informations globales, telles que l'inter-arrivée des paquets du flux [9]. Une classification au niveau des paquets est un processus sans état basé uniquement sur les informations portées par un seul paquet (classification « sans état »). À l'inverse, une classification basée sur les flux peut, si nous le souhaitons, s'effectuer en gardant un état sur le flux à classifier (classification « avec état »). Cela permet d'extraire des statistiques, pour la détection d'intrusion, entre autres.

Classification « en direct » ou « différée »

La classification de trafic peut aussi être catégorisée en deux sous classes : « en direct » ou « différée ». La classification « en direct » fait référence à la classification qui se fait à la réception des paquets ou flux. Les paquets ou flux ont besoin d'être classifiés le plus rapidement possible. Dans le cas d'une classification de niveau flux « en direct », l'intérêt n'est porté que sur un échantillon de paquets du flux. Ce type de classification a un intérêt pour la QoS ou le routage afin de permettre une prise de décision rapide.

À l'inverse, la classification « différée » peut s'effectuer en décalé après réception des données. C'est le cas des systèmes de transaction entre autres. Le choix des caractéristiques est un élément qui conditionne le type de classification effectuée. Par exemple, une classification, de niveau flux, qui nécessite la collecte d'un grand nombre de paquets pour prendre une décision ne pourra pas être envisagée dans le cadre d'une classification « en direct ».

Caractéristiques utilisées

En reprenant l'analyse de [9] nous pouvons voir que la classification de trafic réseau prend en entrée une ou plusieurs de ces caractéristiques :

— **Séries temporelles :** les caractéristiques de séries temporelles peuvent représenter la taille du paquet, intervalle d'arrivée et la direction des paquets consécutifs. L'extraction de caractéristiques sur les 20 premiers semblent raisonnable [92]. De plus, cette technique a montré de bonnes performances [93].

— **En-tête :** l'information contenue dans les en-têtes peut être utilisée, généralement de niveau 3 et 4 en suivant le modèle Open Systems Interconnection (OSI) (par

exemple, extraite du protocol IP, TCP ou User Datagram Protocol (UDP)). Parmi ces caractéristiques, nous retrouvons le numéro de port, le protocole et la taille du paquet, entre autres.

— **Données :** les données contenues dans un paquet peuvent contenir de l'information ou des motifs utilisables pour la classification [94]. Néanmoins, ce type d'approche est limité par l'utilisation du chiffrement qui peut masquer certains motifs [95, 96].

— **Statistiques :** regroupe les statistiques qui peuvent être extraites à partir de flux comme la taille moyenne des paquets, la taille maximale, la taille minimale, l'intervalle minimal d'arrivée entre deux paquets, etc. Les statistiques sont extraites sur un regroupement pouvant aller de 10 à 180 paquets en fonction du jeux de données et des caractéristiques choisies [97, 98]. Ce type de caractéristique n'est pas adapté pour la classification « en direct » car elle nécessite de collecter un ensemble de données important. La seconde limite de ce type de caractéristique, c'est la généralisation, il faut être sûr que les caractéristiques extraites sont représentatives de celles observées dans la réalité.

Les caractéristiques choisies et utilisées peuvent être combinées. C'est le choix qui a été fait par certains en combinant les informations contenues dans l'en-tête et les séries temporelles dans l'objectif de pallier le chiffrement. Généralement, les CNN et LSTM semblent offrir de bonnes performances [92], et les CNN seuls permettent de gérer un faible nombre de paquets [92, 93]. D'autres approches combinent l'en-tête et la donnée. Dans de tels cas, le CNN ou la combinaison de CNN et LSTM sont réputées pour leur grande précision [91, 99–101]. Le choix des combinaisons permet d'offrir un gain de performances mais le choix du type de classification à utiliser doit être à prendre en compte (« en direct » ou « en différé » avec ou sans état). Les caractéristiques statistiques s'utilisent sans combinaison avec d'autres caractéristiques et utilisent en majorité des approches d'apprentissage automatique classiques.

2.2.2 Classifieurs traditionnels

Plusieurs techniques de classification au niveau des paquets ont été mises en œuvre pendant des décennies pour les réseaux IP. La première approche et la plus courante est d'utiliser les numéros de port [102]. Les numéros de port pour chaque application sont attribués par l'Internet Assigned Numbers Authority (IANA) [103]. Cependant, les performances de cette approche n'ont cessé de décroître au cours du temps avec l'arrivée de nouvelle application au port inconnu ou non standardisé. Malgré cela, ce type d'approche reste encore largement utilisé de part sa simplicité. Une autre approche se focalise sur le contenu du paquet, appelé Deep Packet Inspection (DPI). L'objectif est de trouver des

motifs ou des informations permettant de classifier le trafic [94].

La principale limite des deux approches citées est le chiffrement. Des techniques d'obfuscation de ports peuvent être utilisées et rendre obsolète la première approche de classification. Le chiffrement peut également être appliqué sur les données et rendre obsolète la seconde approche [95, 96].

2.2.3 Classifieurs par apprentissage automatique

Afin de faire face à l'obfuscation des en-têtes, voire au chiffrage des flux de données, des techniques d'apprentissage automatique ont été introduites principalement fondées sur les statistiques. Ces statistiques peuvent être associées à des flux, des paquets ou les deux. Plusieurs méthodes d'apprentissage automatique ont été appliquées telles que K-Nearest Neighbors (kNN) [104, 105], Naives Bayes (NB) [106], XGBoost [107].

Cependant, le choix des caractéristiques est important et peut avoir un impact considérable sur les performances du modèle final et sa capacité à généraliser. En outre, le chiffrement peut masquer un certain nombre de caractéristiques, en modifiant la taille des paquets par exemple, et ainsi réduire les performances.

2.2.4 Classifieurs par apprentissage profond

Les méthodes d'apprentissage profond apparaissent comme un moyen d'extraire automatiquement et hiérarchiquement les caractéristiques pertinentes pour la classification. Elles constituent aujourd'hui des méthodes de pointe pour la classification du trafic réseau chiffré. Elles permettent de pallier les limites des approches par apprentissage automatique classiques dont les performances dépendent fortement des caractéristiques extraites et utilisées en entrée des modèles. Parmi les méthodes basées sur le contenu des paquets, nous trouvons principalement des architectures de convolution [91, 100, 108, 109]. Certains utilisent des architectures récurrentes et de convolution pour exploiter les caractéristiques séquentielles existant dans les paquets d'un flux [110, 111]. D'autres n'utilisent que l'architecture récurrente [112] ou des dérivés de l'architecture de convolution comme la convolution de texte [113].

Le Tableau 2.1 inspiré de [9] résume l'ensemble des méthodes existantes jusqu'à aujourd'hui pour la classification de trafic réseau chiffré par apprentissage profond.

Tableau 2.1 Résumé des approches par apprentissage profond pour la classification de trafic réseau extrait de [9].

Publications	**Catégories**	**Méthodes**	**En direct**	**Caractéristiques**	**Années**
Ons Aouedi [114]	Détection d'intrusion	AE	Non	Statistiques	2022
Shahbaz Rezaei [115]	Identification de trafic	CNN	Non	Séries temporelles	2020
Ons Aouedi [116]	Classification d'APP	SSAE	Non	Statistiques	2020
Peng Li [117]	Détection d'intrusion	SAE	Oui	En-tête+données	2018
Wang-2018 [101]	Détection d'intrusion	CNN+LSTM	Oui	En-tête+données	2018
Rezaei [9]	Identification d'APP/OS	CNN	Non	Séries temporelles échantillonnées	2018
Aceto [93]	Classification d'APP	CNN/LSTM/ SAE/MLP	Oui	En-tête+données	2018
Vu [118]	Identification de trafic	AC-GAN	Non	Statistiques	2017
Wang-2017 [100]	Identification de trafic	CNN	Oui	En-tête+données	2017
Seq2Img [108]	Identification d'APP/protocoles	RKHS+CNN	Oui	Séries temporelles	2017
Lotfollahi [91]	Identification d'APP/trafic	CNN/SAE	Non	En-tête+données	2017
Lopez-Martin [92]	Classification variée	CNN+LSTM	Oui	En-tête+séries temporelle	2017
Wei Wang [119]	Identification de trafic	CNN	Non	En-tête+données	2017
Hochst [120]	Identification de trafic	Autoencoder	Non	Statistique+en-tête	2017
Zhanyi Wang [121]	Identification d'APP/trafic	ANN	Oui	En-tête+données	2015

2.2.5 Limites des approches actuelles

Malgré le chiffrement, les approches par apprentissage profond arrivent à montrer de bonnes performances pour la classification de trafic chiffré [9, 91, 100, 101]. Le principal défi de ces techniques est donc de comprendre l'impact de chaque caractéristique des données utilisées par les modèles d'apprentissage profond. Par conséquent, les performances et les capacités de généralisation d'un système ne peuvent pas être évaluées facilement. C'est pourquoi nous chercherons, dans un premier temps, à comprendre quelles sont les caractéristiques utilisées par les approches d'apprentissage profond lors de la classification. Nous pensons que cette information nous aidera à mieux définir l'architecture d'apprentissage profond pour la classification de trafic réseau chiffré.

2.3 Traitement des données

Cette section décrit les données et les traitements effectués afin de réaliser une classification sans état avec des modèles d'apprentissage profond. Comme déjà indiqué, notre premier objectif est de déterminer la meilleure représentation des données. Pour cela, nous devons effectuer plusieurs traitements sur l'ensemble des données.

2.3.1 Collecte des données

Jeux de données « chiffrées »

Les données chiffrées utilisées pour les expériences proviennent de deux sources distinctes. La première source comprend deux jeux de données en accès libre utilisés comme références dans le domaine de la classification des flux réseau chiffrés : ISCXVPN2016 [122] et ISCXTor2016 [123]. Ces ensembles de données présentent une diversité d'applications avec un grand nombre de paquets. Nous utiliserons deux sous-ensembles de ces sources : un jeu de données collecté dans un réseau The Onion Router (TOR), et un jeu de données de paquets chiffrés avec Transport Layer Security (TLS) [7].

La deuxième source comprend les données recueillies dans notre laboratoire en utilisant la même configuration que pour la collecte des données ISCXVPN2016 [122] et ISCXTor2016 [123].

L'ensemble des données collectées est réparti sous la forme des deux jeux de données nommés comme suit :

— **« Jeu de données de référence »** : comprend les données TLS [7] et TOR provenant des jeux de données ISCX. Ils sont utilisés pour l'entraînement et l'évaluation partielle des modèles réalisés.

— **« Notre jeu de données »** : comprend les données TLS [7] et TOR collectées dans notre laboratoire et sont utilisées pour l'évaluation des modèles.

Cette utilisation de sources diverses nous permet de vérifier les capacités de généralisation de notre outil de classification. Pour ces jeux de données, nous définissons sept classes de paquets : Chat, E-mail, Transfert de fichiers, P2P, Voice Over IP (VoIP), Streaming et Navigation Internet. Cette séparation permet de conserver une cohérence concernant les problématiques réseau tout en maximisant la diversité des applications pour chaque classe. VoIP est un type de trafic plus contraint que le Streaming qui lui-même est plus contraint que le Transfert de fichiers. Le but d'un outil de classification est donc de classer les paquets dans l'une de ces classes. Une telle définition des classes conduit à une diversité d'applications au sein de chaque classe. Cela permet d'évaluer au mieux la capacité de généralisation des modèles et, plus particulièrement, leur capacité à faire face à de nouvelles applications. Le Tableau 2.2 représente la liste des applications présentes dans les données de référence et dans notre jeu de données.

Tableau 2.2 Liste des applications dans chaque classe pour chaque groupe des deux jeux de données.

Classes	**Groupes**	**Jeu de données de reférence**	**Notre jeu de données**
Chat	TLS [7]	Gmail, Hangout, Skype	Facebook, Telegram
	TOR	Facebook, ICQ, AIM	qTox
Mèl	TLS [7]	Gmail	Outlook, Gmail
	TOR		GMX Caramail
File transfer	TLS [7]	Facebook, Skype, FTP/SFTP Fillezilla	SSH, Google Drive
	TOR		DropBox
P2P	TLS [7]	uTorrent	Deluge
	TOR		Vuze
Streaming	TLS [7]	Vimeo, Youtube, Netflix	Youtube, Dailymotion
	TOR		
VoIP	TLS [7]	Facebook, Hangout, Skype, VoIPBuster	Facebook
	TOR		qTox
Navigation Internet	TLS [7]	Firefox, Chrome	Mozilla Firefox
	TOR	Navigateur Tor	Navigateur Tor

Jeu de données « non chiffré »

Le jeu de données « non chiffré » comprend 50 000 paquets et a pour objectif d'étudier l'impact du chiffrement et des caractéristiques statistiques des flux. Nous faisons le choix de définir nos classes afin d'être au plus proche des classes du jeu de données « chiffré » de trafic à partir des types de trafics existants. Pour chaque classe de trafic, nous avons dû définir une architecture et simuler l'échange de données. L'utilisation d'application générique n'est pas possible car elle utilise un protocole chiffré. Nous avons donc les classes suivantes :

- **Streaming :** les données sont collectées en utilisant le service IPTV de l'opérateur Free pendant 1 heure. Les données sont transportées en UDP et comprenne de l'audio et de la vidéo.
- **Voix :** les données sont collectées entre deux clients connectés à un serveur Asterisk [124]. La voix est simulée en utilisant un podcast audio d'une durée d'une heure. Les données sont véhiculées en UDP.
- **Navigation internet :** un ensemble de sites, utilisant le protocole HTTP est sélectionné. Les données sont collectées entre le client et le serveur web contacté.
- **Mail :** un serveur de mail local est utilisé. Des adresses mails locales sont utilisées pour s'échanger les mails grâce au protocole Simple Mail Transfer Protocol (SMTP). Les textes contenus dans les mails sont générés grâce à un jeu de données comprenant des mails de type « spam » : 10 000 mails sont collectés.
- **Chat :** les données sont collectées en récupérant les communications entres deux clients au travers d'un serveur Internet Relay Chat (IRC). Les messages sont simulés en utilisant un jeu de données regroupant des commentaires issus d'Amazon sur des produits variés ainsi qu'un jeu de données de SMS.

La classe P2P n'est pas présente car plus complexe à générer sans chiffrement. Les données sont collectées avec la librairie Scapy [125] avec Python 3. Les en-êtes sont supprimés car nous verrons dans la suite du chapitre que celles-ci ont un impact important sur les performances de classification. De plus, le nombre de paquets est équilibré entre les classes.

2.3.2 Pré traitements

Plusieurs niveaux de pré traitements sont effectués. Un premier niveau consiste à filtrer les données originales. Un deuxième niveau est axé sur les représentations et les formats de données. Enfin, le dernier niveau prépare les données pour l'étape d'apprentissage des modèles d'apprentissage profond. Ces traitements sont appliqués sur les deux jeux de

données (« chiffré », « non chiffré »).

Pour le premier niveau de pré traitement, des opérations de filtrage sont utilisées pour éliminer les informations qui peuvent causer un biais. Les paquets d'initialisation de connexion TCP, TLS [7], Quick UDP Internet Connections (QUIC) sont supprimés. Comme le montre l'article [126], certains champs dans les en-têtes de ces paquets peuvent faciliter ou biaiser excessivement la classification. Certains travaux ne les suppriment pas : [91, 100, 108–111, 113]. Nous montrerons lors de l'expérience que certains champs d'en-tête peuvent introduire un biais en utilisant une architecture convolutive pour les données chiffrées.

Le deuxième niveau de pré traitement vient du constat qu'aucune comparaison entre ces différentes représentations et formats n'a été réalisée dans la littérature. Nous avons choisi d'étudier conjointement deux formats et deux représentations des données listées dans le Tableau 2.3. Des octets de « bourrage » sont utilisés pour obtenir des tailles uniformes.

Tableau 2.3 Liste des représentations, formats et formes de données étudiés.

Représentations	**Formats**	**Plage de valeurs**	**Dimensions**
2D	Entier	0 - 255	40×40
	Bit	0 - 1	111×111
1D	Entier	0 - 255	1536×1
	Bit	0 - 1	12288×1

Intuitivement, nous supposons que le format binaire permet d'accéder à un plus haut niveau de granularité de l'information. Cependant, la probabilité qu'une séquence binaire se retrouve dans des paquets de classes différentes est élevée, ce qui peut introduire du bruit. La représentation bidimensionnelle peut réduire cet effet car la probabilité de caractéristiques similaires à deux dimensions entre deux classes différentes est moins probables. D'autre part, le format entier permet de résumer l'information en regroupant les bits en octets, ce qui réduit le bruit.

La Figure 2.1 montre deux paquets en représentation matricielle, un paquet en format entier et un paquet en format binaire. En traitant les données ainsi les paquets réseaux peuvent être assimilées à des images. Tout comme les images, les paquets présentent des caractéristiques locales qui permettent la classification d'où les similarités qui existent dans leurs prétraitements. Dans un cadre réseau, cela peut être la présence de balises HTML qui permettent d'associer le paquet à de la navigation web.

Le « jeu de données de référence » est sous-échantillonné avec équité entre les classes et les applications pour assurer la représentativité. Un total de 81 003 paquets différents

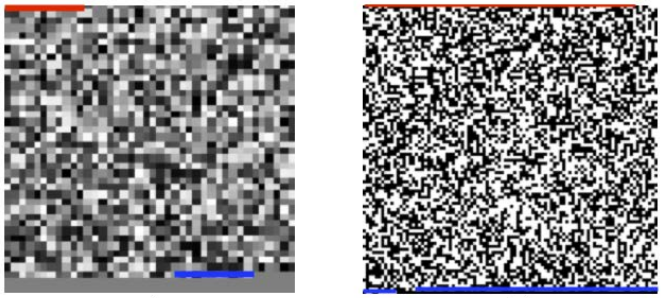

FIGURE 2.1 Paquets en représentation 2D provenant du « jeu de données de référence » au format a) entier et b) binaire, appartenant au groupe TLS [7]. La taille du paquet est de gauche à droite, $\mathbf{40 \times 40}$ et $\mathbf{111 \times 111}$. Les 13 premiers octets de chaque paquet sont colorés en rouge et les derniers en bleu.

est utilisé.

Les données d'entrainement correspondent à 80% des données de chaque jeu de données. Les données de validation et de test se partagent respectivement 10% de l'ensemble des données. Les plus petits paquets sont supprimés afin de faciliter le processus d'apprentissage. De toute évidence, ces paquets ne contiennent que peu ou pas de données, ils pourraient donc être classés sur la base de leurs seuls en-têtes. Cela aurait un sens dans une classification avec état, mais pas dans cette étude où la notion de flux n'est pas exploitée.

Pour les jeux de données « chiffrés » (« jeu de données de référence » et « notre jeu de données »), l'évaluation des modèles est effectuée sur les données de test et sur 21 000 paquets sous-échantillonnés de notre ensemble de données. Ces 21 000 paquets sont de tailles diverses entre 0 et 1536 octets et utilisent des applications non-présentes dans les données d'entraînement. Seules les données au format entier sont normalisées entre 0 et 1. Pour le jeu de données « non chiffré », l'évaluation est effectuée sur des données issues des mêmes applications que pour les données d'entraînements.

2.4 Classification sans état au niveau des paquets par apprentissage profond

Cette section décrit les expériences réalisées afin de déterminer conjointement la meilleure représentation des données et la meilleure hypothèse de modélisation. Cette section s'intéressera uniquement aux jeux de données « chiffré » car ces jeux de données

sont similaires à ceux utilisés dans la littérature. Les expériences montreront les limites qui existent dans les approches actuelles d'apprentissage profond pour la classification sans état au niveau des paquets. Les conclusions de cette analyse nous conduiront à la définition d'une nouvelle architecture nommée SPPNet.

2.4.1 Architecture d'apprentissage profond

Les architectures utilisées pour nos modélisations sont les suivantes :

— Une architecture ResNet18 à convolution est focalisée dans les caractéristiques invariantes en translation au sein des données grâce aux filtres de convolution appris. Ce type d'architecture est présenté dans la section 1.2 du chapitre 1.

— Un modèle de référence utilisant une simple distance euclidienne permet de vérifier la pertinence des résultats obtenus avec les approches d'apprentissage profond.

Par la suite, les modèles seront évalués et interprétées pour tous les types de formats et de représentations de données listées dans le Tableau 2.3.

2.4.2 Résultats et analyses

Performance

Les performances de classification des différents modèles en fonction des différents paramètres et du jeu de données sont respectivement listées dans le Tableau 2.4 pour les données TLS [7] et le Tableau 2.5 pour les données TOR. L'apprentissage, la classification et les mesures sont effectués sur l'architecture Osirim [81], la plateforme de calcul de l'IRIT.

Chaque ligne de chaque tableau représente un modèle évalué prenant en entrée une représentation de données (deux dimensions ou une dimension) et un format de donnée (entier ou binaire). Chaque colonne représente la précision obtenue pour chaque modèle. La précision correspond au rapport entre le nombre de paquets correctement classifiés par rapport au nombre de paquets total évalués. Donc une précision de 1 correspond à une classification parfaite, sans erreur. La répartition des paquets est équilibrée entre les différentes classes ce qui nous permet d'envisager cette métrique. Les paquets évalués peuvent être issus des données de test du « jeu données de référence » (colonne « Test ») ou de « notre jeu de données » construit en laboratoire (colonne « Notre »). L'évaluation est répétée avec différents niveaux de suppression d'en-tête pour chaque paquet. Ainsi, les valeurs de la colonne « Notre L4 » représentent les performances de chaque modèle, entraîné et évalué, sur des paquets, appartenant à « notre jeu de données », dont les en-têtes de niveau 3 (protocole TCP/UDP) ont été supprimés.

TABLEAU 2.4 Accuracy des modèles en fonction du format, de la représentation et du niveau de suppression des en-têtes pour les données TLS [7] : IP (L3), puis IP et UDP/TCP (L3/L4). Les meilleures performances pour chaque ensemble de données et la suppression des en-têtes sont en **gras**.

Data	Modèles	Type	Test L3/L4	Notre L3/L4	Test L4	Notre L4	Test	Notre
TLS [7]	ResNet18 1D	Entier	0.999	**0.381**	0.975	0.422	0.589	**0.502**
		Bit	**0.999**	0.285	**0.997**	**0.460**	0.626	0.273
	ResNet18 2D	Entier	0.991	0.278	0.945	0.341	0.636	0.405
		Bit	0.996	0.235	0.992	0.417	0.703	0.418
	L2 Distance	Entier	0.466	0.231	0.413	0.233	0.397	0.267
		Bit	0.535	0.235	0.470	0.220	0.395	0.266

TABLEAU 2.5 Accuracy des modèles en fonction du format, de la représentation et du niveau de suppression des en-têtes pour les données TOR : IP (L3), puis IP et UDP/TCP (L3/L4). Les meilleures performances pour chaque ensemble de données et la suppression des en-têtes sont en **gras**.

Data	Modèles	Type	Test L3/L4	Notre L3/L4	Test L4	Notre L4	Test	Notre
Tor	ResNet18 1D	Entier	0.999	0.162	0.997	0.162	0.651	0.155
		Bit	**0.999**	0.152	**0.999**	0.152	**0.672**	0.167
	ResNet18 2D	Entier	0.999	**0.210**	0.996	**0.216**	0.615	0.152
		Bit	0.999	0.144	0.998	0.144	0.572	**0.186**
	L2 Distance	Entier	0.394	0.108	0.386	0.108	0.372	0.095
		Bit	0.422	0.133	0.407	0.133	0.377	0.106

Nous avons également considéré une architecture récursive simple composée de cellules GRU [88] en supposant que les caractéristiques présentes dans les données sont séquentielles mais cette architecture ne converge pas. Une explication pourrait être la longueur des données d'entrée et l'absence de motif séquentiel causé par le chiffrage.

Les modèles appris à partir des données sans supprimer les en-têtes ont de bons résultats sur les données de test et sont conformes à ce qui est observé dans la littérature. La longueur des paquets n'a pas d'effet sur les performances de classification ce qui est anormal. Il y a un très grand écart entre les données de test et notre jeu de données reflétant un manque de généralisation, pour tous les modèles, indépendamment de la représentation des données. Les applications qui ne sont pas présentes dans les données d'entraînement sont mal classées. Pour les données TOR, le lieu où les données sont prises n'a aucune influence sur les performances de classification. La différence de performances par rapport aux données TLS [7] peut s'expliquer par l'effet d'un chiffrage plus robuste et de l'offuscation de la taille des paquets.

Après avoir retiré l'en-tête IP, les résultats restent similaires. Cependant, la suppression de l'en-tête TCP/UDP améliore les capacités de généralisation. Sur les données TOR, il n'y a pas d'augmentation de la performance de classification sur nos données (similaire au hasard). Dès que les en-têtes IP et TCP/UDP sont supprimés, comme la taille des paquets sur nos données augmente, les performances augmentent également. Dans les modèles utilisant des données TLS [7], les applications non présentes dans les données d'apprentissage sont correctement classées.

Interprétation

Une interprétation dans l'espace d'entrée avec GradCAM [6] et des « Occlusion Map » [8] montre les limites de tels modèles. GradCAM [6] est appliqué sur la dernière couche de convolution après la non-linéarité. Cette technique permet de voir où le filtre convolutif réagit aux données d'entrées afin d'effectuer la classification. Les couleurs chaudes signifient une forte réaction. Tous les modèles se concentrent sur les en-têtes pour la classification. Néanmoins, il y a une légère focalisation sur le contenu du paquet, ce qui peut expliquer les résultats légèrement supérieurs à l'aléatoire et au modèle de référence. Après avoir supprimé l'en-tête IP, la focalisation devient plus prononcée et plus localisée. D'autre part, la suppression de l'en-tête IP et de TCP/UDP permet une focalisation plus diffuse et moins localisée sur les en-têtes. Cette focalisation montre une meilleure capacité à générer des modèles. Ceci est moins perceptible sur les données TOR et peut être expliqué par l'utilisation de bits de bourrage qui force le modèle à se concentrer sur les éléments moins bruyants situés au début des paquets. L'implantation de GradCAM

[6] sur sept paquets de classes différentes est visible sur la Figure 2.2 pour les données de test TLS [7] au format entier et Figure 2.3 au format binaire. GradCAM montre des résultats similaires sur le format entier.

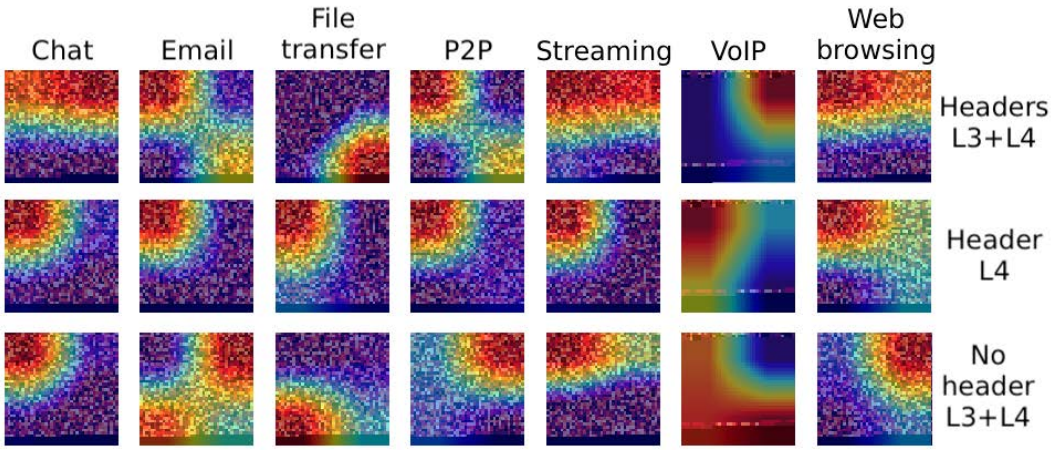

FIGURE 2.2 GradCAM [6] appliquée sur sept paquets de chaque classe en représentation bidimensionnelle au format entier du groupe TLS [7] sur des données de test. Chaque ligne représente un niveau de suppression d'en-tête de paquets. La dernière ligne de paquet ne comprend aucune en-tête de niveau 3 et 4 (IP et TCP/UDP).

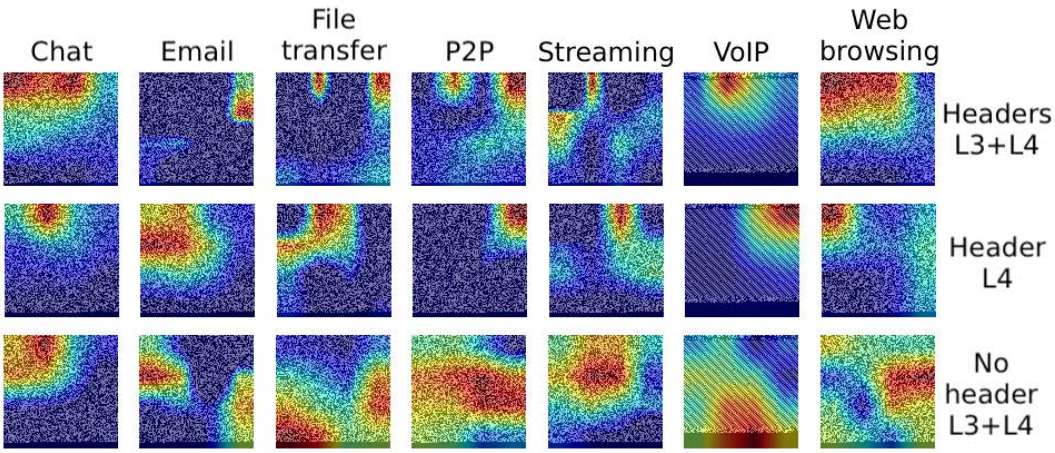

FIGURE 2.3 GradCAM [6] appliquée sur sept paquets de chaque classe en représentation bidimensionnelle au format bit du groupe TLS [7] sur des données de test. Chaque ligne représente un niveau de suppression d'en-tête de paquets. La dernière ligne de paquets ne comprend aucune en-tête de niveau 3 et 4 (IP et TCP/UDP).

Des cartes d'occlusion [8] sont également appliquées avec un filtre de 2×2 pour les paquets entiers et de 3×3 pour les paquets binaires. Le but est de cacher chaque octet ou bit d'un paquet et de voir comment évolue la probabilité de la bonne classe associée à ce paquet attribuée par le modèle de sortie. Avant de supprimer les en-têtes, les modèles se concentrent sur les octets appartenant à la source IP, la destination IP, le port source, le port de destination et parfois d'autres champs comme le numéro de séquence. Cette focalisation explique le manque de généralisation. Après avoir supprimé les en-têtes, les modèles se concentrent sur une structure plus globale située à l'intérieur du paquet mais qui ne peut être interprétée car elle est chiffrée. L'implémentation des « Occlusion Map » sur sept paquets de classes différentes dans les données de test TLS [7] issus du « jeu de données de référence » est présentée dans la Figure 2.4 pour les données TLS [7].

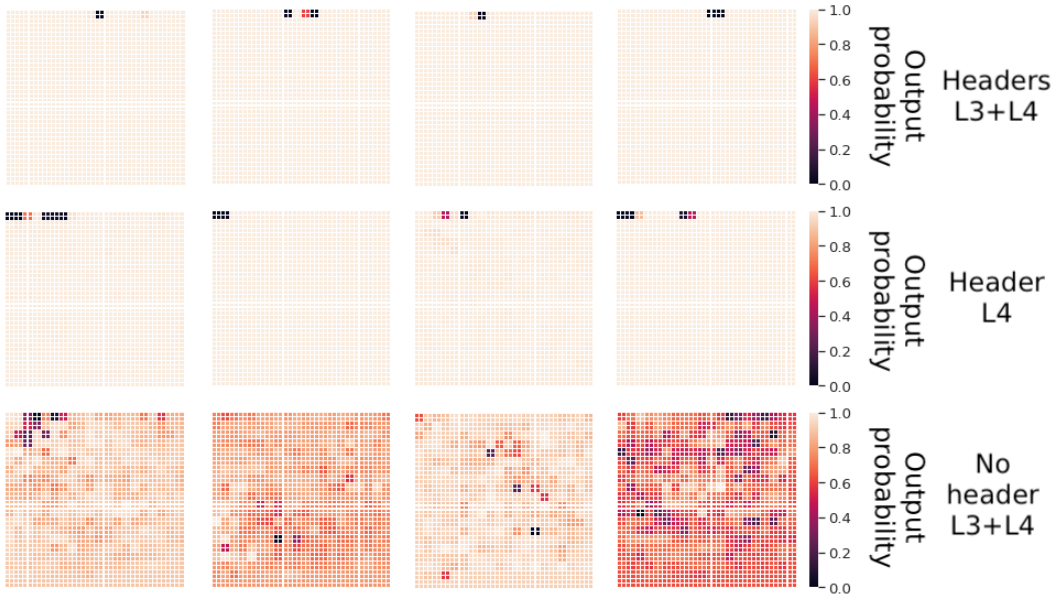

FIGURE 2.4 « Occlusion Map » [8] appliquée sur les paquets en représentation bidimensionnelle au format entier du groupe TLS [7]. Chaque ligne représente un niveau de suppression de paquets et chaque colonne un paquet aléatoire extrait des données. La dernière ligne de paquet de comprend aucune en-tête de niveau 3 et 4 (IP et TCP/UDP).

La suppression des en-têtes empêche les modèles de se concentrer sur des caractéristiques qui ne peuvent pas être généralisées. En effet, les modèles de convolution supposent l'hypothèse de motifs invariants en translation présents dans les données. Cependant, les structures de paquets d'un flux contiennent plusieurs éléments d'information nécessitant des hypothèses de modélisation différentes. Par exemple, le texte brut ne peut pas être

traité comme un simple motif. Cette observation explique l'amélioration des performances des modèles utilisant les premiers paquets d'un flux : [100, 111, 113]. En effet, l'initialisation d'une connexion TLS [7] comprend des informations telles que le nom du serveur qui constitue un motif permettant de déterminer le type d'application. Cependant, ce motif n'est pas généralisable et il est spécifique aux applications présentes dans les données d'entraînement. Par exemple, si nous voulons classifier une application, si le champ TLS [7] nom du serveur change, les modèles ne seront pas en mesure d'associer le paquet à la bonne application. Ceci explique peut-être les meilleures performances obtenues avec la convolution de texte qui est plus à même de détecter ce type de structure [113].

Nos expériences ont montré qu'une représentation unidimensionnelle au format entier avec une architecture convolutive simple offre les meilleures performances de classification. Les en-têtes provoquent un biais et rendent les approches actuelles utilisant des architectures convolutives [91, 100, 108–111, 113] peu précises en pratique. Néanmoins, certaines informations contenues dans les en-têtes devraient être exploitables. Pour ce faire, il est nécessaire de définir des hypothèses et des traitements spécifiques pour chaque information exploitée comme proposé dans la section suivante.

2.5 Architecture de SPPNet

Cette section décrit les expériences réalisées afin de déterminer conjointement la meilleure représentation des données et la meilleure hypothèse de modélisation. Les expériences montreront les limites qui existent dans les approches actuelles d'apprentissage profond pour la classification sans état au niveau des paquets. Les conclusions de cette analyse nous conduiront à la définition d'une nouvelle architecture nommée SPPNet.

2.5.1 Présentation

L'étude précédente nous amène à proposer une nouvelle architecture, appelée SPPNet qui signifie « Servername Protocol Packet Network ». Son approche vise à exploiter toutes les caractéristiques disponibles dans les ensembles de données, identifiées comme généralisables :

— **Le paquet** sans en-tête IP et TCP/UDP.
— **Le type de protocole utilisé**, UDP ou TCP. Le trafic UDP correspond à un trafic lié à VoIP et éventuellement P2P.
— **Le protocole encapsulé par UDP ou TCP** est identifié à partir des ports. Par exemple, le fait de savoir qu'un paquet porte Post Office Protocol over SSL (POP3S) peut faciliter la classification des paquets de courrier électronique.

— **Le nom de domaine** associé à l'adresse IP source ou de destination des paquets. Par exemple, « vesta.web.telegram.org » peut aider à déterminer s'il s'agit d'un paquet de classe « Chat » provenant de l'application Telegram.
— **Le nom du serveur** présent lors de l'initialisation de la connexion TLS [7] s'il est présent. Par exemple, « outlook.office365.com » peut aider le modèle à classer le paquet dans la classe Email.

L'architecture SPPNet est divisée en deux parties, comme le montre la Figure 2.5 : l'extraction des caractéristiques sémantiques de chaque source (première partie) et leurs combinaisons (deuxième partie).

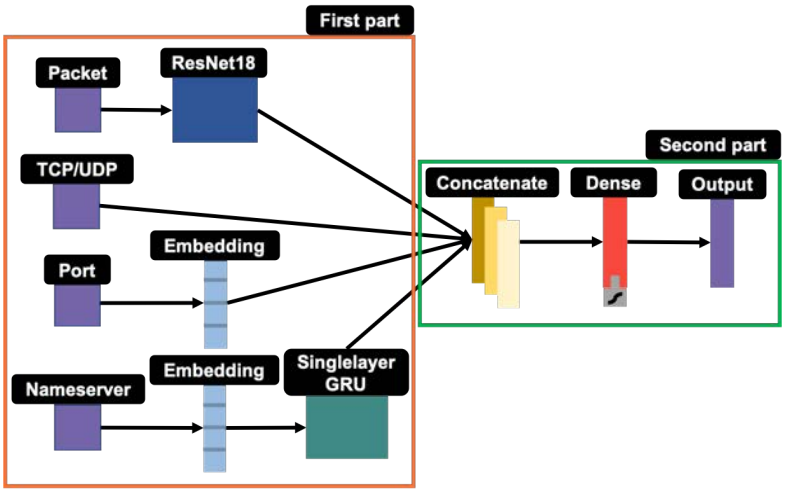

FIGURE 2.5 Diagramme de l'architecture SPPNet.

La première partie est composée des modèles suivant :

— **ResNet18** qui prendra en entrée des paquets en représentation 1D au format entier pour des raisons de performances observées dans le Tableau 2.4. La sortie est un vecteur de 512 dimensions.
— **Couche récurrente** de type GRU [88], composée de blocs de projection qui prendront le nom du serveur ou le nom de domaine en entrée. La sortie du bloc récurrent projette les données dans un vecteur à cinq dimensions.
— **Couche de projection** qui prend en entrée le nom du protocole associé au port. La couche de projection projette les données dans un vecteur bidimensionnel.
— **Couche d'entrée** qui prendra en entrée le type de protocole de transport utilisé (TCP ou UDP). La sortie est une valeur d'une dimension.

La dimension de sortie des couches de projection est fixée à la racine quatrième de la

taille du corpus en se basant les recommandation de [127]. Les sorties de chaque couche et architecture sont ensuite concaténées et connectées à une nouvelle couche entièrement connectée commune à toutes.

2.5.2 Pré traitements

Pour les paquets, les en-têtes Ethernet, IP et TCP/UDP ont été supprimés pour éviter une mauvaise généralisation. En revanche, toutes les tailles de paquets ont été conservées pour les expériences.

Pour les noms de serveurs, ils sont extraits de l'initialisation des connexions TLS [7] et QUIC en récupérant le champ du nom du serveur lorsqu'il est présent. L'enregistrement des messages Canonical NAME (CNAME) de Domain Name System (DNS) avant l'établissement d'une connexion est également utilisé. Une table d'association nom de serveur, adresse source/destination, ports source/destination est utilisée pour associer un nom de serveur à tous les paquets concernés. Dans le cas où un paquet possède à la fois un nom de domaine et un nom de serveur, le nom de domaine est choisi en priorité afin de garantir une bonne généralisation.

Les noms de serveurs sont pré-traités pour l'étape d'apprentissage. Dans le cas du nom de serveur « vesta.web.telegram.org », le traitement est effectué comme suit :

— **Phase 1** : Les points sont supprimés donnant l'entrée : « vesta web telegram org ».
— **Phase 2** : L'ordre des mots est inversé ce qui donne l'entrée : « org telegram web vesta ».

Ces opérations considèrent les noms de serveurs comme une phrase. De plus, en inversant l'ordre de la phrase, nous pouvons coder l'importance de chaque serveur dans la hiérarchie des noms de serveurs. Le nom de serveur « org » étant plus important, plus haut dans la hiérarchie, il se trouve en début de phrase. Un autre intérêt est d'apprendre à distinguer les différentes classes de trafic dans une même application.

2.5.3 Expérimentations

L'architecture SPPNet permet l'extraction de caractéristiques sémantiques liées à la combinaison de ces caractéristiques. Contrairement au simple produit de convolution, nous nous intéressons à la structure des noms de serveurs et non à leur simple présence ou absence comme dans les travaux précédents où nous utilisons la concaténation des premiers paquets d'un flux pour initialiser une connexion [100, 111, 113].

L'apprentissage se fait en deux étapes. Chaque couche ou architecture de la première partie, illustrée sur la Figure 2.5, devient un modèle appris indépendamment de manière

supervisée. Chaque couche ou architecture est connectée à un bloc complètement connecté pendant le processus d'apprentissage. Après l'apprentissage, chaque couche ou architecture est récupérée avec ses poids et déconnectée de son bloc complètement connecté. Les sorties des différentes couches d'architecture sont ensuite concaténées et connectées à une nouvelle couche entièrement connectée commune à toutes. Lors de la deuxième phase d'apprentissage, seul le bloc entièrement connecté est appris afin qu'il pondère l'importance des sorties de chaque couche par rapport à la classification finale.

Deux ensembles de données sont utilisés pour l'entraînement dans chaque phase. Ils ont subi la même division que dans les expériences précédentes et comprennent 81003 paquets échantillonnés de la même manière. Leur rôle est le suivant :

— **Jeu de données « local »** : utilisé pour l'entraînement des modèles appartenant à la première partie de l'architecture. Il comprend des paquets allant de 768 à 1536 octets.

— **Jeu de données « global »** : utilisé pour l'entraînement du modèle final. Il inclut des paquets qui n'existent pas dans le jeu de données « local ». Les tailles des paquets vont de 0 à 1536 octets et suivent une distribution uniforme.

L'avantage d'utiliser différents ensembles de données est d'éviter que la couche finale du modèle ne se concentre inéquitablement sur les modèles d'entrée. Un autre intérêt est de permettre au modèle d'intégrer le manque de performance du modèle ResNet18 sur les petits paquets. Il ne doit pas se concentrer entièrement sur le modèle intégrant les noms de serveurs. En effet, le modèle peut parfois être moins performant sur les gros paquets par rapport au modèle ResNet18, et le modèle final devrait également intégrer le manque d'informations pour certains paquets. Certains paquets n'ont pas de noms de serveurs associés ou pas de protocole identifié. Les ensembles de données « local » et « global » sont également conçus pour maximiser cette diversité. Chaque ensemble de données est divisé en trois sous-ensembles (80% entraînement, 10% validation et 10% test). Comme les classes sont parfaitement équilibrées, la fonction de perte catégorielle à entropie croisée (CCE) est utilisée.

2.5.4 Résultats

Les résultats des modèles basés sur la combinaison des informations sur les données test issus du « jeu de données de référence » et de « notre jeu de données » sur les applications connues (car appris durant l'entraînement) et inconnues sont listés dans le Tableau 2.6. La prise en compte de toutes les informations supplémentaires permet d'améliorer la précision de 5.1% à 14.8% sur « notre jeu de données » du réseau TLS [7] selon le Tableau 2.4 et de réaliser des performances à l'état de l'art en raison du manque

de généralisation des autres méthodes, comme le montre la section 2.4.2.

Le faible nombre de noms de domaines (2000 noms de serveurs différents) a un impact sur la généralisation des modèles. Les applications qui ne sont pas présentes dans les données d'entraînement sont moins bien classées lorsque les noms de serveurs sont présents. La taille des paquets n'a aucune influence sur les performances de classification. En revanche, les petits paquets sont mieux classés avec ces informations supplémentaires.

Tableau 2.6 Accuray obtenue après la combinaison de plusieurs sources d'information dans SPPNet.

Combinaisons	**Test**	**Notre**	**Applications connues**	**Applications inconnues**
Packet/Nom de serveur	0.670	0.553	0.854	0.211
Packet/Port	0.589	0.596	0.675	**0.581**
Packet/Protocol	0.356	0.310	0.318	0.344
Packet/Port/Protocol	0.638	0.622	0.761	0.570
Packet/Port/Protocol/ Nom de serveur	**0.857**	**0.650**	**0.880**	0.484
Packet/Protocol/ Nom de serveur	0.720	0.539	0.875	0.177

Cependant, ces performances sont inférieures d'environ 10% sur les données de test issus du « jeu de données de référence » (colonne « Test ») par rapport aux approches de l'état de l'art basées sur les statistiques de flux [115] mais ces approches ne peuvent pas classifier « en direct » le trafic réseau. De plus, l'évaluation des modèles basés sur les statistiques de flux n'est pas effectuée sur deux ensembles de données différents, il n'y a pas de garantie de généralisation. Néanmoins, l'ajout d'informations sur les statistiques de flux auxquelles le paquet appartient en entrée de SPPNet pourrait augmenter les performances de classification.

Afin de prouver la validité de notre approche, nous avons implanté SPPNet sous la forme d'une preuve de concept et nous avons classifié du trafic « en direct ».

2.5.5 Preuve de concept

Pour la preuve de concept, SPPNet est entraîné en utilisant un total de 237 636 paquets provenant des ensembles de données de référence et du nôtre. L'apprentissage de l'architecture est effectué sur Osirim [81], la plateforme de calcul de l'IRIT, en utilisant un GPU Nvidia GeForce GTX 1080Ti. Il est implémenté sur un ordinateur portable avec Python 3. Le code source est disponible [1]. L'outil de visualisation en Javascript et le

1. Code source : `https://github.com/fmeslet/SPPNet`

diagramme d'interaction entre le serveur et le modèle pour l'inférence sont présentés dans la Figure 2.6.

FIGURE 2.6 Interface graphique (à gauche) et diagramme d'interaction (à droite).

La mise en œuvre du modèle dans une situation réelle soulève les questions techniques suivantes :

— **Cohérence dans la classification des paquets d'un même flux** : il doit être possible de classer les paquets d'un même flux avec une probabilité similaire pour chaque classe.

— **La gestion des petits paquets d'accusé de réception TCP** : certains paquets ne servent qu'à acquitter la bonne réception des données mais ne contiennent pas d'information exploitable pour la classification.

Une table de hachage a été implantée. Chaque entrée de cette table identifie un flux à partir du tuple : IP source, IP destination, port source, port destination. A chaque entrée est associée une probabilité pour chaque classe. La probabilité pour chaque classe d'un paquet est ajoutée à la table et moyennée avec les prédictions précédentes associées aux flux. La classification devient « avec état » car la notion de flux est prise en compte pour la classification.

Dans le cas de la classification des petits paquets d'accusé de réception, nous effectuons d'abord une recherche dans la table de hachage. Si le flux existe, nous prenons la probabilité du flux déjà calculée. En revanche, si le flux n'existe pas, il est classé. En raison de la petite taille du paquet le risque d'erreur étant élevé, ce flux n'est pas ajouté à la table, ni la probabilité du paquet en sortie du modèle. Les paquets qui n'ont pas de lien direct avec les classes d'application (paquet Address Resolution Protocol (ARP), DNS, ...) ne sont pas classés et sont ignorés.

2.5.6 Performances

Cette section résume les performances et l'infrastructure utilisée pour la génération et l'apprentissage des différents éléments de SPPNet. Les mesures sont effectuées sur l'architecture Osirim [81], la plateforme de calcul de l'IRIT, en utilisant un GPU Nvidia GeForce GTX 1080Ti.

Les données d'apprentissages du « jeu de données de référence » comprennent les données d'entraînements (80%) et validation (10%) sont utilisées pour nos mesures. Le RNN, appartenant à la première partie de SPPNet, met 5 minutes et 15 secondes pour être appris sur 72 922 paquets. Le modèle ResNet18 met 9 minutes pour être appris sur 72 922 paquets en représentation 1D au format entier.

Le temps d'inférence et d'apprentissage pour les autres éléments de l'architecture peut être considéré comme négligeable (quelques secondes). L'apprentissage de la seconde partie de l'architecture est considéré comme négligeable (quelques secondes) car elle se compose de deux couches de neurones complètement connectées. Pour rappel, durant l'apprentissage de la seconde partie de SPPNet les paramètres des autres éléments de l'architecture restent fixes.

2.6 Étude de l'impact du chiffrement sur la classification

Dans cette section deux expérimentations seront menées sur le jeu de données « non chiffré ». Dans un premier temps, l'objectif sera de comprendre la capacité des réseaux de neurones à trouver des informations de structure à travers le chiffrement. Dans un second temps, une ouverture sera proposée sur l'utilisation des statistiques pour la classification de trafic réseau chiffré en presque en « direct ».

2.6.1 Étude de l'impact du chiffrement

L'objectif de cette expérimentation est de déterminer la capacité des réseaux de neurones à distinguer des informations de structure à travers le trafic réseaux chiffrés. Nous avons montré que les informations contenues dans l'en-tête ont un impact important sur la classification. Après leur suppression, nous constatons des performances légèrement supérieures à l'aléatoire visible dans le Tableau 2.4 pour les données TLS [7]. L'étape d'interprétation nous a permis de voir que le modèle se focalisait sur des informations contenues dans le paquet. Nous fondons deux hypothèses :

— **Hypothèse n°1 :** le modèle arrive à distinguer des motifs associés au type de chiffrement utilisé. Le type de chiffrement utilisé donne une information sur

l'application et donc sur la classe associée au paquet. Par exemple, le chiffrement Advanced Encryption Standard (AES) en mode GCM, est utilisé par l'application Zoom. Le mode varie donc suivant l'application et donne ainsi une information relative au contenu du paquet.

— **Hypothèse n°2 :** le modèle arrive à percevoir la taille du paquet ce qui lui offre une information sur le type de classe. Les paquets subissent un « padding » pour uniformiser leurs tailles formant une bande noire visible sur la Figure 2.1. La taille de cette bande peut être apprise par le modèle ce qui donne comme information la taille du paquet.

Selon nous, il n'est pas possible de percevoir d'autres informations à travers le chiffrement sinon cela signifierait que le chiffrement est inefficace.

Cette évaluation sera effectuée sur le jeu de données « non chiffré » sur lequel un chiffrement identique sera appliqué. L'objectif sera de comparer les performances de classification des données réseaux « non chiffrées » avec celles étudiées précédemment. Les données sont mises sous la forme d'un vecteur à une dimension avec des valeurs entre 0 et 255. Les valeurs sont ensuite normalisées entre 0 et 1.

Nous avons ensuite appliqué un chiffrement AES en mode CBC avec une unique clé et un unique vecteur d'initialisation. Il n'est donc pas possible au modèle de percevoir des informations liées au chiffrement. La matrice de confusion obtenue est présentée dans le Tableau 2.7.

Tableau 2.7 Matrice de confusion pour la classification du jeu de données « réseau » chiffré avec AES CBC.

		Prédiction				
		HTTP	Mail	Chat	Streaming	VoIP
Réalité	HTTP	**0.908**	0.036	0.054	0	0.002
	Mail	0.065	**0.921**	0.014	0	0
	Chat	0.129	0.068	**0.756**	0.004	0.043
	Streaming	0	0	0	**1.0**	0
	VoIP	0	0.08	0.03	0	**0.989**

La précision de la classification est de 91%. Le réseau de neurones arrive à classifier le trafic malgré le chiffrement. La Figure 2.7 illustre la différence de « padding » qui existe entre deux paquets de classes différentes, ici streaming et VoIP.

Les différences de taille qui peuvent exister entre les différents paquets de chaque classe vont influencer la taille du « padding » utilisée. Le réseau de neurones se focalise

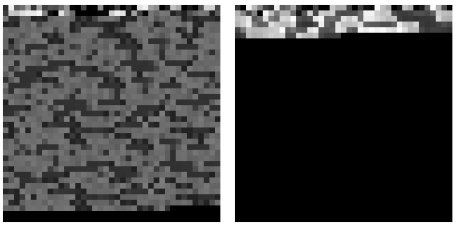

FIGURE 2.7 Paquet extrait du jeu de données « réseau » avant chiffrement. A gauche un paquet issu de la classe HTTP et à droite un paquet issu de la classe VoIP.

sur la partie du paquet dont les bits ne sont pas à zéro et ainsi, il peut en déduire la taille du paquet comme observé lors de l'application des « Occlusion Map » [8]. La seconde hypothèse est donc validée. Ce constat permet d'expliquer deux choses. Premièrement, les mauvaises performances du réseau de neurones sur les données provenant du réseau TOR proviennent de l'uniformisation du réseau effectuée sur les tailles de paquet. Deuxièmement, les mauvaises classifications sont liées à des similarités de taille de paquet qui peuvent exister. Par exemple, dans le cas de la classe « Chat », les tailles de paquet varient fortement en fonction de la taille des messages véhiculés. Ainsi, certains paquets peuvent être assimilés à la classe HTTP lorsqu'il atteint une taille élevée.

Ce qui permet au réseau de neurones d'effectuer une classification performante provient de variabilité statistique telle que la taille des paquets perçu au travers du « padding ». Ce constat nous pousse à nous demander comment intégrer cette information, sans biais, dans le cadre d'une classification « en direct ». Nous pensons que l'utilisation de statistiques peut être une bonne solution.

2.6.2 Études de l'impact des statistiques

L'étude de l'impact des statistiques s'intéresse à extraire un ensemble de statistiques sur un flux afin d'effectuer la classification. Cette approche possède deux limites :

— Il n'est pas possible d'effectuer de la classification « en direct ». Il est nécessaire de « bufferiser » un certain nombre de paquets afin de prendre la décision de classification. Les flux de trop petite taille ne pourront alors pas être classifiés.
— Les statistiques peuvent être fortement influencées par les caractéristiques du réseau sur lequel la capture des données a été effectuée.

Les données utilisées proviennent du jeu de données « non chiffrés ». 3325 flux ont été extraits avec une répartition équitable entre les classes. Les 20 premiers paquets du flux ont été utilisés pour effectuer la classification. Du « padding » a été utilisé pour pouvoir

exploiter les flux de taille inférieure. Ce choix est effectué en se basant sur les travaux effectués dans la classification de flux réseau basé sur les statistiques [115, 116, 118]. Le Tableau 2.8 présente le format utilisé en entrée du modèle pour l'apprentissage du réseau de neurones.

TABLEAU 2.8 Format des caractéristiques extraites pour l'apprentissage du réseau de neurones.

Timestamps	0	0.1	0.9	1.7	3.8	...	10
Tailles des paquet	235	728	1488	200	200	...	1488

La classification permet une précision de 99.7%. Pour rappel, les classes sont équilibrées, l'ensemble des classes est correctement classifié. Cependant, ces bonnes performances ne sont pas généralisables. Elles s'expliquent par les biais présents dans notre jeu de données collecté. En effet, notre jeu de données « non chiffré » a été collecté d'une unique façon pour chaque classe (même réseau, mêmes serveurs, mêmes applications). Ce manque de diversité fait que tous les paquets de la classe « Streaming » ont la même taille. C'est un biais qui existe dans certains travaux comme [115]. Ainsi, un modèle entraîné sur un type de réseau avec une configuration particulière ne pourrait pas classifier le trafic réseau sur une autre architecture avec une autre configuration.

Pour faire face à ces biais, nous proposons une méthode permettant de rendre généralisable l'exploitation des statistiques. Pour cela, nous utilisons des techniques de normalisation et d'uniformisation. Dans un premier temps, nous prenons un nombre de mesures fixe, équitablement réparties entre l'identifiant de temps du premier paquet du flux reçu et celui du dernier paquet du flux. Ensuite, nous collectons la somme des octets transmis entre chaque instant du flux et nous normalisons la somme entre 0 et 1. Ainsi, la variabilité propre au réseau n'est pas capturée, car l'instant d'arrivée n'est pas pris en compte, seul l'instant d'arrivée du premier et du dernier paquet est utilisé ce qui réduit l'impact de la variabilité. De plus, la somme des octets reçus est normalisée à l'échelle du flux. Il n'y a donc pas de risque d'être influencé par une différence de taille entre deux classes qui peut être propre au réseau. La Figure 2.8 montre l'espace latent obtenu après projection des données avec un β- VAE [10], dont l'architecture est présentée lors du chapitre 1. L'architecture du β- VAE [10] utilisé pour la projection est présentée dans le Tableau 2.9 avec $\beta = 0.01$.

Les caractéristiques statistiques projetées montrent une séparation entre les différentes classes. Leur chevauchement est cohérent, nous retrouvons aux extrémités les classes de trafic les plus antagonistes : VoIP et Navigation Internet. Le trafic VoIP est caractérisé par un nombre important de paquets de petite taille échangés avec un faible délai. À

TABLEAU 2.9 Architecture du β- VAE [10] utilisé pour la projection des statistiques.

Encoder	**Paramètres**	**Dim. de sortie**
Input	0	20
Dense (LeakyReLU)	420	20
Dense (LeakyReLU)	210	10
Dense (LeakyReLU)	52	5
Dense (LeakyReLU) mean	12	2
Dense (LeakyReLU) var	12	2
Decoder	**Paramètres**	**Dim. de sortie**
Input	0	2
Dense (LeakyReLU)	15	5
Dense (LeakyReLU)	60	10
Dense (LeakyReLU)	220	20
Dense (Sigmoid)	420	20

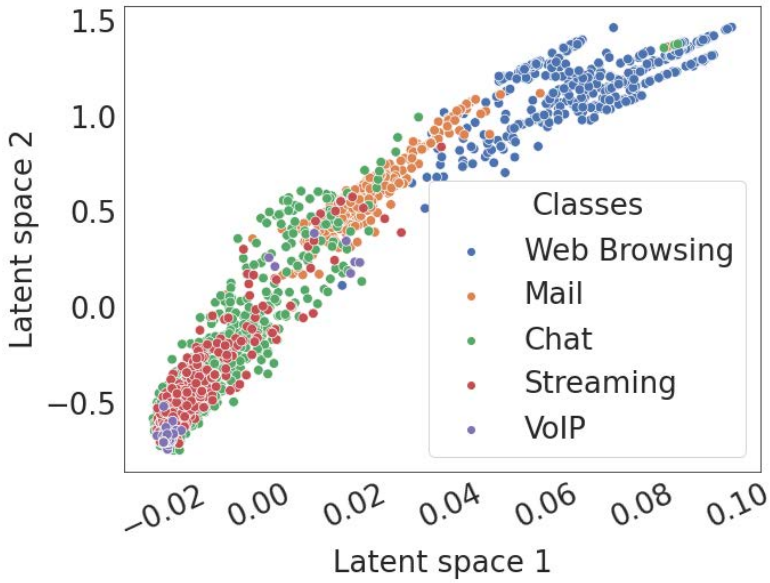

FIGURE 2.8 Espace latent obtenu après projection des statistiques extraites du jeu de données « réseau ».

l'inverse, le trafic lié à de la navigation internet est plus sporadique et il est véhiculé par des paquets de taille importante. Lors de la classification avec un CNN, nous obtenons 87% de précision. La matrice de confusion obtenue est présentée dans le Tableau 2.10.

En échantillonnant l'espace latent, nous pouvons à l'aide du décodeur du β- VAE [10] reconstruire une série statistique propre à chaque classe. Cette possibilité ouvre la voie à

Tableau 2.10 Matrice de confusion pour la classification du jeu de données « réseau » à partir des statistiques avec un traitement de normalisation et d'uniformisation.

		Prédiction				
		HTTP	Mail	Chat	Streaming	VoIP
Réalité	HTTP	**0.961**	0.039	0	0	0
	Mail	0.026	**0.947**	0.026	0	0
	Chat	0	0.026	**0.934**	0.039	0
	Streaming	0	0.013	0.132	**0.526**	0.329
	VoIP	0	0	0	0	**1.0**

l'augmentation de données pour l'apprentissage profond ou la génération de trafic.

2.7 Conclusion

Dans ce chapitre, nous avons étudié les architectures d'apprentissage profond pour la classification sans état du trafic chiffré. Nous avons montré que, sans aucun pré-traitement, les modèles peuvent se concentrer sur des caractéristiques qui ne peuvent pas être généralisées pour la classification basée sur les statistiques ou les paquets. Pour surmonter cette limitation, nous avons d'abord traité séparément les en-têtes et les données. Ensuite, nous avons défini les différentes sources d'information et nous les avons étudiées indépendamment afin d'obtenir de bonnes performances de généralisation. Enfin, nous avons proposé un pré-traitement permettant l'intégration des statistiques sans biais.

Nous avons proposé une nouvelle architecture modulaire, appelée SPPNet, qui implémente cette approche et permet, en utilisant la classification au niveau des paquets, une classification « en direct ». Son implantation, combinée avec une table de flux, a permis d'obtenir de meilleures performances en prenant en compte la notion de flux. Cette approche offre une modularité en définissant de nouvelles sources d'information comme le montre notre preuve de concept. Une version avec état de SPPNet peut maintenant être considérée. Des statistiques telles que les corrélations entre les paquets pourraient permettre d'augmenter les performances.

Cependant, cette vision nécessite de prendre en considération la variété des architectures et typologie de réseau existant qui peuvent modifier de façon significative les statistiques pour chaque classe de données. Pour remédier à ce problème, nous proposons un ensemble de traitements de normalisation permettant d'effacer les spécificités qui pourraient être causées par la source des données utilisées. La projection nous montre

qu'avec cette uniformisation certaines classes de trafic reste difficile à séparer tels que VoIP et Streaming. Une autre possibilité pourrait être l'augmentation des bases de données d'apprentissage, de façon artificielle, afin d'améliorer les performances des classifieurs face à la diversité de réseau existant. Nous proposons d'explorer cette voie en travaillant sur la génération de trafic réseau par apprentissage profond.

Dans le prochain chapitre, nous allons définir une nouvelle approche permettant la génération de trafic réseau sans connaissance sur la structure des protocoles ou des caractéristiques de trafic.

Chapitre 3

Génération de trafic réseau universelle

Sommaire

3.1 Introduction

Dans ce chapitre, nous étudions la génération de trafic réseau. Nous montrons les limites des approches actuelles et nous proposons une alternative de la génération de trafic réseau universelle par apprentissage profond.

La génération de trafic est un outil important pour de nombreuses activités liées aux réseaux de données, telles que la simulation, la planification et le provisionnement [11, 12]. Une telle génération pourrait être fondée sur des modèles théoriques lorsqu'ils sont disponibles ; cependant, nous pensons que l'imitation du trafic précédemment collecté est une solution beaucoup plus générique. Elle fournit un outil complet permettant de prendre en compte les diverses caractéristiques des systèmes de communication actuels ou futurs (IoT, modèles de services 5G, ...). Ainsi, la génération peut également être utilisée comme un moyen d'amélioration de généralisation des classifieurs réseau, par apprentissage profond, en leur offrant des données synthétiques plus hétérogènes lors de l'apprentissage.

Aujourd'hui, les générateurs traditionnels [11, 12], étudiés dans la section 3.2.1, sont confrontés aux problèmes suivants :

— Leur adaptabilité est limitée. Ils ne sont pas capables de travailler sur tous les types de protocole.

— Ils ne sont pas en mesure de générer des informations supplémentaires en conjonction avec le trafic telles que : le Signal-to-Noise Ratio (SNR), la fréquence utilisée.

Afin de surmonter les limitations précédentes, nous introduisons une nouvelle approche appelée NeCSTGen, fondée sur des modèles d'apprentissage profond tels que : VAE et RNN pour la génération de divers trafics réseau. Notre objectif est de définir un outil polyvalent capable de produire tout type de trafic réseau ; à l'aide de techniques d'apprentissage profond, NeCSTGen peut fonctionner sans aucune connaissance du modèle de trafic. Quelles que soient les données de trafic réseau que nous collectons, nous pouvons générer un trafic qui présente les mêmes caractéristiques statistiques sans avoir besoin de comprendre, en profondeur, le fonctionnement du protocole à générer. NeCSTGen ne produit pas lui-même les données de niveau utilisateur, mais peut utiliser la sortie de tout autre outil dédié.

Pour démontrer son adaptabilité, nous testons notre approche sur différents ensembles de données provenant de divers réseaux. Nous fournissons également le code source [1] pour une recherche reproductible et nous montrons qu'il est possible de générer des fichiers « .pcap » réalistes qui peuvent être exploités pour différents cadres d'application.

1. Code source : `https://github.com/fmeslet/NeCSTGen`

Dans une première partie, nous présentons les différents travaux qui existent dans le domaine de la génération de trafic réseau. Dans une seconde partie, nous formalisons notre approche et nous présentons les données utilisées pour la génération de trafic réseau. Dans une troisième partie, nous présentons et discutons de l'architecture développée nommée NeCSTGen et des différents problèmes qu'il vise à résoudre. NeCSTGen est une approche adaptative qui nous permet de travailler avec tous les types de trafic, sans connaissance préalable du trafic à générer. De plus, nous pouvons générer de longues séquences de trafic cohérent. Dans la quatrième et dernière partie, nous présentons et comparons les performances de NeCSTGen avec les données originales.

3.2 État de l'art

Les générateurs de trafic réseau sont des outils informatiques utilisés pour simuler la transmission de données à travers un réseau de communication. Ils permettent de tester la performance, la sécurité et la capacité d'un réseau en envoyant des paquets de données pour simuler différents types de trafic. Ils sont essentiels pour les administrateurs de réseau qui cherchent à optimiser les performances et à garantir la sécurité de leur réseau [11, 12].

Nous proposons de classer les générateurs en deux catégories, les générateurs traditionnels et les générateurs par apprentissage profond. Les générateurs traditionnels sont les plus répondus. Face à certaines de leurs limites, des générateurs par apprentissage ont émergé.

Dans un premier temps, nous présentons les approches traditionnelles que nous classons en différentes catégories. Dans un second temps, nous présenterons et classons les approches par apprentissage profond. Pour finir, nous parlerons des limites des approches traditionnelles et par apprentissage profond.

3.2.1 Générateurs traditionnels

Synthèse

En s'inspirant de [12], nous pouvons classer les générateurs traditionnels en six catégories : les générateurs à débit maximum, les générateurs basés sur des scripts, les générateurs de haut niveau et auto-configurables, les générateurs de relecture et les générateurs basés sur des modèles. Pour chaque catégorie un tableau répertorie les différents générateurs existants jusqu'à aujourd'hui : les générateurs à débit maximum (Tableau 3.1), les générateurs basés sur des scripts (Tableau 3.2), les générateurs de haut

niveau et auto-configurables (Tableau 3.3), les générateurs de relecture (Tableau 3.5) et les générateurs basés sur des modèles (Tableau 3.6). Le fonctionnement de chacune de ces catégories de technique est détaillé dans les parties suivantes en reprenant la synthèse par [11].

Tableau 3.1 Générateurs à débit maximum utilisés pour la génération de trafic [11, 12].

Générateur de trafic	**Description**
Iperf [128]	Permet de tester la bande passante, le taux de perte de paquets et la gigue au niveau applicatif.
BRUTE [129]	Générateur de paquets au niveau du noyau.
Pacgen [130]	Générateur de paquets Ethernet, IP, TCP/UDP. Il permet de créer des paquets personnalisés avec une charge utile personnalisée.
nuttcp [131]	Détermine le débit brut de la couche réseau TCP/UDP en transférant des tampons de mémoire d'un système source à travers un réseau d'interconnexion.
BRUNO [132]	Générateur de paquets implémenté au niveau matériel.
KUTE [133]	Générateur de paquets au niveau du noyau.
nping [134]	Outil open source pour la génération de paquets réseau varié, l'analyse des réponses et la mesure des temps de réponse.
Spirent SmartBits [135]	Matériel permettant de tester, simuler, analyser, dépanner, développer et certifier l'infrastructure d'un réseau.
IxChariot [136]	Outil commercial pour l'évaluation instantanée de la performance des réseaux complexes avant et après le déploiement.
UDP Generator [137]	Générateur de trafic UDP pour l'évaluation de performance.
Ixnetwork [138]	Générateur commercial d'évaluation de performance de niveau L2-3 qui s'adapte aux besoins de l'entreprise.
Colosoft Packet Builder [139]	Permet de créer des paquets réseau personnalisés ; les utilisateurs peuvent utiliser cet outil pour vérifier la protection de leur réseau contre les attaques et les intrusions.
IP-Packet [140]	Ensemble d'outils permettant d'inspecter et de détourner les connexions réseau.
TTCP [141]	Mesure le débit d'un trafic TCP sur un chemin IP.
LANTraffic [142]	Logiciel de génération de trafic pour les plateformes Windows de 10Mbps à 10Gbps (performance maximale avec les plateformes x64) utilisant différents protocoles (UDP, TCP, Internet Control Message Protocol (ICMP),).
IPGen [143]	Générateur de paquets IP, ICMP, TCP, UDP avec une fausse IP source.
HexInject [144]	Injecteur et renifleur de paquets réseaux polyvalents.

TABLEAU 3.2 Générateurs basés sur des scripts utilisés pour la génération de trafic [11, 12].

Générateur de trafic	**Description**
MoonGen [145]	Générateur de paquets à grande vitesse scriptable construit avec des scripts Lua.
OSNT [146]	Générateur matériel open-source basé sur la plateforme NetFPGA.
Scapy [125]	Générateur de trafic sous Python.
Linux pktgen [147]	Générateur de trafic implémenté comme un module Linux. Les paramètres sont l'équipement de sortie, le délai, le nombre de paquets.
Libtins [148]	Bibliothèque C++ multiplateforme de collecte et d'élaboration de paquets réseau haut niveau.
EPB [149]	Outil permettant d'envoyer des paquets Ethernet personnalisés sans interface graphique, avec peu de bibliothèques nécessaires.
DPDK pktgen [150]	Générateur de trafic alimenté par le cadre de traitement rapide des paquets nommé DPDK.
TRex [151]	Générateur de trafic open source, à faible coût, avec ou sans état, alimenté par DPDK.
Ostinato [152]	Générateur de paquets au niveau de l'utilisateur avec une interface.
Nemesis [153]	Utilitaire en ligne de commande pour l'élaboration et l'injection de paquets réseau.
WRAP [154]	Générateur de trafic configurable, avec état, niveau L1 à L7, pour générer de grands volumes de trafic basé sur des sessions.

Tableau 3.3 Générateurs de haut niveau et auto-configurables utilisés pour la génération de trafic [11, 12].

Générateur de trafic	**Description**
HARPOON [155]	Générateur de trafic basé sur les flux qui peut imiter les mesures basées sur Netflow.
SWING [156]	Générateur de trafic en boucle fermée, sensible au réseau, capable d'extraire des distributions pour le comportement des utilisateurs, des applications et du réseau à partir de mesures réelles.
TMIX [157]	Émulateur de trafic pour ns-2 basé sur la caractérisation au niveau de la source des connexions TCP.
LiTGen [158]	Générateur de trafic en boucle ouverte, au niveau des paquets, basé sur une modélisation réaliste du trafic IP.
LANforge FIRE [159]	Générateur commercial pour l'émulation et le test d'équipements. Il prend en charge les connexions TCP avec état et peut générer une charge de trafic sur de nombreux composants du réseau.
Skaion TGS [160]	Générateur commercial extensible de niveau utilisateur capable de gérer de gros volumes de données.
Breaking-Point [161]	Générateur commercial pour des tests complets de performance et de sécurité d'équipements réelles ou virtuelles à l'échelle d'une entreprise.
Byte-Blower [162]	Générateur commercial pour le test de réseau (Wi-Fi, 5G) et d'équipements IP dans des conditions et des scénarios variés.
GlobeTraff [163]	Générateur de trafic basé sur les outils WebTraff/ProwGen.
GL traffic generator [164]	Générateur de trafic réseau et applicatif pour des mesures de performances.
NTG [165]	Émule les utilisateurs finaux d'un réseau en générant un véritable trafic réseau réel basé sur des scénarios.
D-ITG [166]	Cadre étendu de génération de charges de travail pouvant produire du trafic pour un large éventail de scénarios de réseau

TABLEAU 3.4 Générateurs de relecture utilisés pour la génération de trafic [11, 12].

Générateur de trafic	**Description**
TCPivo [167]	Moteur de relecture à haute vitesse au niveau du noyau.
Bit-Twist [168]	Générateur de paquets Ethernet basé sur libpcap.
Trafgen [169]	Générateur de trafic réseau multi-thread.
EAR Replay [170]	Générateur de trafic réseau local sans fil avec émulation de l'environnement.
Fragroute [171]	Intercepte, modifie et réécrit le trafic de sortie destiné à un hôte spécifié. L'outil aide à tester les systèmes de détection d'intrusion dans les réseaux, les pare-feu.
PlayCap [172]	Rejoue les captures Wireshark, tcpdump et libpcap.
Divide and Conquer [173]	Technique de relecture pour les traces d'équipements OC-48 utilisant plusieurs interfaces Gigabit Ethernet.

TABLEAU 3.5 Générateurs de scénarios spéciaux utilisés pour la génération de trafic [11, 12].

Générateur de trafic	**Description**
ParaSynTG [174]	Générateur de trafic web.
VoIP TG [175]	Le générateur crée plusieurs flux de trafic simulant un flux VoIP, sans établissement d'appel, juste du trafic, et ne prend actuellement en charge que le trafic unidirectionnel et les flux à débit constant.
Yersinia [176]	Générateur de trafic pour des attaques réseaux Spanning Tree.
httperf [177]	Générateur de divers trafic HTTP.
Surge [178]	Génère des requêtes Web destinées à imiter les propriétés statistiques mesurées.
YouTube Workload Generator [179]	Méthodes de génération de charge de travail pour le trafic vidéo imité de YouTube
Graph-Based Traffic Generator [180]	Générateur de traces de flux basé sur des modèles de graphes de dispersion du trafic

Tableau 3.6 Générateurs basés sur des modèles utilisés pour la génération de trafic [11, 12].

Générateur de trafic	**Description**
TG [181]	Générateur de niveau paquets utilisant diverses distributions de temps inter-paquet et de taille de paquet.
Packet Sender [182]	Utilitaire open source permettant l'envoi et la réception de paquets TCP, UDP et Secure Socket Layer (SSL) ainsi que la génération de requêtes HTTP/Hypertext Transfer Protocol Secure (HTTPS).
PackETH [183]	Permet de créer et d'envoyer n'importe quel paquet ou séquence de paquets sur une liaison Ethernet.
Mausezhan [184]	Générateur de trafic permettant d'évaluer les communications VoIP ou réseau multicast.
MxTraf [185]	Générateur de trafic réseau où un petit nombre d'hôtes peut être utilisé pour saturer un réseau avec un mélange de trafic réglable.
Solarwind WAN killer [186]	Générateur de trafic réseau commercial pour l'évaluation de performance réseau.
Omnicor TG [187]	Générateur commercial utilisé pour tester la performance des réseaux dans des conditions de débit maximal ou partiel.
Self Similar TG [188]	Le trafic auto-similaire est obtenu par l'agrégation de plusieurs sous-flux, chacun consistant en une alternance de périodes de marche et d'arrêt distribuées selon la loi de Pareto.
Sources-OnOff [189]	Générateur de trafic basé sur des distributions variées à paramétrer.
Traffic Generator Tool [190]	Génère et reçoit des flux de paquets unidirectionnels niveau utilisateur UNIX entre des sources et des destinations dans un réseau.
STG-10G [191]	Générateur commercial de trafic UDP ou TCP via plusieurs interfaces IP avec états.
Jugi's TG [192]	Utilisé pour envoyer du trafic IP, des flux UDP, TCP en une fois, à un récepteur.
MGEN [193]	Générateur permettant d'effectuer des tests et des mesures de performance du réseau IP en utilisant le trafic TCP et UDP.

Générateurs à débit maximum

Les générateurs de trafic réseau à débit maximal sont utilisés pour tester les performances du réseau, de bout en bout, en envoyant de manière répétée des paquets à un débit constant ou maximal possible en bits par seconde ou en paquets par seconde. Les générateurs de cette catégorie, tels que iperf2 [128], offrent peu de variations dans le contenu de l'en-tête et de la charge utile des paquets. Ces générateurs conviennent pour tester rapidement les performances d'un réseau.

Des outils comme BRUTE [129], BRUNO [132], KUTE [133] et Ostinato [152] sont largement utilisés dans l'ingénierie des réseaux pour tester les caractéristiques de la bande passante, du délai, de la gigue et du taux de perte. BRUTE [129] et KUTE [133] sont des outils de génération de trafic au niveau des paquets qui fonctionnent au niveau du noyau Linux, garantissant ainsi un comportement plus contrôlable. BRUNO [132] est une extension de la même méthodologie à une plateforme matérielle spécifique, fournissant des valeurs plus précises à la fois en termes de débit et de temps inter-paquet. L'évaluation des performances de KUTE [133] montre qu'il se rapproche mieux de la valeur attendue que les autres outils de génération de trafic. Ostinato [152] est un outil très récent de génération de trafic au niveau de l'utilisateur qui offre une interface graphique conviviale pour définir divers flux de trafic.

Générateurs basés sur des scripts

Ces générateurs permettent aux utilisateurs de modifier dynamiquement l'ensemble du contenu de l'en-tête et des données des paquets par le biais d'une logique codée complexe. Les exemples populaires de générateurs de cette catégorie sont : DPDK pktgen [150] et moongen [145]. Ils permettent aux utilisateurs de créer n'importe quel type de paquet, avec presque n'importe quelle valeur d'en-tête de paquet, tout en modifiant dynamiquement les paquets au moment de l'exécution. Ils permettent la création de scénario spécifique dans l'objectif d'évaluer la performance d'un réseau.

Générateurs de haut niveau et auto-configurables

Les générateurs de trafic réseau de haut niveau et auto-configurables sont conçus pour un type spécifique d'application ou de protocole de couche supérieure. Ces générateurs, tels que HARPOON [155] et SWING [156] sont fondés sur un modèle de niveau supérieur du trafic réseau et sont capables de produire un trafic artificiel dont les caractéristiques sont proches de la mesure originale. Ils peuvent analyser des mesures réelles pour extraire des valeurs telles que les caractéristiques des flux, les caractéristiques des sessions et des

objets, ainsi que les séquences de temps inter-paquets et de taille des paquets.

Bien que ces générateurs soient capables de variations réalistes pour la génération de paquets d'applications ou de protocoles spécifiques, la charge de travail résultante consiste toujours en un ensemble limité d'applications ou de protocole. La validation de ces générateurs implique la comparaison de traces de trafic réelles et synthétiques à l'aide d'un large éventail de mesures telles que le débit d'octets, la distribution du temps d'interconnexion, la distribution de la taille des fichiers, la distribution de la fréquence des adresses IP et la distribution du volume d'octets, de paquets et de flux. Toutefois, les chiffres présentés dans ces études manquent souvent d'une comparaison quantitative avec les métriques utilisées.

Générateurs de relecture

Les moteurs de relecture sont conçus pour injecter des paquets provenant d'un fichier de trace préexistant dans une interface réseau aux intervalles de temps indiqués dans le fichier de capture. Parmi ces méthodes, nous pouvons citer TCPivo [167], entre autres. Ces systèmes de relecture peuvent produire des charges de travail qui reflètent le trafic original, mais ils sont généralement sans état et ne peuvent pas s'adapter à la congestion du réseau ou aux événements topologiques (coupure de lien réseau par exemple) au cours d'une expérience.

Par conséquent, la relecture continue du même fichier de trace peut donner lieu à des schémas de trafic irréalistes. En effet, une telle relecture continuera à envoyer des paquets même lorsque les liaisons entre les points d'extrémités sont interrompues, alors qu'un contrôle de flux TCP réaliste aurait limité les transmissions de paquets supplémentaires. Pour résoudre ce problème, les chercheurs ont mis au point des techniques permettant de rejouer les traces capturées sur plusieurs équipements dotés d'interfaces Gigabit Ethernet afin d'augmenter la bande passante disponible [173]. Cependant, même avec ces améliorations, il est encore difficile de reproduire avec précision les schémas de trafic originaux, en particulier en ce qui concerne l'asymétrie du temps de transmission inter-paquet due à des interruptions logicielles imprécises et à la saturation du débit due aux limitations de la bande passante.

Générateurs de scénarios spéciaux

Les générateurs de trafic réseaux spéciaux sont conçus pour reproduire des conditions de réseau spécifiques et, par conséquent, ils utilisent souvent des mesures uniques adaptées à leur scénario donné. Des exemples de générateurs de trafic spéciaux sont : ParaSynTG [174]

et YouTube Workload Generator [179], où des méthodes de génération sont présentées pour des types de trafic spécifiques tels que la navigation internet et le streaming YouTube. Dans [174], l'outil ParaSynTG est validé à l'aide de mesures spécifiques au web telles que la fréquence des requêtes et la distribution de la taille des documents. Dans le cas du trafic YouTube [179], les auteurs se concentrent sur les caractéristiques basées sur le cache du proxy.

Générateurs basés sur des modèles

Les générateurs de trafic basés sur des modèles utilisent des modèles stochastiques pour générer des traces au niveau des paquets qui peuvent imiter un trafic réseau réaliste. MultiGenerator (MGEN) [193] est un exemple de générateur de trafic populaire qui créé et transmet des paquets suivant des distributions aléatoires de leurs intervalles de temps, de la taille des paquets, etc. Ces générateurs permettent aux utilisateurs de spécifier un modèle de distribution aléatoire avec des paramètres qui correspondent au scénario prévu de la charge de travail du trafic réseau. Grâce à des distributions aléatoires soigneusement sélectionnées, ils peuvent générer un trafic statistiquement similaire aux charges de travail du trafic dans des environnements de production spécifiques.

Toutefois, nous pouvons nous demander si le trafic généré par les générateurs de trafic basés sur des modèles suit les statistiques établies par le modèle stochastique. L'étude [194] a montré que les statistiques de sortie de certains générateurs de trafic basés sur un modèle ne correspondent pas à ce que les résultats analytiques suggèrent. Il est donc important de tester ces générateurs pour diverses statistiques, y compris le temps inter-paquet, la distribution de la taille des paquets et la corrélation, que le modèle stochastique impliquerait. Cela permet de s'assurer que le trafic généré est réaliste et précis pour un scénario donné.

3.2.2 Générateurs par apprentissage profond

Synthèse

Face aux limites des générateurs traditionnels, des méthodes d'apprentissage profond ont été utilisées. Parmi les limites existantes, nous pouvons citer le manque d'adaptabilité de ces outils, spécialiser à une dynamique de protocole spécifique, entre autres. Nous reviendrons sur ces limites dans la partie 3.2.3. Les méthodes par apprentissage profond fonctionnent mieux que les approches traditionnelles pour l'estimation statistique non paramétrique grâce à l'utilisation des Generative Adversarial Network (GAN), entre autres [195]. Nous pouvons les repartir en deux catégories : les générateurs de statistiques

et les générateurs de paquets. Les Tableaux 3.7 et 3.8 résument l'ensemble des approches, par apprentissage profond, utilisées pour la génération de trafic réseau.

Tableau 3.7 Générateurs de statistiques utilisés pour la génération de trafic par apprentissage profond.

Générateur de trafic	**Description**
GAN pour l'IoT [196]	Utilise les GAN pour la génération de taille de paquet, de débit pour les NIDS.
Générateur de flux [197]	Utilise les GAN pour la génération de flux individuel et un traitement spécifique pour les variables catégorielles lors de la génération (adresses IP, ...).
Zen [198]	Génération de trafic cellulaire spatio-temporel avec interactions avec des LSTM.
GAN pour les données chiffrées [118]	Utilise une approche non supervisée via un « Auxiliary Classifier GAN » pour résoudre le problème des classes mal réparties pour la classification de traffic.
PacketCGAN [199]	Utilise des « Conditional GAN » pour régler le problème des classes mal réparties pour l'amélioration des classifieurs de trafic réseau chiffré.
Évaluation de GAN [200]	Évalue un modèle « B-WGAN-GP » pour la génération d'enregistrements de trafic NetFlow en utilisant plusieurs classifieurs.
Détection d'attaque [201]	Propose un nouveau modèle hybride de sur-échantillonnage utilisant un GAN pour améliorer les performances de détection des attaques dans les NIDS basés sur les anomalies.
Augmentation des données de flux [202]	Génère des statistiques de flux avec un LSTM permettant d'améliorer la performance des classifieurs réseau.
ZipNet-GAN [203]	Utilise des architectures GAN adaptées à la super-résolution du trafic mobile (MTSR) pour déduire des modèles de trafic mobile finement localisés collectés à partir de mesures de données grossières agrégées par un nombre limité de sondes de réseau avec une granularité arbitraire.

Tableau 3.8 Générateurs de paquets utilisés pour la génération de trafic par apprentissage profond.

Générateur de trafic	**Description**
AttackGAN [204]	Effectue de la modification de paquets en se basant sur la structure d'un « SeqGAN » pour contourner les systèmes de détection des NIDS.
ITCGAN [205]	Inspiré par les triple-GAN permet de faire face aux problèmes des classes mal distribuées pour la classification, entre autres.
PcapGAN [206]	S'intéresse à l'augmentation des données, issues de fichiers « .pcap », pour la détection d'intrusion.
PAC-GAN [207]	Générateur de paquets réseau avec des CNN et des GAN tel que des ICMP, requête DNS, requête HTTP pouvant être transmis sur le réseau et permettant d'obtenir une réponse.
Facebook Chat GAN [208]	Imiter le trafic du réseau de chat Facebook et modifier le comportement réseau de véritables logiciels malveillants en imitant le trafic d'utilisateurs légitimes tout en échappant à la détection.

Générateurs de statistiques et de paquets

Les générateurs de statistiques vont chercher à générer des statistiques associées à des distributions comme la taille des paquets, le temps inter-packets, etc. En revanche, les générateurs de paquets cherchent à regarder la structure d'un paquet afin de pouvoir l'exploiter directement sur le réseau [207].

Ces deux catégories permettent l'utilisation des générateurs de trafic réseau pour des cadres variés. Certaines approches cherchent à faire face au problème des classes mal distribuées pour l'apprentissage des classifieurs de trafic réseaux. Certaines architectures génèrent des statistiques [118, 196, 202]. D'autres utilisent les GANs pour la génération de paquets réalistes [199]. D'autres approches utilisent la génération de trafic de réseau pour l'évaluation ou l'amélioration des NIDS. Nous retrouvons l'utilisation des GANs pour l'estimation de distributions [196, 200, 201] ou pour la génération de paquets réalistes [199] et la modification de paquets [204]. Les GANs permettent l'estimation non-paramétrique de densité et donc une génération plus fine que les approches traditionnelles.

3.2.3 Limites des approches actuelles

Les générateurs traditionnels sont confrontés à plusieurs problèmes. Ils ne sont pas conçus pour travailler sur tout type de protocole. Leurs domaines d'applications sont limités à certains protocoles et il ne peuvent donc pas faire face à l'émergence des nouvelles générations de protocoles présents dans l'IoT, par exemple. De plus, ils ne sont pas en mesure de générer des informations supplémentaires en conjonction avec le trafic tel que : le SNR, la fréquence utilisée, etc.

Les approches par apprentissage profond font également face à de nombreuses limites. La génération de très longues séquences de paquets n'est pas possible avec les architectures d'apprentissage profond séquentiel actuelles de l'état de l'art telles que les architectures Transformer [209]. De plus, il faut connaître le fonctionnement détaillé du protocole afin de pouvoir estimer l'ensemble des paramètres (champs et statistiques) de manière cohérente lors de la génération. Par conséquent, aucune approche n'est capable de traiter différents types de protocoles ou de dynamiques sur les données à générer. Enfin, il n'y a pas de contrôle sur la génération : il n'est pas possible de générer des comportements spécifiques dans le réseau en suivant une approche multi-niveau comme c'est le cas pour HARPOON [155], une approche traditionnelle.

3.3 Modélisation

Dans cette section, nous définissons la structure globale du fonctionnement de l'étape d'apprentissage et de génération de notre approche. Nous expliquons en détails l'étape de sélection et d'extraction des caractéristiques utilisées en entrée de NeCSTGen. Nous présentons également la formalisation de notre approche ainsi que le jeu de données utilisé pour l'apprentissage et l'évaluation du modèle. La formalisation posera les bases du fonctionnement de notre approche NeCSTGen pour la génération de trafic réseau universelle.

3.3.1 Structure globale

Nous distinguons deux partie dans le fonctionnement de notre approche, l'apprentissage et la génération.

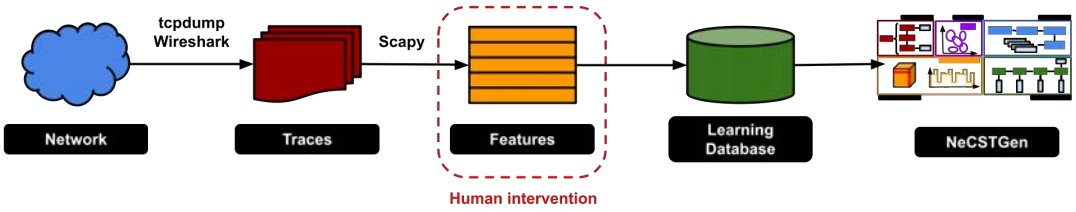

FIGURE 3.1 Illustration différentes étapes du processus d'apprentissage.

Les étapes du processus d'apprentissage sont illustrées sur la Figure 3.1. Durant l'apprentissage, les données sont collectées depuis un réseau physique ou simulé. Ces données sont des traces réseaux, collectées à l'aide de tcpdump [210] ou Wireshark [211]. Les données peuvent également être issues d'autres variables d'environnements que nous souhaitons simuler (le rapport signal sur bruit par exemple).

À l'aide de la librairie Scapy [125], les caractéristiques sont extraites depuis les traces réseau. L'extraction de ces caractéristiques nécessite une intervention humaine. Leur choix dépend de la qualité et de ce que nous souhaitons générer, mais également des connaissances que nous possédons sur le trafic à générer. Les caractéristiques extraites sont différentes suivant le niveau de la génération : flux, paquets ou agrégats (de flux ou d'applications par exemple). Ces caractéristiques servent à la construction de la base de données d'apprentissage utilisée par notre approche NeCSTGen.

Les différentes étapes du processus de génération sont illustrées sur la Figure 3.2. Les données d'apprentissage précédemment collectées vont être utilisées comme référence pour

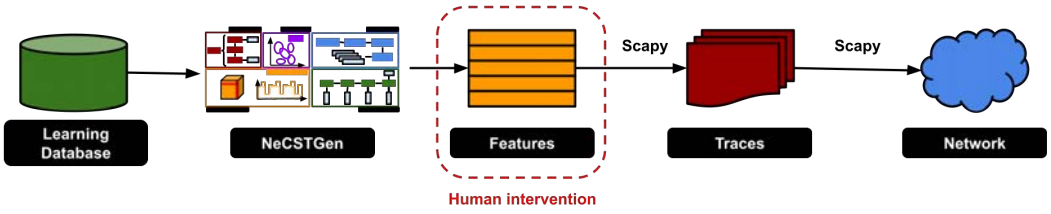

FIGURE 3.2 Illustration des différentes étapes du processus de génération.

la génération. Par exemple, si nous souhaitons générer un événement particulier, il est nécessaire de donner en entrée du modèle les caractéristiques extraites à partir des traces de cet événement. NeCSTGen fournit en sortie un ensemble de caractéristiques, de niveau paquet, flux ou agrégat. Ces caractéristiques servent à reconstruire un ensemble de traces réseau à l'aide de l'outil Scapy [125]. Ces traces peuvent être injectées dans un réseau physique simulé.

NeCSTGen peut être généralisée à tous les niveaux de génération. Ainsi, si nous souhaitons générer un ensemble de flux avec un ensemble de paquets pour chacun, la génération se fait en deux étapes. Dans un premier temps, nous générons un ensemble de flux avec ces caractéristiques. Dans un deuxième temps, nous générons les paquets associés à chaque flux.

3.3.2 Extractions des caractéristiques

À partir des traces réseaux, récoltées à l'aide de tcpdump [210] ou Wireshark [211], des caractéristiques sont extraites. Les caractéristiques employées lors du processus d'apprentissage sont différentes de celles utilisées lors de la génération. Les caractéristiques produites lors de la génération permettent la construction d'un paquet, flux ou agrégat. En revanche, les caractéristiques utilisées lors de l'apprentissage peuvent être plus nombreuses afin de fournir une compréhension fine du trafic à générer par NeCSTGen.

Les caractéristiques extraites sont différentes en fonction de ce que nous souhaitons générer. Générer un paquet ne nécessite pas les mêmes informations que pour générer un flux ou un agrégat (de flux, d'application, etc.). Extraire ces caractéristiques permet une focalisation prononcée uniquement sur l'information pertinente à générer. Il pourrait, par exemple, être envisagé que les caractéristiques soient les octets ou les bits qui forment l'en-tête d'un paquet comme c'est envisagé pour certaines approches par apprentissage profond [207]. Néanmoins, cela introduit deux limites :

— **La dimension** : la taille des paquets peut parfois être importante et générer plusieurs paquets peut s'avérer long en temps de calcul pour l'apprentissage et la

génération.

— **La cohérence** : rien ne garantit une cohérence entre les différents paquets à générer. Nous supposons que le générateur apprendra de lui-même la notion de paquet, flux et d'agrégat.

Les caractéristiques extraites sont représentées sous la forme de vecteur que nous formaliserons dans la partie 3.3.3. Leur choix est effectué par une intervention humaine en fonction de la qualité de la génération souhaitée et des connaissances sur les données à générer. Nous pouvons distinguer deux types de caractéristiques :

— **Les caractéristiques de dynamique** : la différence de temps avec le paquet précédent, le débit en octets par seconde, le débit en octets par minute, etc. Les mesures sont effectuées avec des fenêtres sautantes et glissantes. Une transformation logarithmique est appliquée sur ces mesures afin de faciliter l'apprentissage de NeCSTGen dans le cas où les distributions sont très étalées.

— **Les caractéristiques de structure** : la taille d'un paquet, la taille d'un flux, le champs « flags », les protocoles à différents niveaux d'encapsulation, la taille totale du paquet, les ports source et destination, les drapeaux, la taille de l'en-tête, la taille des données, etc.

L'absence de connaissance sur la structure du protocole à générer, de la part de l'expert humain, conduit à uniquement extraire des caractéristiques génériques de trafic, comme la taille des paquets ou le débit. En revanche, une connaissance fine permettra une génération plus complexe et performante en s'intéressant à certains champs spécifiques de l'en-tête ou de l'environnement à générer.

Pour nos modélisations, nous avons choisi d'utiliser pour chaque niveau de trafic à générer (paquets, flux et agrégat) les caractéristiques répertoriées dans le Tableau 3.9. Elles nous permettent de rester générique tout en intégrant quelques variables spécifiques aux protocoles à générer. Les données catégorielles sont transformées en valeurs continue, et les données sont normalisées entre 0 et 1 afin de pouvoir être apprises par NeCSTGen.

TABLEAU 3.9 Caractéristiques utilisées lors de l'apprentissage pour la génération de chaque niveau de trafic et pour chaque type de protocole utilisé.

Niveau de trafic	**Protocoles**	**Caractéristiques**
Paquets	TCP / UDP	Type de paquet applicatif (HTTP « Request » ou « Reply » par exemple), valeur du champs « flags », numéro du port source (oui ou non correspond au port applicatif), numéro du port destination (oui ou non correspond au port applicatif), taille total du paquet, différence de temps (en secondes) avec le paquet précédent, débit instantané en octets, débit en octets par seconde, débit en octets par minute, débit en paquets par seconde, débit en paquets par minute, taille de l'en-tête, taille de la donnée.
	LoRaWAN	Type de paquet, taux de codage du paquet, bande passante du paquet reçu, facteur d'étalement, fréquence radio, statut du CRC, taille du paquet, différence de temps (en secondes) avec le paquet précédent, rapport signal sur bruit mesuré à la réception du paquet, RSSI mesuré à la réception du paquet, débit instantané en octet, débit en octets par seconde, débit en octets par minutes, débit en paquets par seconde, débit en paquets par minute, taille de l'en-tête, taille de la donnée.
Flux	TCP	Nombre de paquet dans un flux, différence de temps (en secondes) avec le début du flux précédent.
Agrégat	-	Nombre d'éléments dans l'agrégat, différence de temps (en secondes) avec le début de l'élément précédent.

3.3.3 Formalisation

Notations

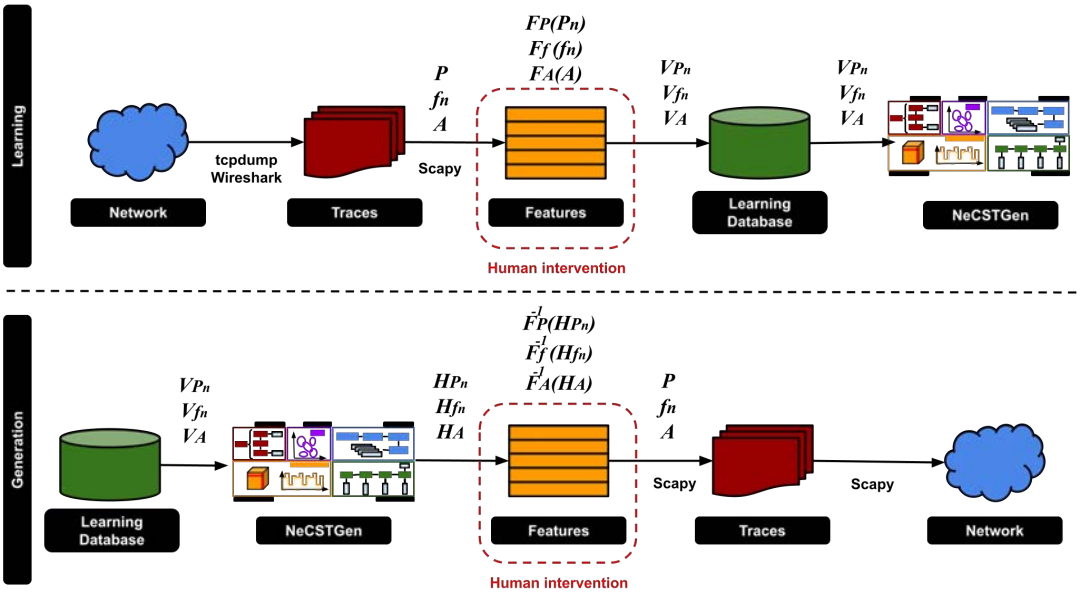

FIGURE 3.3 Illustrations des différentes étapes du processus d'apprentissage (en haut) et de génération (en bas) et leurs notations associées.

L'objectif de notre approche est de permettre la génération d'un trafic réseau similaire au trafic fourni en entrée. La Figure 3.3 illustre les différentes étapes du processus d'apprentissage et de génération ainsi que les notations associées à chacune des étapes. Ces notations seront explicitées au cours de cette partie. Nous définissons trois niveaux de génération :

- **Niveau paquet :** nous notons P l'ensemble des paquets générés p_n, appartenant au même protocole applicatif (tels que : HTTP, SMTP ...).
- **Niveau flux :** nous notons f_n un flux individuel composé d'un ensemble de paquets p_n^j, appartenant au tuple : addresse IP source, adresse IP destination, port source et port destination, avec j la position des paquets dans le flux. Dans le cas des protocoles ne possédant pas de port ou d'adresse IP (par exemple LoRaWAN) les paquets peuvent être regroupé en fonction des adresses source et destination.
- **Niveau agrégat :** nous notons A l'agrégat des flux générés par une application (un même logiciel par exemple). D'autres niveaux pourraient être définis tels que : utilisateurs, instances (navigateur web) ....

Chaque niveau est généré indépendamment. La formule (3.1) résume les différents niveaux de génération.

$$\begin{aligned} &P = \{p_0, \ldots, p_n\} \text{ avec } n \in \mathbb{N} \\ &f_n = \{p_n^0 \ldots p_n^j\} \text{ avec } j, n \in \mathbb{N} \text{ and } f_n \subset P \\ &A = \{f_0, \ldots, f_n\} \text{ avec } n \in \mathbb{N} \end{aligned} \tag{3.1}$$

Pour chaque élément (paquet, flux ou agrégat), nous avons attribué des caractéristiques extraites, grâce à l'expertise humaine, avec une fonction, notée F_c avec $F_c(x) \in \mathbb{R}^k$ où k dépend de la nature de x qui correspond à un paquet et $c \in \{P, f, A\}$, flux ou agrégat. Par exemple, dans le cas d'un ensemble de paquets à générer, la différence de temps avec le paquet précédent peut être une caractéristique obtenue avec $F_P(p_n)$.

Le but de notre approche est de prendre en entrée un ensemble de paquets notés P_{input} et d'extraire des caractéristiques pour chaque élément, notées $V_P = F_P(P_{input})$, lors de la phase d'apprentissage illustré sur la Figure 3.1. La formule (3.2) présente l'extraction des caractéristiques lors de la phase d'apprentissage.

$$\begin{aligned} &V_{P_n} = F_P(P_n) \text{ avec } n \in \mathbb{N} \\ &V_{f_n} = F_f(f_n) = \{v_n^0, \ldots v_n^j\} \text{ avec } v_{p_n}^j = F_P(p_n^j), j, n \in \mathbb{N} \text{ et } V_{f_n} \subset V_{P_n} \\ &V_A = F_A(A) = \{v_{f_0}, \ldots, v_{f_n}\} \text{ avec } n \in \mathbb{N} \end{aligned} \tag{3.2}$$

NeCSTGen apprend sur un ensemble de caractéristiques V et génère un autre ensemble de caractéristiques H caractéristiques avec $n \in \mathbb{N}$ qui permet de construire les paquets, flux ou agrégat à générer. Les éléments de H sont obtenus lors de la phase de génération illustrée sur la Figure 3.1.

À partir des vecteurs de caractéristiques H, les éléments tels que les paquets, flux et agrégats subissent une transformation inverse afin de pouvoir être utilisées dans des traces réseau. Pour la génération de paquets, à partir des vecteurs générés h_n, Scapy [125] génère les en-têtes de protocole et un fichier « .pcap » est créé. La formule (3.3) présente la transformation inverse des caractéristiques en paquet, flux ou agrégat lors de la phase de génération.

$$\begin{aligned} &P_n = F_P^{-1}(H_{P_n}) \text{ avec } n \in \mathbb{N} \\ &f_n = F_f^{-1}(H_{f_n}) = \{p_n^0, \ldots p_n^j\} \text{ avec } p_n^j = F_P(h_n^j), j, n \in \mathbb{N} \text{ et } H_{f_n} \subset H_{P_n} \\ &A = F_A^{-1}(H_A) = \{f_0, \ldots, f_n\} \text{ avec } n \in \mathbb{N} \end{aligned} \tag{3.3}$$

L'approche que nous proposons est adaptative, les caractéristiques choisies dépendent des données en entrée et de l'expertise humaine, contrairement aux approches tradition-

nelles [156, 197, 202, 212].

Génération de paquets

L'objectif est de générer un vecteur qui correspondra à toutes les caractéristiques permettant la production d'un paquet.

Un paquet, noté p_n, est composé de deux parties : les en-têtes et les données. Nous décidons de nous concentrer uniquement sur la génération des en-têtes. En effet, les en-têtes ont des structures en champs, chaque valeur a une position relativement bien définie. En revanche, les données utiles ont des structures plus complexes à exploiter, il y a des informations séquentielles et peu de structure en champs.

Il est nécessaire de définir la limite entre l'en-tête et les données d'un paquet. Au vu des résultats précédents, nous décidons de considérer comme appartenant à l'en-tête les protocoles de niveau 2, 3 et 4 du modèle OSI. Les protocoles de niveau supérieur (HTTP, SMTP, ...) sont considérés comme des applications. Les protocoles définis comme application ont des structures beaucoup plus variables et sont plus proches des structures textuelles. La structure en champ est moins définie. Par exemple, une requête HTTP « GET » inclut l'adresse d'un nom de domaine de taille variable avec une structure de type texte.

Génération de flux

L'objectif est de générer un ensemble de vecteurs de caractéristiques similaires aux vecteurs $v_n^0, \dots v_n^j$, avec $n \in \mathbb{N}$ et j la position des paquets associés dans le flux.

Nous définissons un flux f_n comme l'ensemble des paquets p_n^j qui appartiennent au tuple : (IP source, IP destination, ports source et destination).

Nous distinguons deux modes de fonctionnement pour un flux : (i) le mode connecté incluant TCP, (ii) le mode non connecté incluant UDP. Un flux en mode connecté a un début, un milieu et une fin. Par exemple, dans TCP, les valeurs du champ « flags » seront utilisées pour distinguer ces parties. Ce n'est pas le cas pour UDP. Ainsi, pour être en mesure de générer un flux, nous devons être capables de :

- **identifier** les paquets qui caractérisent un flux ;
- **identifier**, dans le cas du mode connecté, la position des paquets, notée j, qui ne changent pas à l'intérieur d'un flux. Par exemple, dans un flux TCP, le paquet avec le champs « flags » à SYN sera toujours le premier paquet du flux.

L'identification de ces informations sera expliquée dans la partie 3.4.

Génération d'agrégats

L'objectif est de générer un ensemble de vecteurs de caractéristiques $V_A = F_A(A)$ représentant un agrégat.

La génération de ces ensembles est similaire à la génération d'un flux, sauf qu'elle ne nécessite pas d'identifier la position de l'élément. En effet, contrairement à la génération d'un ensemble de paquets P, les vecteurs de caractéristiques $v^{f_n} = F_A(f_n)$ peuvent être définies par deux variables seulement : la taille des éléments de l'agrégat et la différence de temps avec l'élément précédent dans l'agrégat.

3.3.4 Jeux de données

Les données utilisées proviennent de trois sources et sont de natures différentes telles qu'on pourrait en trouver au sein des « Smart Cities ». La première source de données est un réseau DARPA simulé [213]. Les données ont été collectées sur un réseau similaire à celui d'une entreprise. Ces données ont l'avantage d'être non chiffrées, avec des protocoles courants et bien documentées.

La deuxième source provient de la capture de commandes vocales réalisée avec un Google Home [214]. Chaque fichier de capture comprend une commande vocale.

La dernière source de données est un réseau LoRaWAN déployé dans un centre-ville et collecté à partir de différentes passerelles [215]. Les passerelles d'où proviennent les données collectées ont la particularité d'avoir des dynamiques différentes. Cela nous permet d'avoir une grande diversité de comportement du réseau. Nous sélectionnons deux applications avec des dynamiques différentes pour notre génération.

Le Tableau 4.5 liste les jeux de données utilisés, les applications et les quantités de paquets qui leur sont associées.

Tableau 3.10 Liste des applications dans chaque classe pour chaque groupe des deux jeux de données.

Jeu de données	Application	Nombre de paquets	Total
DARPA [213]	HTTP	461 603	1 027 138
	SMTP	437 221	
	SNMP	128 314	
Google Home [214]	QUIC/UDP	382 410	832 278
	TCP	449 868	
LoRaWAN [215]	fport 1	492 640	903 495
	fport 10	410 855	

Ces ensembles de données proviennent de différents réseaux avec différents types de

trafic. Ainsi, nous pouvons évaluer l'adaptabilité de notre approche sur différents types de trafic. Les données ont été séparées en deux parties pour chaque application : entraînement (70%) et validation (30%). Les données de validation sont utilisées pendant la formation pour déterminer les meilleurs hyper-paramètres et éviter le sur-ajustement. Pendant le processus de génération, toutes les données sont utilisées. La séparation des données ne comprend pas de jeux de données de test car l'objectif n'est pas de correspondre exactement à une sortie définie mais de générer des données similaires en respectant certains critères abordés lors de l'évaluation de NeCSTGen dans la section 3.5.

3.4 Network Clustering Sequential Traffic Generation (NeCSTGen)

Dans cette section, nous allons nous intéresser à l'architecture et au fonctionnement de NeCSTGen. Dans un premier temps, nous présenterons les différents modèles qui composent l'architecture ainsi que leur fonctionnement. Dans un second temps, nous détaillerons le fonctionnement des différents types de génération (paquets, flux et agrégat).

3.4.1 Architecture de NeCSTGen

Présentation générale

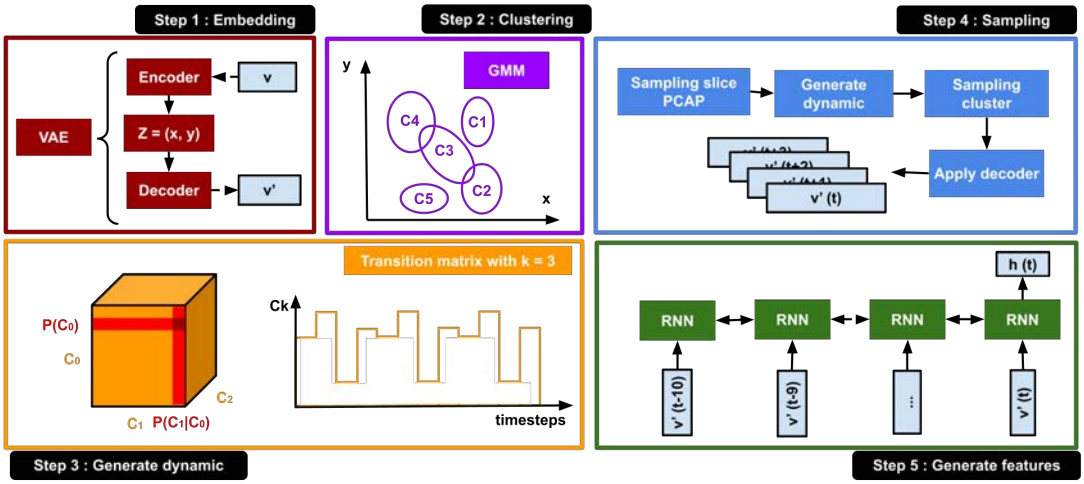

FIGURE 3.4 Aperçu de l'architecture d'apprentissage profond, appelé NeCSTGen, pour la génération de trafic réseau.

La Figure 4.2 montre l'architecture proposée, nommée NeCSTGen, ainsi que les différentes étapes pour la génération du trafic réseau. Chaque étape est exécutée de manière séquentielle. Les modèles de l'architecture prennent en entrée la sortie d'un ou plusieurs modèles des étapes précédentes. Au total, quatre modèles sont utilisés pour la génération : (i) VAE, (ii) GMM, (iii) GRU bidirectionnel, (iv) Matrice de transition. L'entraînement des modèles de NeCSTGen est effectué sur l'infrastructure OSIRIM [81].

Variational Autoencodeur (VAE)

Le VAE, représenté à l'étape 1 de la Figure 4.2, projette les caractéristiques extraites v_n du paquet p_n dans un espace latent bidimensionnel Z. Le but de la projection est de mettre en évidence les paquets ayant des caractéristiques similaires ainsi que d'extraire des informations dynamiques de haut niveau. Ainsi, un échantillon de l'espace latent Z, noté z_n, encapsule des informations sur un type de paquet (champs d'en-tête et statistiques) ainsi que des informations sur le débit au moment de la capture (dynamique de haut niveau).

Le modèle est composé d'un encodeur E et d'un décodeur D avec $D \approx E^{-1}$. L'espace latent est choisi de dimension deux pour faciliter l'interprétation et éviter l'utilisation d'une méthode de réduction de dimension. Après avoir été identifiées, les données sont codées sous forme d'étiquettes puis normalisées. Les détails de l'architecture du VAE utilisé sont représentées dans le Tableau 3.11.

Tableau 3.11 Architecture du VAE de l'étape 1 de la Figure 4.2.

Encodeur	**Paramètres**	**Dim. de sortie**
Input	0	$length(v_n)$
Dense (LeakyReLU)	80	8
Dense (LeakyReLU)	54	6
Dense (LeakyReLU)	28	4
Dense (LeakyReLU) mean	10	2
Dense (LeakyReLU) var	10	2
Décodeur	**Paramètres**	**Dim. de sortie**
Input	0	2
Dense (LeakyReLU)	12	4
Dense (LeakyReLU)	30	6
Dense (LeakyReLU)	56	8
Dense (Sigmoid)	81	$length(v_n)$

Gaussian Mixture Model (GMM)

Le modèle GMM est présenté à l'étape 2 de la Figure 4.2. Le but de ce modèle est d'identifier des groupes d'éléments similaires v_n sous forme de clusters, notés C_k avec $k \in \{0, .., K\}$ où K est un nombre de clusters à trouver dans l'espace latent. Le nombre K est défini par expertise humaine, par visualisation de la distribution des points dans l'espace latent. Ces éléments peuvent être des paquets, des flux ou des éléments d'un agrégat. Par exemple, dans le cas d'un paquet, cette similarité peut s'expliquer par des champs d'en-tête similaires entre deux paquets, des statistiques réseau similaires ou les deux.

L'identification peut se faire dans l'espace Z, dans le cas de la génération au niveau des paquets, ou de deux variables à échantillonner, dans le cas de la génération d'agrégats.

Recurrent Neural Network (RNN)

Le RNN est présenté à l'étape 5 de la Figure 4.2. Il s'agit d'un GRU bidirectionnel. Son but est de générer les caractéristiques de chaque paquet à générer, noté h_n. Il garantit la cohérence des caractéristiques générées entre les paquets d'un même flux f_n ou groupe de paquets P. Contrairement au VAE, il se concentre uniquement sur les statistiques à court terme.

Le modèle prend en entrée une séquence de caractéristiques échantillonnées dans l'espace latent Z et reconstruites avec le décodeur du VAE, noté $v'_{n-10} \dots v'_n$. En prenant en entrée les caractéristiques échantillonnées, grâce à la projection du VAE, il a une vision de haut niveau de la dynamique et des caractéristiques des paquets à générer. Le modèle utilise les caractéristiques des paquets précédents pour générer les caractéristiques du paquet suivant. Nous décidons d'utiliser les dix caractéristiques précédentes du paquet généré. La taille choisie est arbitraire et pour des raisons de temps de calcul, un plus grand nombre de vecteurs de caractéristiques pourrait être considéré. Ce modèle est uniquement utilisé pour générer un agrégat de paquets P ou un flux f.

Le modèle RNN fournit deux sources d'informations nécessaires à la génération : (i) des variables catégorielles, le plus souvent les champs des paquets générés, (ii) des variables continues telles que la taille des paquets. Ainsi, $h_n = \{h_n^{cat.}, h_n^{cont.}\}$. Intuitivement, la génération peut être assimilée à la génération de séries temporelles multivariées. L'architecture multitâche récurrente utilisée est représentée dans le Tableau 3.12.

Pendant le processus d'apprentissage, le modèle va essayer d'optimiser une fonction de perte qui résulte de la combinaison de plusieurs fonctions de perte représentées dans la formule (3.4).

TABLEAU 3.12 Architecture multitâche récurrente de l'étape 5 de la Figure 4.2.

RNN	**Paramètres**	**Dim. de sortie**
Input	0	$length(v'_{n-10} \dots v'_n)$
Bidirectional GRU	3264	-
Dense (LeakyReLU)	231	-
Dense (LeakyReLU)	132	-
Sortie cat.	0	$length(h_n^{cat.})$
Sortie cont.	0	$length(h_n^{cont.})$

$$L(\theta) = \alpha L_{CCE}(\theta_{cat.}) + \beta L_{MAE}(\theta_{cont.}) \text{ avec } \alpha, \beta = 0.5 \tag{3.4}$$

La fonction de perte catégorielle à entropie croisée $L_{CCE}(\theta_{cat.})$ permet de générer des caractéristiques catégorielles telles que le champ « flags », dans le cas de TCP. La fonction de perte $L_{MAE}(\theta_{cont.})$ est une erreur absolue moyenne. Elle permet de générer des caractéristiques quantitatives. Pour rappel, c'est à partir du vecteur généré h_n que Scapy [125] génère les en-têtes de protocole et un fichier « .pcap » est créé.

Matrice de transition

La génération de la séquence est traitée à l'étape 3 de la Figure 4.2. C'est une matrice de dimension j qui apprend les transitions entre les clusters $C_{i-j,k} \dots C_{i,k}$ à partir des données d'apprentissage. La valeur de j est arbitraire et dépend de la taille de la séquence de clusters utilisée pour construire la matrice de transition.

Les modèles RNN ont des difficultés à générer de longues séries temporelles. De plus, la plupart des phénomènes sur les réseaux sont régis par des comportements humains, difficiles à prévoir. Afin de remédier aux problèmes précédents, nous décidons de générer des séries temporelles de manière empirique en utilisant l'échantillonnage.

La matrice de transition est un modèle de plus haut niveau utilisé pour générer une séquence d'informations qui peut être associée à une dynamique globale. Cette dynamique globale va permettre de conditionner la génération des caractéristiques des éléments à générer (paquets, flux, agrégat de flux, ...).

Cette information de haut niveau sur les caractéristiques est déjà identifiée par les clusters C_k. Nous pouvons donc générer une longue séquence de clusters, identifiés grâce aux GMM, en échantillonnant la matrice de transition afin de générer une dynamique globale.

3.4.2 Génération de paquet

Pour chaque paquet p_n, nous extrayons un vecteur de caractéristiques $v_n = F(p_n)$. Chaque vecteur de caractéristiques v_n est projeté dans un espace latent, noté Z, avec z_n les vecteurs projetés. Cette projection est réalisée à l'aide de la matrice E instanciée par l'encodeur VAE lors de l'étape 1 de la Figure 4.2. Le but de la projection est de mettre en évidence les paquets ayant des caractéristiques similaires ainsi que d'extraire des informations dynamiques de haut niveau. Ainsi, un échantillon de l'espace latent Z, noté z_n, encapsule des informations sur un type de paquet (champs d'en-tête et statistiques) ainsi que des informations sur le débit au moment de la capture (dynamique de haut niveau). La représentation et l'interprétation de l'espace latent permettent une compréhension fine pour la génération de paquets. Elles sont illustrées dans la Figure 3.5.

L'ensemble des vecteurs $\{z_0, \ldots, z_n\}$ qui appartiennent au même cluster, noté C_k, est identifié par le GMM. Cette identification est réalisée lors de l'étape 2 de la Figure 4.2. L'objectif de ce modèle est d'identifier des groupes d'éléments similaires v_n sous forme de clusters, notés C_k avec $k \in \{0, ..., K\}$ où K est un nombre de clusters à identifier dans l'espace latent, par visualisation de celui-ci, grâce à une intervention humaine. L'identification peut s'effectuer automatiquement car l'algorithme GMM le permet. Cependant, l'identification automatique ne garanti pas une identification correct de l'ensemble des clusters. Ces clusters peuvent être des paquets, des flux ou des éléments d'un agrégat. Par exemple, dans le cas d'un paquet, cette similarité peut s'expliquer par des champs d'en-tête similaires entre deux paquets, des statistiques réseau similaires ou les deux. GMM associe à chaque cluster C_k une densité de probabilité gaussienne de paramètres (μ_k, Σ_k), noté $P_k \sim N(\mu_k, \Sigma_k)$ avec μ_k l'espérance et Σ_k la matrice de covariance. La formule (3.5) présente la notion de clusters.

$$\begin{aligned} &C_k = \{z_0, \ldots, z_n\} \text{ avec } z_i =< v_i, E >, \forall i \in \{0, \ldots, n\}, \\ &P_k \sim N(\mu_k, \Sigma_k) \text{ avec } n \in \mathbb{N} \text{ et } k \in \{0, \ldots, K\}. \end{aligned} \tag{3.5}$$

GMM est un algorithme approprié car le VAE impose une contrainte de loi Normale $\mathcal{N}(0, 1)$ sur la projection des caractéristiques dans l'espace latent Z. La Figure 3.6 représente la projection bi-dimensionnelle z_n. A gauche, l'histogramme issue du nuage de points généré par l'encodeur du VAE montre la concentration des points. À droite, les cercles représentent la covariance Σ_k associée à chaque densité de probabilité.

Ainsi, chaque cluster C_k identifie un ensemble de paquets ayant des caractéristiques similaires. Pour générer un vecteur de caractéristiques, nous échantillonnons un cluster C_k en utilisant la densité de probabilité associée P_k. Le $\widetilde{z_n}$ obtenu permet d'effectuer la transformation inverse en utilisant la matrice D, instancié par le décodeur VAE, avec

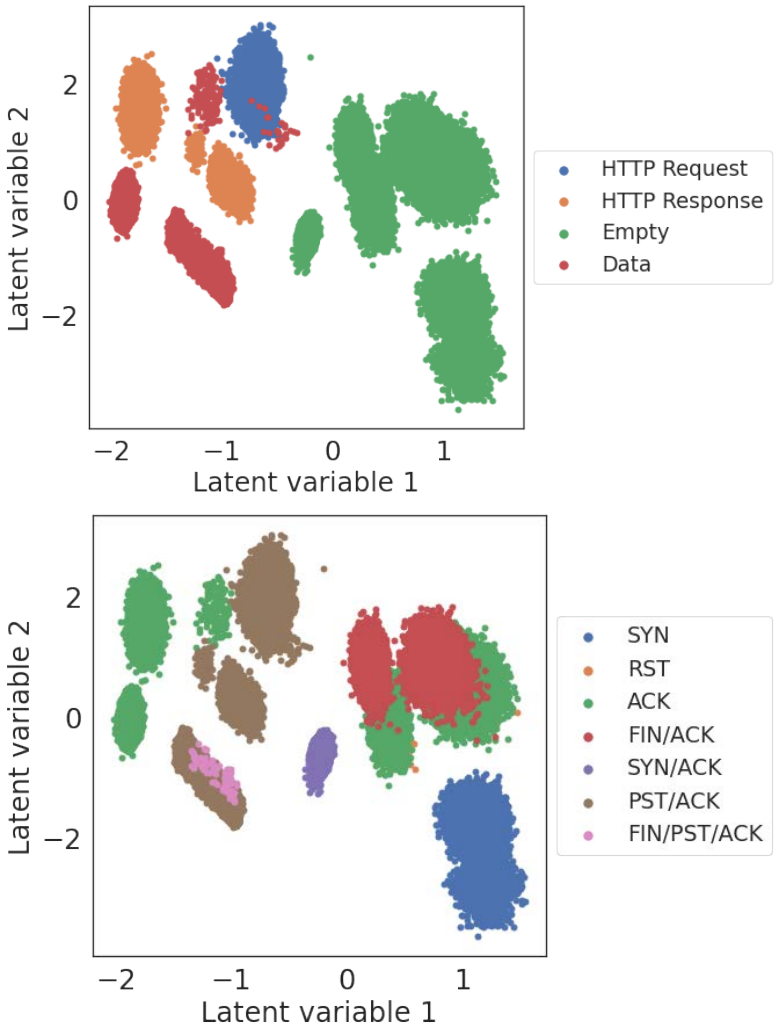

FIGURE 3.5 Espace latent de l'étape 1 de la Figure 4.2 pour HTTP. Chaque point représente le vecteur z_n, la projection bi-dimensionnelle de v_n avec E. La légende montre quelques critères qui expliquent la formation des clusters C_k.

$D \approx E^{-1}$, pour obtenir les caractéristiques du cluster v'_n. La formule (3.6) présente cette transformation avec ϵ l'erreur additive de transformation. La Figure 3.7 illustre l'application de la transformation présenté dans la formule (3.6).

$$\begin{aligned} &\widetilde{z_n} \sim P_k(\alpha) \text{ avec } \widetilde{z_n} \in C_k \\ &v'_n =< \widetilde{z_n}, D > \text{ et } v'_n \approx v_n + \epsilon \end{aligned} \tag{3.6}$$

Les informations contenues dans le vecteur v'_n sont suffisantes pour générer un paquet. Cependant, si nous voulons échantillonner deux vecteurs consécutifs v'_{n-1} et v'_n, il n'y

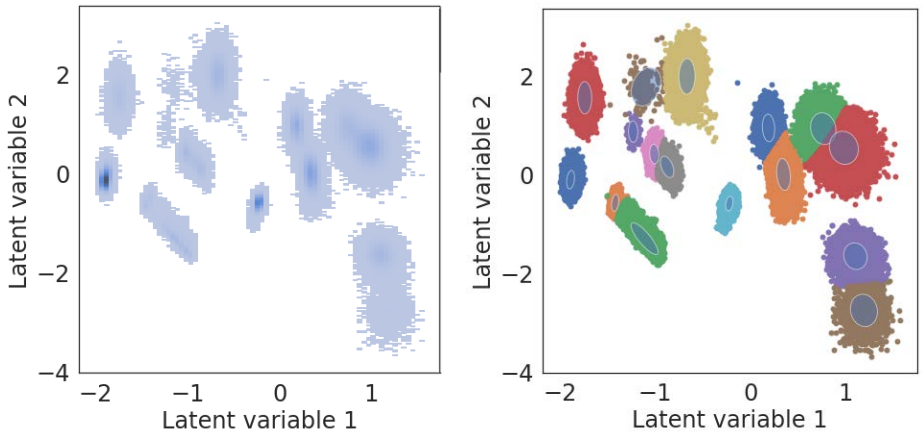

FIGURE 3.6 Espace latent de l'étape 1 de la Figure 4.2 pour HTTP. Chaque point représente les vecteurs z_n, la projection bidimensionnelle de v_n avec E. À gauche, l'histogramme issue du nuage de points généré par l'encodeur du VAE montre la concentration des points. À droite, les cercles représentent la covariance Σ_k de chaque densité de probabilité associée à un cluster C_k défini lors de l'étape 2.

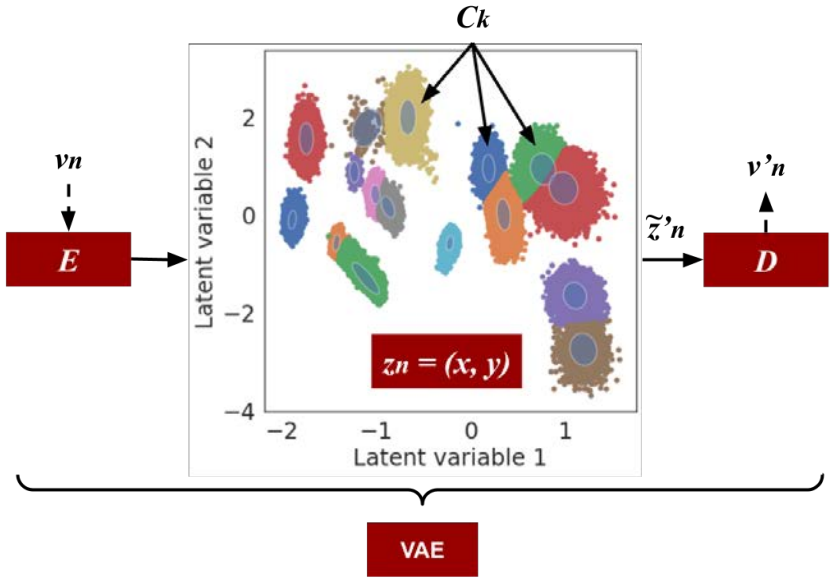

FIGURE 3.7 Illustration de l'application de la transformation présentée dans la formule (3.6).

a pas de cohérence entre les valeurs des caractéristiques des deux vecteurs. Pourtant, il existe une cohérence entre les valeurs des champs d'en-tête de deux paquets p_{n-1} et p_n d'un même flux. Par conséquent, nous introduisons le RNN ainsi que la matrice de

transition pour la génération de flux ou d'agrégats cohérents détaillés dans la partie suivante.

3.4.3 Génération de flux

Générer un flux f_n consiste à générer un ensemble de vecteurs $v'^{\,0}_n, \dots v'^{\,j}_n$ dont les valeurs doivent être cohérentes dans le temps avec j représente la position du paquet associé p^j_n dans le flux. Pour cela, nous définissons $\{h^0_n, \dots h^j_n\} = H$ les vecteurs de caractéristiques à générer afin d'obtenir un flux cohérent.

L'espace latent Z nous permet d'identifier les traits des paquets qui caractérisent un flux. Dans le cas du mode connecté, nous définissons un flux avec trois parties : un début, un milieu et une fin respectivement notés B, M et E. Nous notons b le nombre de paquets au début du flux, m le nombre de paquets au milieu du flux et e le nombre de paquets à la fin du flux. A chaque paquet dont la position est identifiée, nous associons un espace latent noté respectivement Z^B, Z^M et Z^E présenté dans la formule (3.7).

$$\begin{aligned} &Z^B_0 \dots Z^B_i \text{ avec } i \in [0, b] \text{ et } i, b \in \mathbb{N} \\ &Z^M_0 \dots Z^M_i \text{ avec } i \in [0, m] \text{ et } i, m \in \mathbb{N} \\ &Z^E_0 \dots Z^E_i \text{ avec } i \in [0, e] \text{ et } i, e \in \mathbb{N}. \end{aligned} \tag{3.7}$$

Un espace latent peut donc projeter des paquets dont la position dans le flux est différente. En procédant ainsi, nous pouvons identifier les paquets qui caractérisent un flux en fonction de leur position. Ainsi, dans le cas d'un flux en mode non connecté, un espace latent unique Z est suffisant. Pour chaque espace latent, nous trouvons un ensemble de clusters, notés $C^X_{i,k}$ avec :

— X : l'appartenance des caractéristiques aux paquets de sous-ensembles (début, milieu ou fin), donc $X = \{B, M, E\}$.
— i : la position des paquets au sein du groupe avec $i \in [0, b], [0, m], [0, e]$.
— k : le numéro du cluster $k \in \{0, \dots, K\}$.

La formule suivante (3.8) détaille pour chaque espace latent la notation utilisée pour identifier les clusters.

$$\forall i \in [|0, x|], C^X_{i,k} \in Z^X_i \text{ avec } x \in \{b, m, e\} \text{ et } X \in \{B, M, E\}. \tag{3.8}$$

La Figure 3.8 illustre la projection des différents espaces latents utilisés pour générer le début (noté $Z^B_0 \dots Z^B_b$ avec $b = 4$), le milieu (noté $Z^M_0 \dots Z^M_m$ avec $m = \mathbb{N}$) et la fin (noté $Z^E_0 \dots Z^E_e$ avec $e = 4$) d'un flux HTTP.

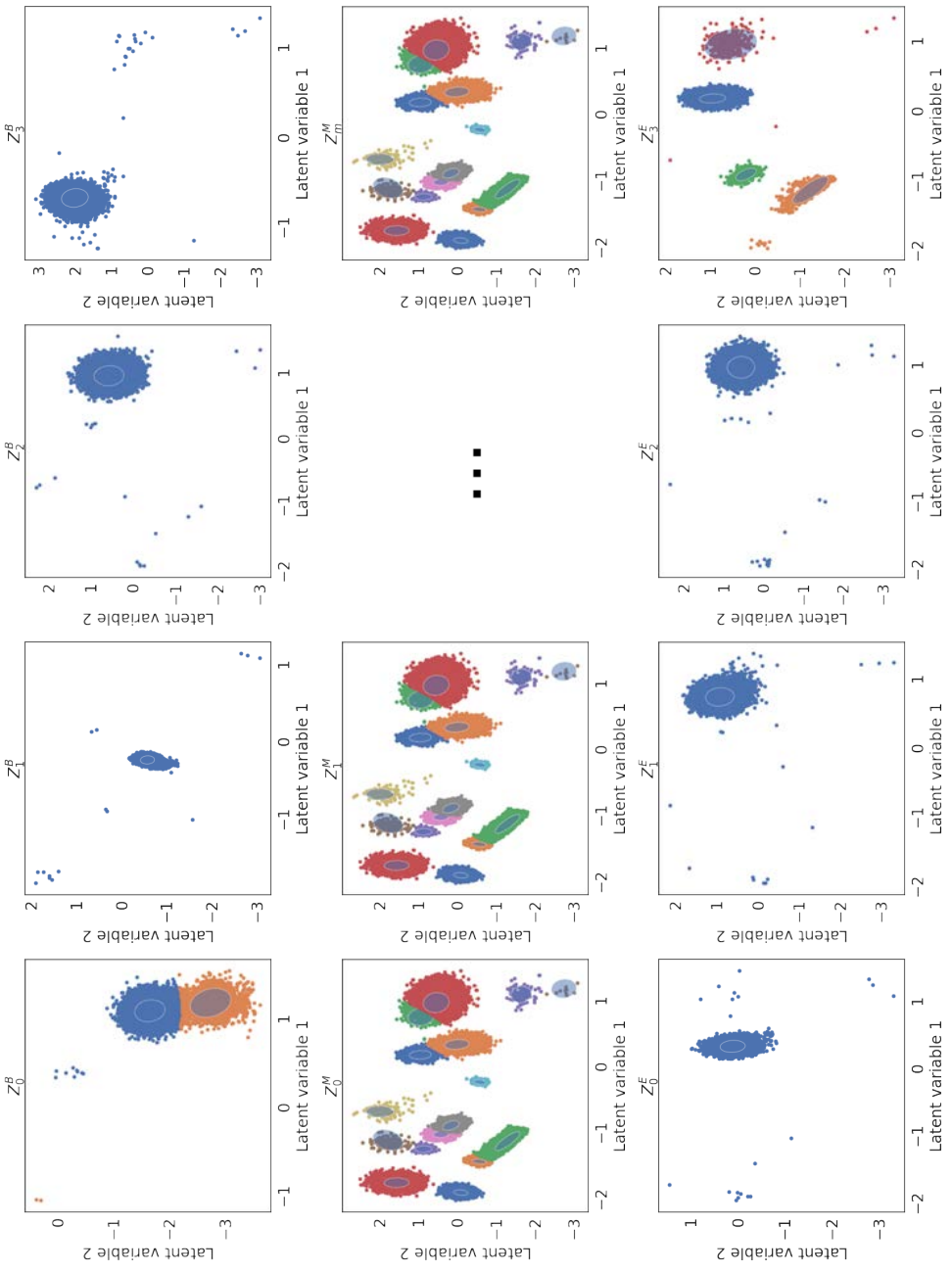

FIGURE 3.8 Espaces latents de l'étape 1 de la Figure 4.2 utilisés pour générer le début (noté $Z_0^B \dots Z_b^B$ avec $b = 4$), le milieu (noté $Z_0^M \dots Z_m^M$ avec $m \in \mathbb{N}$) et la fin (noté $Z_0^E \dots Z_e^E$ avec $e = 4$) d'un flux HTTP. Chaque point représente les vecteurs z_n. Les cercles représentent la covariance Σ_k de chaque densité de probabilité associée à un cluster $C_{i,k}^X$, avec $X \in \{B, M, E\}$ défini lors de l'étape 2.

La génération d'un flux va consister à trouver les séquences de clusters $C^X_{i,k}$ à échantillonner afin de générer les caractéristiques de chaque paquet d'un flux. Ainsi, la probabilité d'échantillonner le cluster $C^X_{i,k}$ dépend des clusters échantillonnés précédant $C^X_{i-j,k}$ avec j le rang des clusters précédents. Cette tâche est réalisée par la matrice de transition lors de l'étape 3 de la Figure 4.2. La formule (3.9) explique ce concept.

$$\forall X \in \{B, M, E\}, P(C^X_{i+1,k}) = P(C^X_{i,k} | C^X_{i-1,k} \dots C^X_{i-j,k}) \text{ avec } i, j \in \mathbb{N}. \tag{3.9}$$

La matrice de transition est calculée à partir des clusters dans les données. Pour la génération, il suffit d'échantillonner la matrice de transition afin de générer une longue séquence de clusters. La Figure 3.9 illustre la génération des clusters $C^X_{i+1,k}$ pour les protocoles HTTP et SMTP.

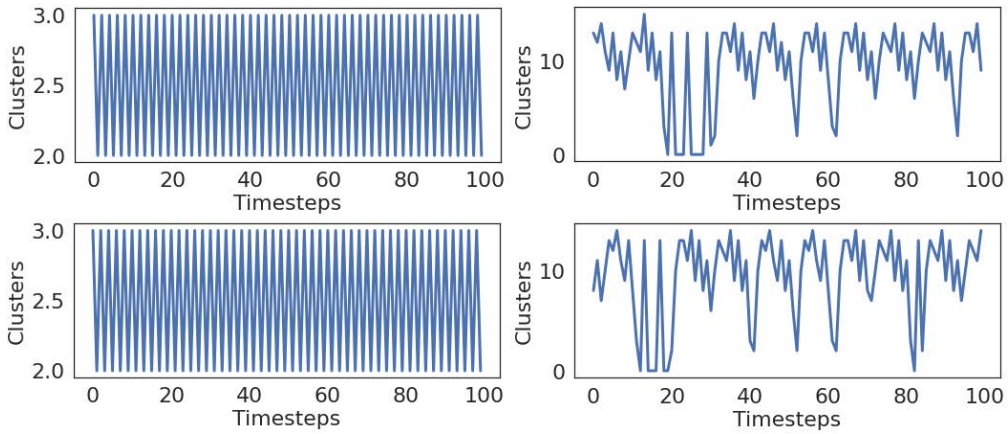

FIGURE 3.9 Génération d'une séquence de clusters C^X_k, échantillonnées grâce à la matrice de transition, pour les protocoles Simple Network Management Protocol (SNMP) (à gauche) et HTTP (à droite). La dynamique en haut représente la vraie séquence de clusters et en bas la séquence de clusters générée conditionnée avec les $C^X_{i-1,k} \dots C^X_{i-7,k}$ clusters précédents donc $j = 7$.

La matrice de transition est un modèle de plus haut niveau utilisé pour générer une séquence d'informations qui peut être associée à une dynamique globale. Ces dynamiques globales vont permettre de conditionner la génération des caractéristiques des éléments à générer.

Une fois la séquence de clusters générée, chaque cluster est échantillonné pour obtenir les $\widetilde{z_n^{j-10}}, \dots \widetilde{z_n^j}$ avec $\widetilde{z_n}$ un échantillon de l'espace latent. Ensuite, le décodeur D, issu du VAE, est utilisé pour obtenir les vecteurs $v_n'^{j-10} \dots v_n'^{j}$. Enfin, avec le RNN, nous pouvons obtenir le vecteur h_n^j, cohérent avec le précédent vecteur généré. La génération

avec le modèle RNN est définie par :

$$RNN({v'_n}^{j-10} \dots {v'_n}^{j}) = h_n^j \text{ avec } n, j \in \mathbb{N}. \tag{3.10}$$

h_n^j contient les informations pour la construction d'un paquet p_n^j à l'intérieur du flux f_n. Par exemple, la différence de temps entre p_n^{j-1} et p_n^j. Ainsi, la génération de la dynamique à court terme est effectuée par le modèle RNN. la dynamique à long terme est intégrée dans l'espace latent Z représenté sur la Figure 3.10.

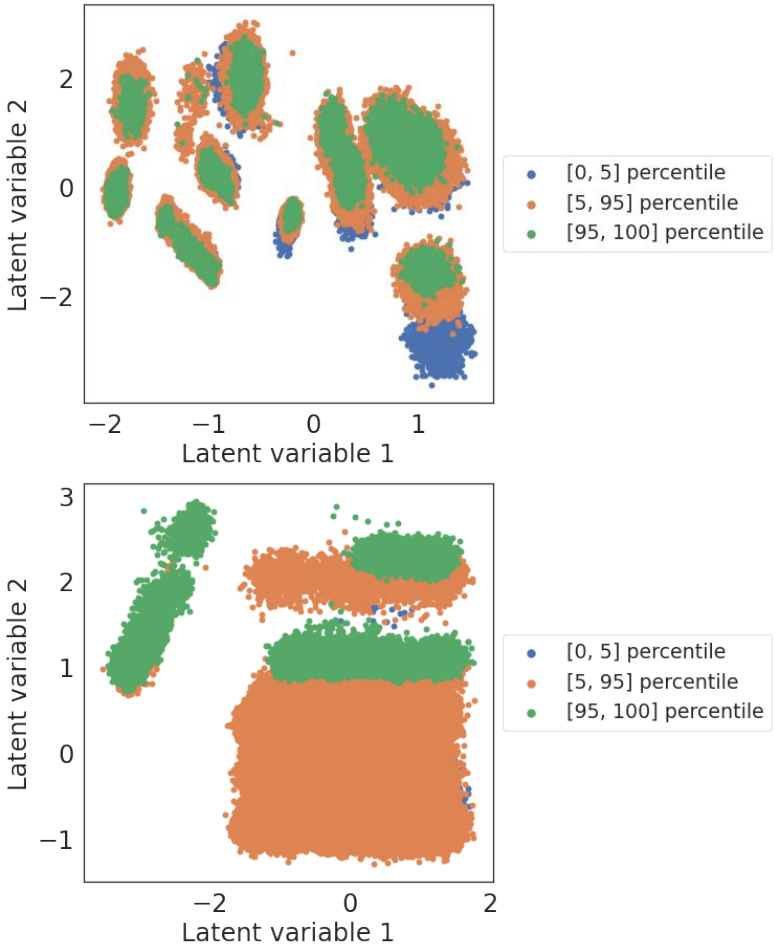

FIGURE 3.10 Intégration de la dynamique dans l'espace latent représentée sous forme de plage de percentile en octets/s. Chaque point représente un paquet projeté, noté z_n^j. En haut, l'espace latent de HTTP et en bas celui de LoRaWAN (champs « fport » à 1).

Ainsi, la sortie du décodeur du VAE, noté D, c'est-à-dire ${v'_n}^{j-10} \dots {v'_n}^{j}$, agit comme un a priori qui va conditionner la génération du court terme car celle-ci a une influence

sur le long terme.

L'utilisation d'une combinaison de modèles permet un meilleur contrôle lors de la génération. Il est possible d'échantillonner une séquence précise de cluster, noté $C_{i,k}$, pour obtenir un comportement précis. De plus, l'échantillonnage permet de créer de la variabilité ce qui permet une génération plus réaliste. Si un seul modèle est utilisé pour la génération celui-ci aura tendance à converger vers un état stable qui n'est pas adapté avec la réalité de la variabilité d'un trafic réseau.

3.4.4 Génération d'agrégat

Un agrégat A peut être caractérisé par seulement deux variables : la taille des éléments de l'agrégat et la différence de temps avec l'élément précédent dans l'agrégat. Ce sont les informations contenues dans v_n. Ces deux variables sont suffisantes et ne nécessitent pas une projection dans un espace Z avec un VAE.

L'identification des clusters C_k et leur génération suivent la même procédure que la génération des flux. Néanmoins, aucun modèle RNN n'est utilisé, car il n'y a pas de caractéristique dynamique à court terme entre le flux ou l'ensemble de flux. En effet, la plupart des flux sont pilotés par un comportement humain hautement imprévisible. L'échantillonnage nous permet de simuler ce caractère aléatoire. Ainsi, $v'_n = h_n = F(f_n)$.

3.4.5 Performances

Cette section résume les performances et l'infrastructure utilisée pour la génération et l'apprentissage des différents éléments de NeCSTGen. Le Tableau 3.13 répertorie le temps d'apprentissage et de génération des différents modèles de NeCSTGen en fonction du nombre de paquets utilisés. L'apprentissage, la génération et les mesures sont effectuées sur l'architecture Osirim [81], la plateforme de calcul de l'IRIT, en utilisant un GPU Nvidia GeForce GTX 1080Ti.

TABLEAU 3.13 Mesures de performances d'apprentissage et de génération pour chaque élément de NeCSTGen et pour chaque protocole.

Modèles	**Jeux de données**	**Applications**	**Nombre de paquets**		**Temps de calcul**	
			Apprentissage	**Génération**	**Apprentissage**	**Génération**
VAE	DARPA [213]	HTTP	461 603	10 000	13 min	10 s
		SMTP	437 221	10 000	12 min	10 s
		SNMP	128 314	10 000	4 min	< 1 s
	Google Home [214]	QUIC/UDP	382 410	10 000	14 min	< 1 s
		TCP	449 868	10 000	18 min	< 1 s
	LoRaWAN [215]	fport 1	492 640	10 000	14 min	< 1 s
		fport 10	410 855	10 000	12 min	< 1 s
RNN	DARPA [213]	HTTP	461 603	10 000	4 min	25 min
		SMTP	437 221	10 000	4 min	25 min
		SNMP	128 314	10 000	3 min	23 min
	Google Home [214]	QUIC/UDP	382 410	10 000	17 min	24 min
		TCP	449 868	10 000	14 min	25 min
	LoRaWAN [215]	fport 1	492 640	10 000	9 min	30 min
		fport 10	410 855	10 000	3 min	30 min

Les données d'apprentissages comprennent les données d'entraînements (70%) et validation (30%). Le temps de génération et d'apprentissage de la matrice de transition et du GMM peut être considéré comme négligeable (quelques secondes). Dans le cas de la matrice de transition, l'apprentissage consiste à un calcul fréquentiel et la génération à de l'échantillonnage. Pour le GMM l'apprentissage est également très court, car les clusters peuvent être facilement identifiables graphiquement et la génération consiste à de l'échantillonnage. La génération d'un flux ou d'un agrégat de flux utilise le GMM et la matrice de transition uniquement. Ainsi, le temps de génération d'un agrégat de flux ou autre peut être considéré comme négligeable.

Le temps de génération total se calcule en additionnant le temps de génération du VAE et du RNN. Le temps mesuré ici est effectué pour la génération de 10 000 paquets. La différence de temps d'apprentissage et de génération s'explique par le nombre de paquets utilisés pour chaque jeu de données, mais également le nombre de variables utilisées lors de l'apprentissage, répertoriées dans le Tableau 3.9.

3.4.6 Conclusion

Nous avons présenté notre approche et son fonctionnement au niveau du flux, du paquet et de l'agrégat. Nous avons également montré comment apprendre à générer l'aspect dynamique, à différentes échelles par la projection d'un VAE et une approche basée sur l'apprentissage multi-tâche utilisant un RNN. Enfin, nous avons présenté une solution basée sur une matrice de transition permettant de générer du trafic sur une longue séquence tout en gardant un aspect stochastique et une bonne dynamique globale. Dans la prochaine partie, nous étudierons les performances de l'approche en comparant nos résultats sur les données d'origine.

3.5 Validation

Dans cette section, nous comparons les résultats avec les approches traditionnelles appartenant à la catégorie « Générateurs de haut niveau et autoconfigurables », la catégorie des générateurs traditionnels la plus proche de NeCSTGen, telle que : HARPOON [155], LitGen [158] et Swing [156]. Les résultats seront également comparés à d'autres approches par apprentissage profond [216].

3.5.1 Méthodologie

Pour démontrer l'adaptabilité et les performances de NeCSTGen, nous utilisons plusieurs jeux de données provenant de différentes sources, protocoles et réseaux. L'évaluation s'effectue en se basant sur une heure ou quelques heures de trafic généré, en fonction du type de trafic. Cela représente entre 32 000 paquets (pour LoRaWAN) à 40 000 paquets pour les autres protocoles. La validation est effectuée à différents niveaux en suivant les recommandations de [12] :

- **Dynamique des paquets** : évalue le réalisme en termes de débit en octets/s, temps d'arrivée des paquets.
- **Champs des paquets** : estime les erreurs sur les champs à générer, les erreurs et la cohérence entre les valeurs générées à partir de différents champs.
- **Flux et dynamique des agrégats** : en termes de taille des flux et de volume des flux.
- **Caractéristiques des échelles** : données par une analyse en ondelettes est effectuée pour évaluer la qualité de la génération à différentes échelles.
- **QoS/Quality of Experience (QoE)** : la distribution des Round Trip Time (RTT).

Notre objectif est d'évaluer la génération de la dynamique ainsi que les champs des paquets. Nous définissons un ensemble de métriques, pour chaque niveau de génération, en suivant les recommandations de [12].

Pour la dynamique, nous étudions le temps d'arrivée entre éléments, noté Δ_t, et la somme des tailles des éléments (qui peuvent être des paquets, des flux ou des agrégats de flux, notés l_i^e) sur 1 seconde, notée L^e. Le débit D est alors évalué en suivant l'équation (3.11).

$$D = \frac{L_e[\text{octets}]}{\Delta_t[\text{seconde}]} \text{ avec } L_e = \sum_{i=0}^{n} l_i^e \tag{3.11}$$

Dans le cas d'une génération de flux, nous évaluons également la distribution des RTT.

En plus, nous utilisons une Logscale Diagram Estimate (LDE) [217] pour effectuer une analyse par transformée en ondelettes discrètes. Cette méthode nous permet d'évaluer le respect de la dynamique temporelle à différentes échelles de temps. Elle est utilisée pour l'évaluation des générateurs traditionnels [155, 158] qui est expliquée en détail dans [218]. Nous reviendrons en détails sur son fonctionnement dans la section 3.5.5.

Pour les champs d'en-tête, nous comparons la distribution des valeurs générées aux données originales. Ensuite, nous évaluons leur évolution au cours de la génération. Enfin,

nous étudions les variables générées entre elles afin de comparer leur cohérence avec les données originales.

3.5.2 Dynamique des paquets

Tout d'abord, nous avons essayé de vérifier si la génération de l'agrégation des paquets était cohérente d'un point de vue temporel et de sa distribution. Pour l'étude de cette partie, nous définissons un générateur d'agrégat de flux parfait A. Les flux ne sont pas générées, nous réutilisons des séquences issues des données d'apprentissage. Seuls les paquets associés à chaque flux sont générés. Pour rappel, un flux est caractérisé par sa taille en nombre de paquets et la différence de temps avec le début du flux précédent. L'agrégat de flux est étudié dans la partie 3.5.4.

Distribution de temps inter-paquet

La Figure 3.11 montre la CDF de la différence des temps d'arrivée des paquets pour chaque classe de trafic. La distribution est cohérente avec les données d'origine pour chaque classe de trafic. Dans le cas de SNMP, de très longues périodes sans communication peuvent se produire, ce qui explique la forme étirée de la CDF. Ces événements sont très rares et donc difficiles à modéliser.

Distribution des tailles de paquet

La Figure 3.12 montre la distribution des différentes tailles de paquets en octets. Le modèle a des difficultés à intégrer les phénomènes rares. En effet, certaines tailles de paquets sont élevées mais en faible proportion pour les classes de trafic modélisées. Néanmoins, la distribution reste similaire aux données originales. De plus, pour le trafic fonctionnant en mode connecté (utilisant le protocole TCP), les résultats montrent une cohérence entre le champ « flags » et la taille des paquets. En effet, lorsque la valeur du champ est « SYN » (paquet de signalisation) les paquets sont plus petits que lorsque la valeur du champ est « PUSH/ACK » (paquet de données).

Débit du trafic

La Figure 3.13 montre la dynamique temporelle du trafic de différentes classes étudiées. La dynamique temporelle est respectée malgré les différentes structures. Les différences de dynamique qui peuvent exister entre les différentes sources de données originales et les données générées sont similaires. De plus, la quantité de paquets générés pour une durée de temps similaire reste la même.

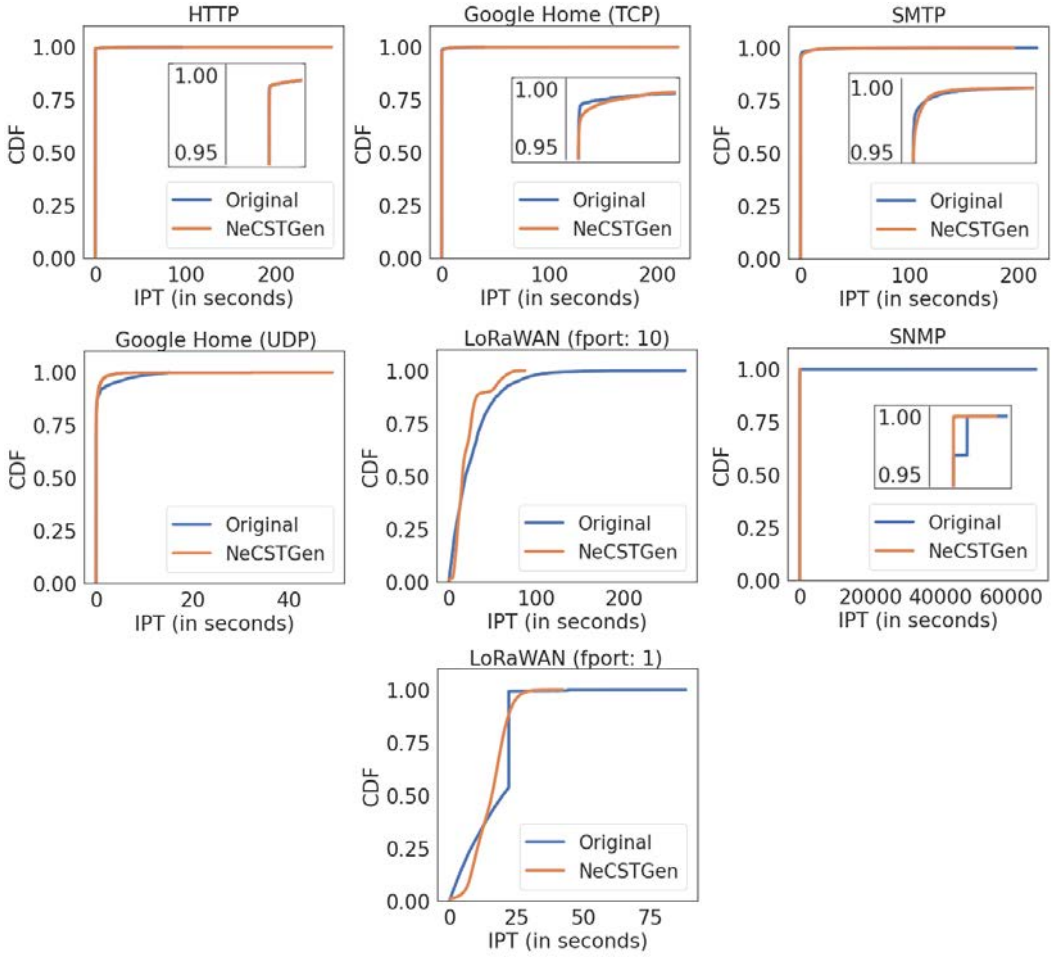

FIGURE 3.11 Cumulative Distribution Function (CDF) de la différence des temps d'arrivée des paquets en secondes pour chaque classe de trafic étudiée.

La Figure 3.14 montre la CDF des données originales et générées en octets/s. Les légères variations pour les protocoles SNMP et SMTP s'expliquent par l'existence d'une longue période d'absence de communication ou d'une courte période de forte communication. Cette variation est rare et difficile à appréhender car elle est causée par une activité humaine hautement imprévisible.

Les résultats obtenus pour la dynamique des paquets sont similaires à ceux observés dans la littérature pour les approches par apprentissage profond [196] et les générateurs traditionnels [155, 156, 158]. Cependant, nous pouvons travailler avec tous les types de trafic aux dynamiques variées. De plus, elles sont également supérieures aux approches par apprentissage profond [196] car ces dernières ne s'intéressent pas à la génération de

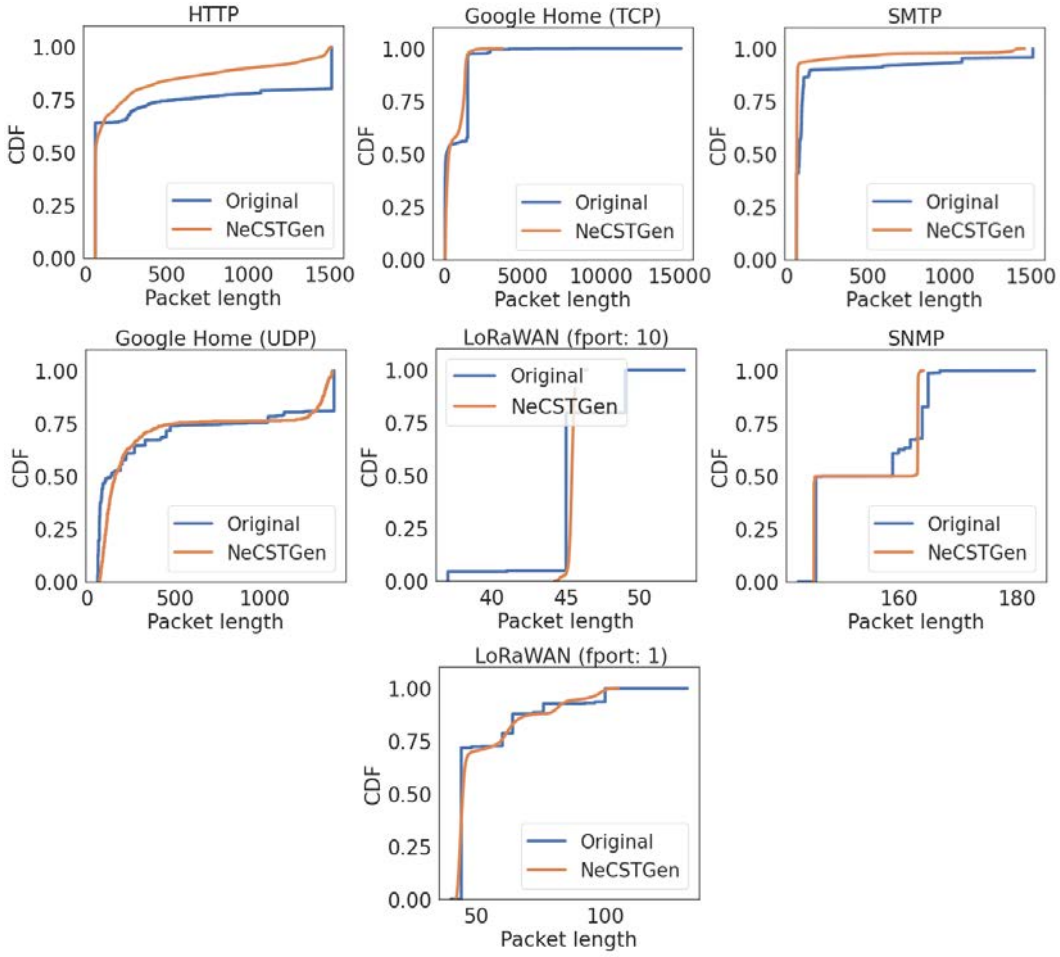

FIGURE 3.12 CDF des distributions de la taille des paquets (en octets) pour chaque protocole étudié.

séquences sur une granularité aussi fine (temps inter-paquets).

3.5.3 Champs des paquets

Cette partie se concentrera sur les valeurs des caractéristiques h_n générées. Le but est d'évaluer la cohérence des champs générés entre eux. Dans le cas du trafic LoRaWAN, nous nous intéresserons aux informations d'environnement générées et à leur cohérence avec les valeurs des variables de protocole générées.

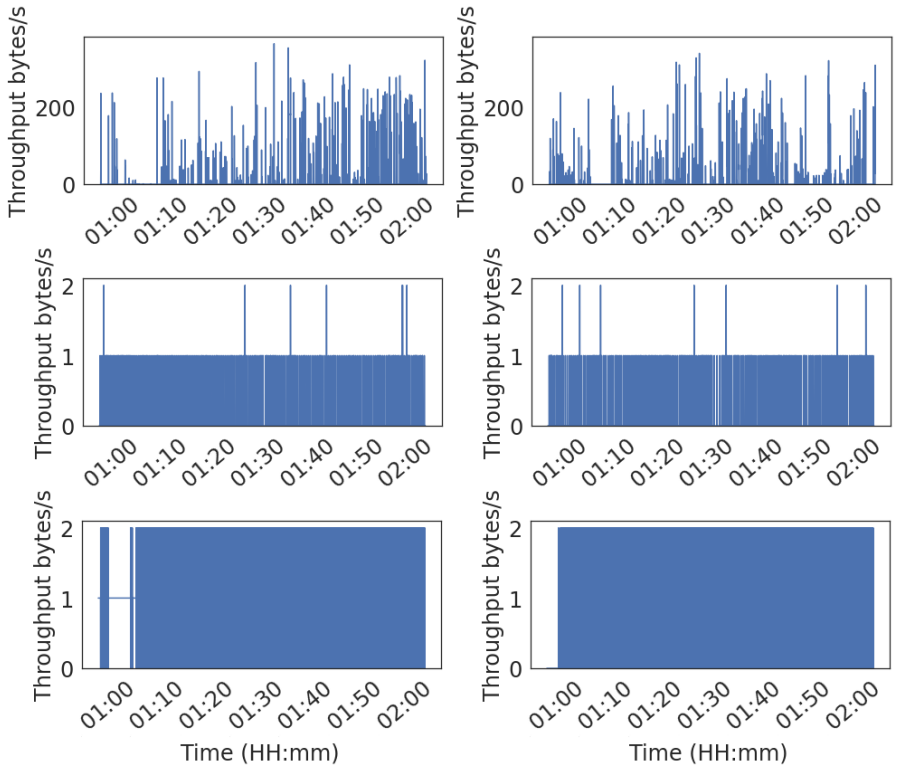

FIGURE 3.13 Débit en octets/s. De haut en bas : HTTP, LoRaWAN (champs « fport » à 1), SNMP. A gauche les données originales et à droite les données générées.

Champs

Le champs « flags » permet de délimiter le flux lors de la communication. Ainsi, nous évaluons la distribution des valeurs des champs « flags » ainsi que la cohérence de la séquence de ces valeurs lors de la génération.

La Figure 3.15 montre la distribution de la valeur du champ « flags » pour chaque classe de trafic. Nous pouvons voir que la distribution de la taille des paquets est cohérente pour chaque valeur de « flags ». De même, la direction est cohérente pour chaque position de champ. Les paquets dont le champ « flags » est défini sur « SYN » vont du client au serveur. Cette cohérence se retrouve dans l'espace latent, précédemment présenté, dans la Figure 3.5.

Nous avons également évalué la génération de séquences de champs « flags » (par exemple « SYN - SYN/ACK - ACK » dans le cas de TCP). Nous constatons une différence moyenne d'environ 5% entre la proportion originale et celle générée pour chaque séquence de valeurs de « flags ». Il peut donc y avoir une erreur dans l'identification des flux, mais

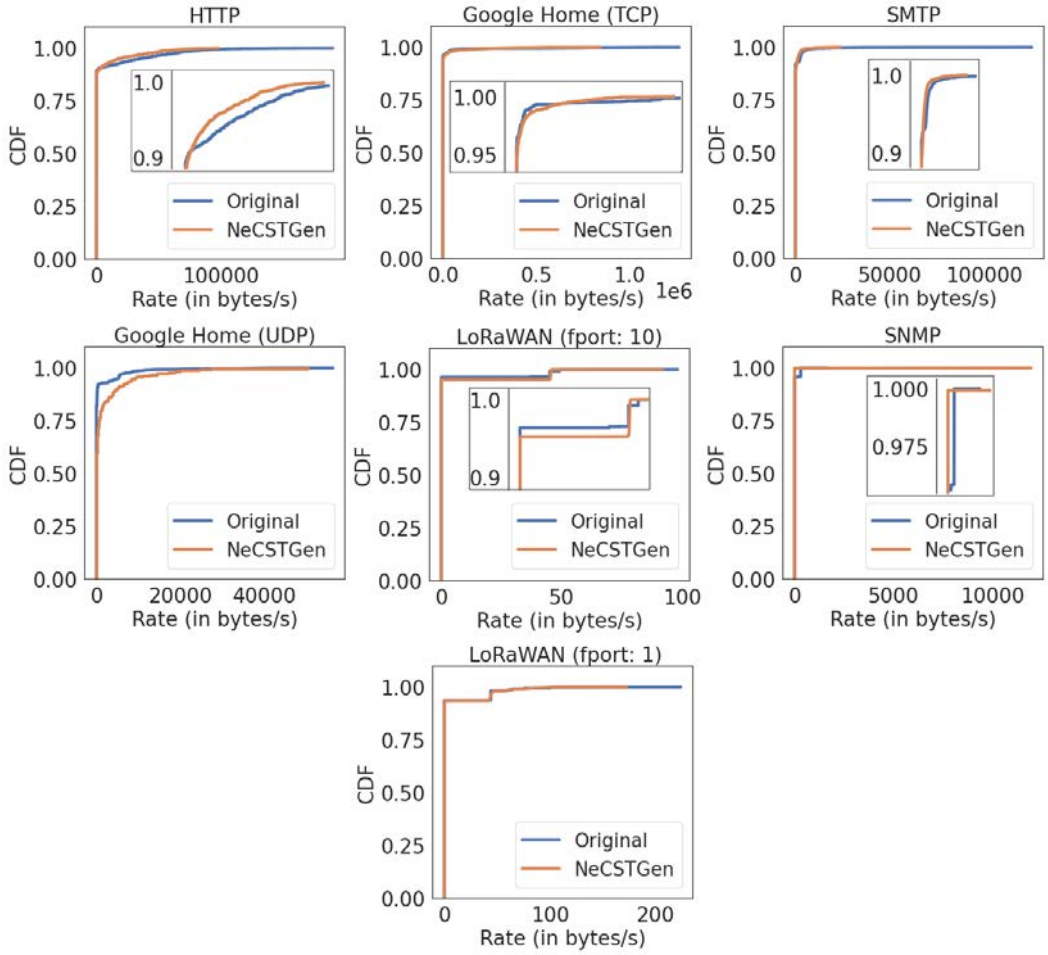

FIGURE 3.14 CDF des distributions de débit en octets/s pour chaque protocole étudié.

celle-ci est acceptable et peut être corrigée par un traitement manuel ou automatique.

Rapport signal sur bruit (SNR)

Nous générons d'autres variables liées au protocole LoRaWAN et à son environnement comme le SNR. Cette variable peut avoir un impact sur d'autres variables du protocole LoRaWAN. Dans un premier temps, nous évaluons l'impact de la variable SNR sur la variable Received Signal Strength Indication (RSSI). Dans un second temps, nous évaluons l'impact de la variable SNR sur le facteur d'étalement (« Spreading Factor »). La Figure 3.16 montre l'évolution du SNR et du RSSI sur les données originales et générées.

Nous pouvons voir sur la Figure 3.17 que la variable SNR influence la variable RSSI.

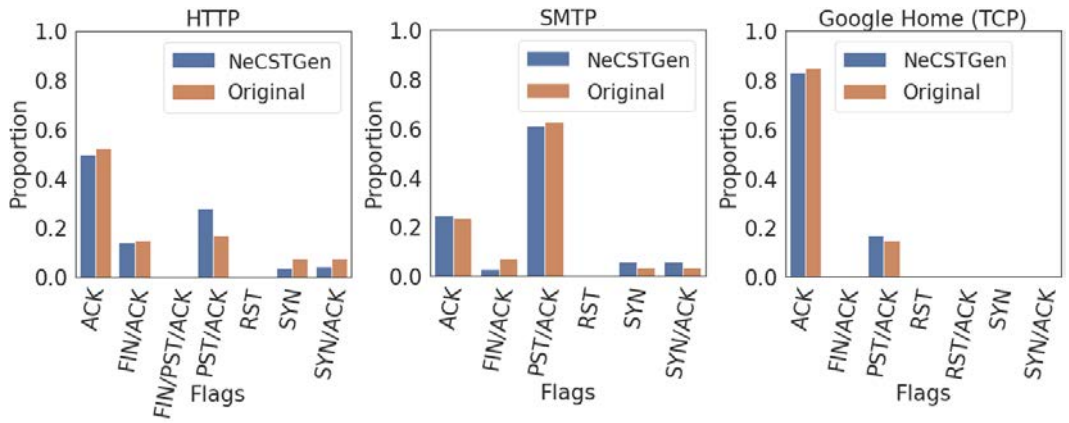

FIGURE 3.15 Diagramme en barre de répartition des valeurs du champ « flags » pour chaque classe de trafic utilisant le protocole TCP.

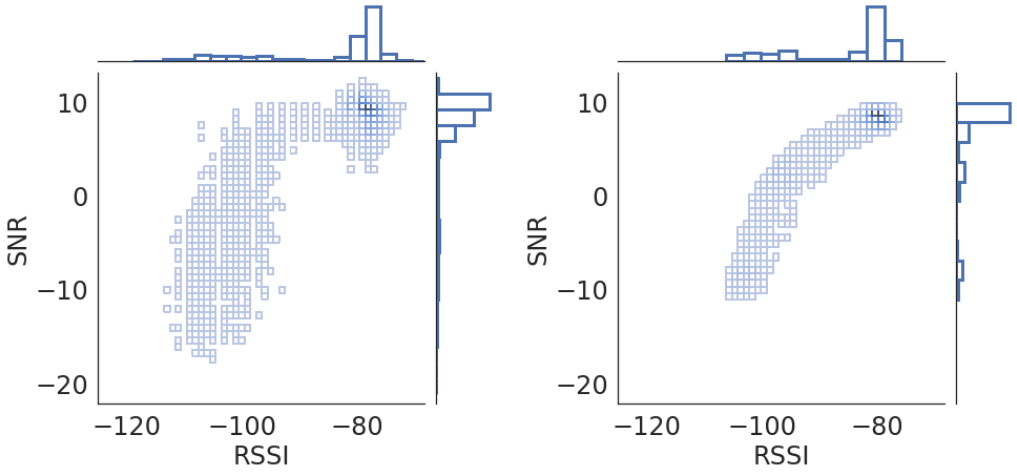

FIGURE 3.16 Diagramme de densité de l'évolution des SNR et RSSI pour le protocole LoRaWAN (« fport » à 1) : à gauche les données originales et à droite les données de NeCSTGen.

Cette influence est conservée dans les données générées. Dans un deuxième temps, nous évaluons l'impact de la variable SNR sur la variable facteur d'étalement.

L'évolution entre ces deux variables pour les données générées est cohérente avec les données originales. Notre approche est capable de générer des variables externes au protocole tout en gardant une cohérence globale avec les champs du protocole. À notre connaissance, aucune approche n'est capable de réaliser ce type de génération. Nous pouvons constater, dans le cas de LoRaWAN avec « fport » à 1, que les facteurs

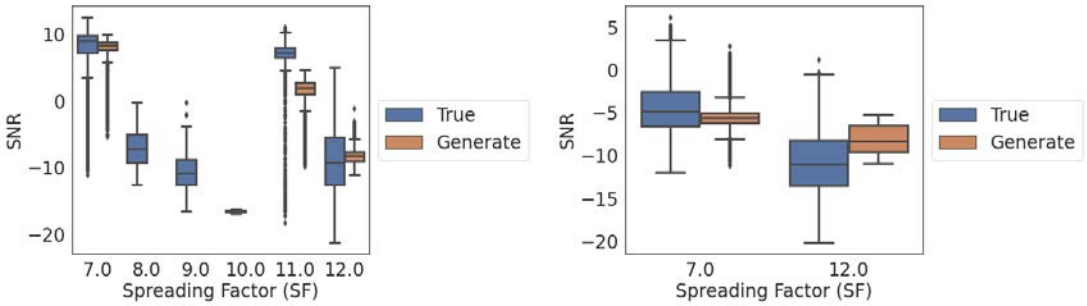

FIGURE 3.17 Diagramme de densité de l'évolution du SNR et du facteur d'étalement pour le protocole LoRaWAN avec le champ « fport » à 1 (gauche) et « fport » à 10 (droite).

d'étalement 8, 9 et 10 ne sont pas générés. En effet, ces valeurs de facteur d'étalement représentent moins de 1% des données originales. Ces valeurs sont considérées comme rares et donc difficiles à modéliser.

3.5.4 Flux et agrégat de dynamique

TABLEAU 3.14 Pourcentage de similarité entre les CDF des différentes métriques du trafic réel et du trafic généré pour chaque application.

Applications	Paquets/s	Flux/s	Temps inter-flux (IFT)
HTTP	99.9%	98.6%	99.8%
SMTP	99.9%	99.9%	99.9%
Google Home (UDP)	98.5%	99.9%	99.8%
Google Home (TCP)	99.5%	99.9%	99.8%

Le Tableau 3.14 montre le pourcentage de similarité entre la CDF des données originales et NeCSTGen des différentes métriques pour chaque type de trafic étudié en calculant Intersection Over Union (IoU). La métrique est calculée en suivant l'équation (3.12).

$$IoU = \frac{(A_{true} - A_{diff.})}{A_{true}} \text{ avec } A_{diff.} = |A_{true} - A_{necstgen}| \tag{3.12}$$

où A_{true} correspond à l'aire de la CDF des données originales et $A_{necstgen}$ l'aire de la CDF des données générées avec NeCSTGen. Un pourcentage élevé montre une forte correspondance entre le trafic original et le trafic généré. Les trafics SMTP et Google Home sont fortement influencés par l'activité humaine. Ainsi, la différence de temps entre deux flux peut être importante et interprétée comme un événement rare qui influence fortement la forme du trafic. Néanmoins, ce type d'évènements est rare et il n'impacte pas de façon importante la métrique IoU. Pour les métriques paquets/s et flux/s, les

résultats sont du même ordre à ce que nous pourrions trouver dans la littérature pour les générateurs traditionnels [155, 156, 158] et les générateurs par apprentissage profond [196].

3.5.5 Caractéristiques des échelles

Pour cette partie, nous générons l'agrégat de flux A et les flux f_n. Nous utilisons une LDE [217] pour effectuer une analyse par transformée en ondelettes discrètes.

En reprenant l'exemple de [219] nous considérons une série temporelle $X_{0,k}$ avec $k \in \mathbb{N}$, à la finesse de résolution 2^n avec $n \in \mathbb{N}$, et interprétons X_0 comme un processus de trafic réseau (ici le débit en octets par 2 millisecondes). Nous grossissons X_0 en moyennant (avec un facteur de normalisation) sur des blocs non super-positionnés de taille 2. Cette opération est illustrée dans l'équation (3.13).

$$X_{1,k} = \frac{1}{\sqrt{2}}(X_{0,2k} + X_{0,2k+1}) \tag{3.13}$$

Nous obtenons une nouvelle série-temporelle X_1, une résolution grossies de l'image original de la série X_0. Prendre la différence plutôt que la moyenne dans l'équation (3.13) résulte dans une quantité appelé « details » illustré par la formule (3.14).

$$d_{1,k} = \frac{1}{\sqrt{2}}(X_{0,2k} + X_{0,2k+1}) \tag{3.14}$$

Ainsi, il est possible de reconstruire la représentation originale de la série temporelle X_0 depuis la représentation grossie X_1 en ajoutant les « details » d_1, donc $X_0 = 2^{-1/2}(X_1 + d_1)$. Nous pouvons itérer ce processus pour toutes les échelles qui sont présentées dans la série temporelle originale. Ainsi, nous pouvons écrire $X_0 = 2^{-n/2}X_n + 2^{-n/2}d_n + ... + 2^{-n/2}d_1$. Nous nous référons à l'ensemble des « details » $d_{j,k}$ comme la série discrète de coefficient en ondelettes (Haar). Ils constituents ce que l'on appelle communément une transformation en ondelettes discrète (Haar), et elles peuvent être calculées itérativement en utilisant les équations (3.13 - 3.14).

Pour analyser les propriétés d'échelles d'une série temporelle donnée, nous utilisons une transformée en ondelette discrète (Haar). Nous examinons le comportement d'une statistique des coefficients d'ondelettes, à chaque niveau de résolution ou d'échelle. Dans cette section, nous utilisons une statistique connue qui est la fonction d'énergie E_j, défini par la formule (3.15)

$$E_j = \frac{1}{N_j}\sum_k |d_{j,k}|^2, \text{ avec } j = 1, 2, \ldots, n \text{ avec } n \in \mathbb{N} \tag{3.15}$$

où N_j est le nombre de coefficients à l'échelle j. Nous interprétons E_j comme la

moyenne de l'énergie contenue dans l'échelle j et nous examinons comment la valeur évolue lorsque nous passons d'une petite à une grand échelle. Pour cela, nous affichons $log(E_j)$ comme une fonction d'échelle j, de la plus fine à la plus importante, et nous utilisons le résultats de la fonction d'énergie affiché pour déterminer les aspects qualitatif des comportements d'échelles de la série temporelle étudiée (pour rappel, le débit).

La Figure 3.18 montre la LDE effectuée sur certaines classes de trafic étudiées. L'échelle j est affiche sur l'axe en bas et son temps correspondant (en secondes) est affiché sur l'axe du haut pour référence. Pour chaque graphe, une ligne droite montre l'absence d'échelonnement dynamique (par exemple, un processus de Poisson) et un creux la présence de périodicité dans le trafic

En général, la courbe des données générées est similaire à celle des données originales. Cependant, une différence importante est observée pour SMTP, HTTP, LoRaWAN (« fport » à 10) et Google Home. Si le trafic a beaucoup d'aléatoire ou des comportements différents, la dynamique globale n'est pas respectée. Les performances obtenues sont moins bonnes que celles des approches traditionnelles [155, 156, 158]. Ainsi, nous perdons en performance mais nous gagnons en adaptabilité.

3.5.6 QoS / QoE

La Figure 3.19 montre les CDF des RTT des différentes classes de trafic étudiées. Là encore, les événements rares sont difficiles à modéliser. Cependant, la dynamique au sein d'un flux reste très similaire pour toutes les classes de trafic. Les performances obtenues sont similaires à celles des approches classiques [155, 156, 158] mais ne sont pas étudiées pour les approches par apprentissage profond.

3.6 Conclusion

Dans ce chapitre, nous proposons NeCSTGen, une architecture reproductible pour la génération de trafic réseau réaliste. NeCSTGen permet de générer des paquets, des flux et des agrégats, ainsi que des données externes sans aucune connaissance du protocole ou de la dynamique. Chaque modèle de l'architecture vise à apprendre différents aspects de la structure de données à générer (champs, variables externes et dynamique du protocole). Les modèles sont connectés les uns aux autres afin de permettre une génération conjointe de la dynamique et des variables du protocole.

A travers les résultats, nous avons montré la polyvalence de notre approche tout en gardant une cohérence globale au sein du trafic généré. Ainsi, NeCSTGen peut être utilisé

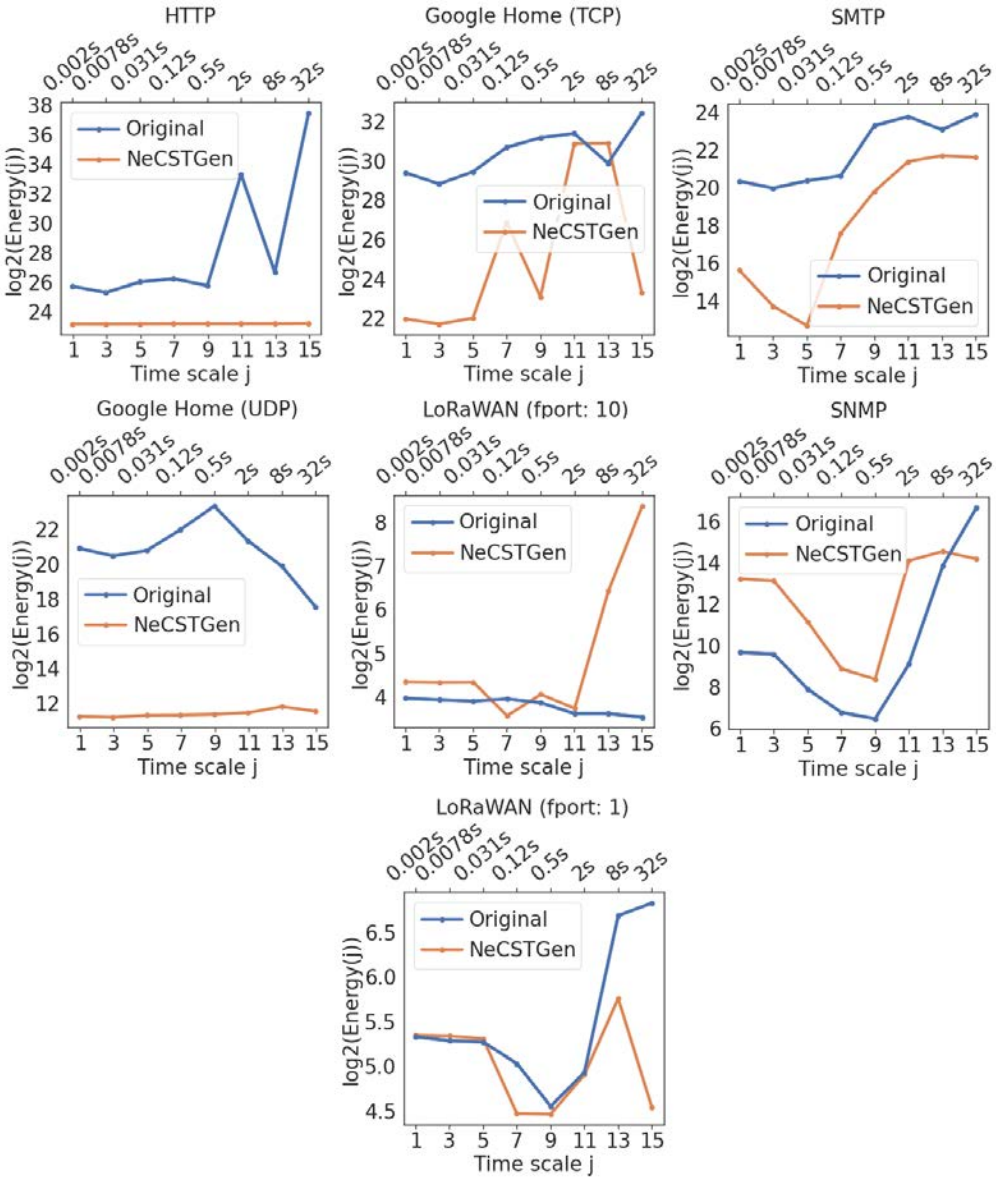

FIGURE 3.18 LDE des différents protocoles étudiés.

pour différents cadres applicatifs, quelque soit le type de trafic et de protocoles, pour le dimensionnement de réseau ou l'évaluation de performance.

Cependant, notre approche fait face à deux limites importantes. La première limite, c'est la variabilité du trafic humain ou la génération d'évènements rares, difficilement prévisibles. Être capable de simuler ce type de trafic peut permettre un gain de perfor-

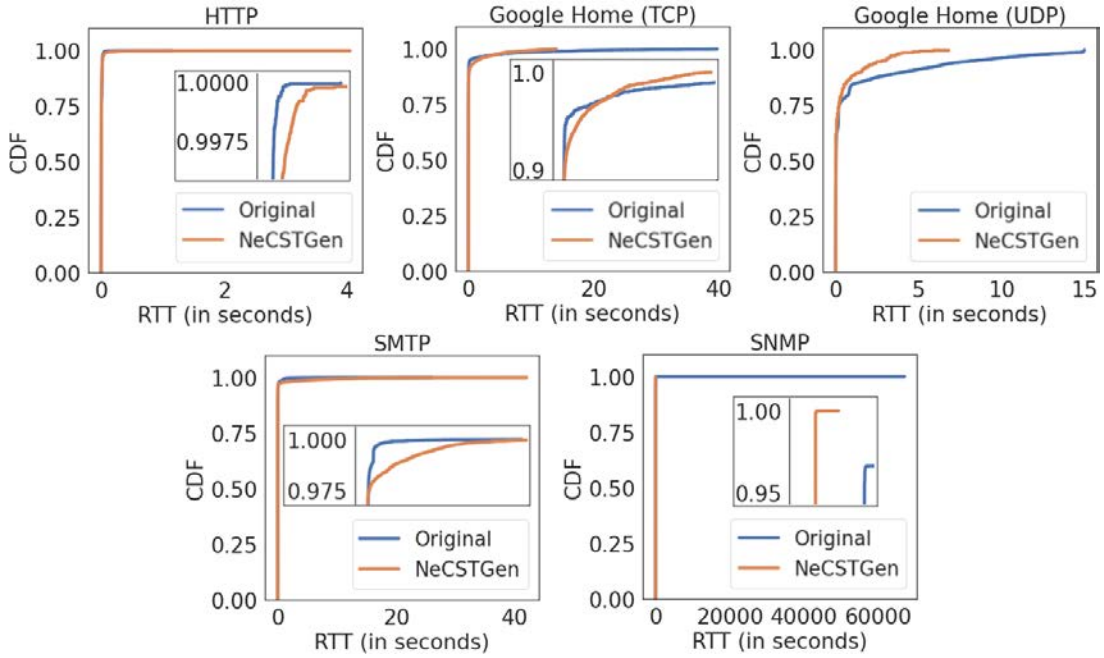

FIGURE 3.19 CDF des distributions RTT pour chaque protocole étudié.

mance et de réalisme non négligeable pour effectuer de la génération de trafic tel que la navigation internet ou le « chat » très liée au comportement humain. La seconde limite, c'est la structure du protocole utilisé. Malgré la versalité de NeCSTGen il est nécessaire de comprendre le fonctionnement et la structure du protocole afin d'extraire les caractéristiques pertinentes pour la génération. Être capable d'apprendre la structure du protocole permettrai d'identifier automatiquement les caractéristiques pertinentes pour la reconstruction d'un paquet.

Dans le prochain chapitre, nous aborderons l'apprentissage de la structure des protocoles au travers des méthodes d'apprentissage profond en l'appliquant à la compression d'en-têtes.

Chapitre 4

Compression universelle des en-têtes pour l'IoT

Sommaire

4.1 Introduction

Dans ce chapitre, nous étudions la compression des en-têtes pour l'IoT. Nous montrons les limites des approches actuelles et nous proposons une alternative universelle, par apprentissage profond, pour la compression des en-têtes de trafic réseau.

Le développement rapide des réseaux et la quantité toujours croissante de données qu'ils doivent transporter ont créé à la fois des problèmes et des opportunités. Dans un premier temps, les méthodes de compression traditionnelles ont été utilisées pour faire face au faible débit des réseaux. Aujourd'hui, les appareils IoT sont confrontés aux mêmes problèmes, auxquels s'ajoutent des contraintes telles que la consommation d'énergie et la puissance de calcul. De plus, les équipements IoT transportent, le plus souvent, une quantité de données plus faibles (par exemple une mesure de températures sur 8 bits) que la taille de l'en-tête (comprenant de l'Internet Protocol Version 6 (IPv6) avec Message Queuing Telemetry Transport (MQTT) ou Constrained Application Protocol (CoAP)) qui permet de transporter les données.

Dans ce chapitre, notre objectif, est d'effectuer la compression d'en-têtes réseau en surmontant les limites des approches actuelles :

— Fonctionner sur tous les types de protocoles sans connaissance préalable de leur structure n'est pas possible.
— Fonctionner sur différentes topologies (étoile, maillé). Cela n'est pas toujours possible avec toutes les approches actuelles de compression des en-têtes.
— Pouvoir compresser l'en-tête et les données.

Pour surmonter les limitations précédentes, nous introduisons une nouvelle méthode de compression sans perte, appelée DCH. Notre objectif est de définir un outil polyvalent capable de compresser l'en-tête de n'importe quel type de trafic réseau ; avec l'aide de techniques d'apprentissage profond, DCH peut fonctionner sans aucune connaissance de la structure du protocole.

Pour démontrer son adaptabilité, nous testons notre approche sur plusieurs ensembles de données : DARPA [213], MQTT [220], LoRaWAN [215] et réseau CoAP. Nous fournissons également le code source [1] pour une expérience reproductible. Nous montrons qu'il est possible de mettre en place notre approche sur un équipement IoT pour compresser les en-têtes de n'importe quel type de trafic sans connaissance préalable sur la structure du protocole.

Dans une première partie, nous présentons les différents travaux qui existent dans le domaine de la compression d'en-tête de trafic réseau. Dans une seconde partie, nous

1. Code source : `https://github.com/fmeslet/DCH`

formalisons notre approche et nous présentons son fonctionnement dans un cadre général. Dans une troisième partie, nous discutons de l'architecture développée, nommée DCH, et des différents problèmes qu'elle vise à résoudre. Dans une quatrième partie, nous montrons la transformation de modèle effectuée afin de rendre embarquable, sous forme de table, le modèle développé par apprentissage profond. Enfin, dans une dernière partien nous présentons les performances de DCH et nous les comparons avec les approches actuelles.

4.2 États de l'art

4.2.1 Introduction

La compression d'en-tête est, comme son nom l'indique, une technique de compression qui vise à réduire la taille de l'en-tête d'un paquet véhiculé sur un réseau informatique. Les objectifs de la compression d'en-tête sont multiples : en véhiculant moins de bits nous pouvons, réduire l'utilisation de la bande passante [221], réduire la probabilité d'erreur (car moins de bits sont transmis), réduire le délai [222] et réduire la consommation d'énergie [223].

Une communication sur un réseau informatique implique une ou plusieurs parties prenantes, un ou plusieurs émetteurs et un ou plusieurs récepteurs. Lors de la communication, il est nécessaire d'effectuer l'opération inverse à la compression : la décompression. Pour cela, le processus de compression d'en-tête comprend deux parties : un compresseur et un décompresseur. Le rôle du compresseur, noté C, est de réduire la taille de l'en-tête d'un paquet, noté H. Le rôle du décompresseur, noté D, est de prendre en entrée l'en-tête du paquet compressé H^c et retrouver l'en-tête du paquet original H. La formule (4.1) présente la compression et la décompression.

$$\begin{aligned} C(H) &= H^c \\ D(H^c) &= H \end{aligned} \tag{4.1}$$

La plupart des techniques de compression vont considérer comme faisant partie de l'en-tête, les protocoles appartenant aux niveaux 2 et 3 du modèle OSI tels que UDP/TCP, IP.

Pour réduire la taille de l'en-tête, le compresseur peut utiliser deux sources d'informations : le contenu de l'en-tête à compresser et le contexte. Le contexte, noté S, est une information précédemment reçue qui permet d'accroître la connaissance de la structure de l'en-tête d'un paquet et donc améliorer ses performances de compression et décompression. Le contexte peut, par exemple, être la valeur d'un champs des paquets précédemment

reçu. Ainsi, dans le cadre d'une compression avec contexte, la formule (4.1) devient la formule (4.2).

$$\begin{aligned} C(H,S) &= H^c \\ D(H^c,S) &= H \end{aligned} \tag{4.2}$$

L'utilisation de contexte introduit deux problématiques : la désynchronisation et l'identification de contexte. La désynchronisation se définit par la perte de cohérence entre le contexte utilisé par le compresseur et celui utilisé par le décompresseur. En l'absence de cohérence, la décompression ne peut pas se dérouler correctement. Le second problème c'est l'identification. La forme de contexte utilisée doit être définie, elle peut être associée à un flux, une application, un équipement, etc. Dans le cas où le contexte est associé à un flux, il peut être nécessaire de rajouter un identificateur de flux qui permet au décompresseur de déterminer le contexte à associer au paquet à decompresser.

Le processus de compression nécessite l'utilisation de règle, d'algorithmes qui vont permettre de spécifier les opérations à effectuer pour la compression et la décompression d'un paquet. Dans le cadre de la compression d'en-têtes, ces techniques doivent permettre une compression et décompression sans perte. Pour la compression d'en-têtes, des techniques spécifiques ont été développées. Néanmoins, d'autres approches de compression de données sans perte existent et pourraient être envisagées dans un cadre réseau. Ce sont les approches de compression sans perte, générique, que les approches par apprentissage profond arrivent à concurrencer. Ces méthodes seront présentées dans les sections suivantes.

4.2.2 Méthodes de compression d'en-têtes traditionnelles

Synthèse

Le Tableau 4.2 extrait de [14] fait un résumé des différentes méthodes de compression standardisées qui existent à ce jour.

Tableau 4.1 Profils de compression de RoHC [13].

ID du profil	**Protocoles compressés**	**v1 RFC (Année)**	**v2 RFC (Année)**
0x00	Non compressé	-	3095 (2001)
0x01	Real-time Transport Protocol (RTP)/ UDP/IP	3095 (2001)	5225 (2008)
0x02	UDP/IP	3095 (2001)	5225 (2008)
0x03	Encapsulating Security Payload (ESP)/ IP	3095 (2001)	5225 (2008)
0x04	IP	3843 (2004)	5225 (2008)
0x06	TCP/IP	-	4996 (2007)
0x07	RTP/UDP-Lite/IP	4019 (2005)	5225 (2008)
0x08	UDP-Lite/IP	4019 (2005)	5225 (2008)

Tableau 4.2 Résumé des méthodes de compression d'en-tête standardisé extrait de [14].

Nom	**RFC**	**Année**	**Service**	**Protocoles**
Thinwire I [224]	914	1984	Filaire	TCP/IP
Thinwire II [224]	914	1984	Filaire	TCP/IP
CTCP [225]	1144	1990	Filaire	TCP/IP
CIPX [226]	1553	1993	Filaire	Network Control Protocol (NCP)/ Internetwork Packet Exchange (IPX), IPX
CRTP [227]	2508	1999	Filaire	RTP/UDP/ Internet Protocol Version 4 (IPv4), UDP/IPv4
IPHC [228]	2507	1999	Sans fils	TCP/IP, UDP/IP, IPv4, IPv6, implémentation compatible pour CRTP [227]
RoHCv1 [229]	3095	2001	Sans fils	RTP/UDP/IP, UDP/IP, ESP/IP, pour plus de profils voir le Tableau 4.1
ECRTP [230]	3545	2003	Sans fils	RTP/UDP/IP, UDP/IP
RoHCv2 [231]	5225	2008	Sans fils	RTP/UDP(-Lite)/IP, UDP(-Lite)/IP, ESP/IP, TCP/IP, IPv4, IPv6
6LoWPAN [232]	6282	2011	LoWPWAN	IPv6
HPACK [233]	7541	2015	Filaire/ Sans fils	HTTP
SCSH [234]	8724	2020	LPWAN	UDP/IP, Générique

Les méthodes traditionnelles

Thinwire I [224] et II [224] sont les premières approches de compression d'en-tête conçues pour les réseaux Telnet. Le compresseur et décompresseur stockent les paquets précédents dans le but de pouvoir décompresser des transmissions partielles sur la base de la réception de valeurs de champs modifiées qui sont transmises en décalées. Ils établissent la notion de contexte. CTCP [225] reprend ce principe en intégrant la notion de « delta », qui exploite la différence entre deux paquets. CRTP [227] applique le principe de compression du « delta » de manière variable sur le protocole RTP. Robust Checksum-Based Compression (ROCCO) [235] met à jour CRTP [227] en affinant la compression de l'en-tête sur les liaisons fortement sujettes aux erreurs avec de longs temps d'aller-retour. Elle a été remplacée par la version ultérieure Robust Header Compression Version 1 (RoHCv1). Ensuite, ECRTP [230] est introduit comme un raffinement de CRTP [227] et une amélioration de RoHCv1 [229]. Il peut fonctionner avec des transmissions en temps réel même avec des taux élevés de perte de paquets, de réorganisation et de longs délais. Une amélioration majeure est apportée avec l'arrivée de IPHC [228]. Il est plus robuste aux erreurs de désynchronisation de contexte et de décompression. D'autres techniques de compression plus spécifiques ont vu le jour, comme HPACK [233]. HPACK [233] a été défini comme une alternative sécurisée à DEFLATE et intégrée à HTTP/2 [236]. Les champs d'en-tête redondants dans les requêtes HTTP consomment inutilement de la bande passante et augmentent la latence. HPACK [233] tente de surmonter cette limite grâce au code Huffman et une table de compression associée aux différents champs HTTP. Une autre technique, appelée IPv6 over Low-power Wireless Personal Area Networks (6LoWPAN) [232] a également été développée. Son but est de réduire la surcharge des en-têtes IPv6 pendant la transmission sur les réseaux à faible puissance. La dernière approche normalisée est appelée Static Context Header Compression (SCSH) [234]. Le but de cette compression est de définir un ensemble de règles partagées entre l'émetteur et le récepteur. Ces règles, étiquetées par un ID, vont définir les modifications à apporter aux en-têtes à transmettre.

4.2.3 Méthodes de compression par apprentissage profond

4.2.4 Synthèse

Le Tableau 4.3 effectue une synthèse des techniques de compression sans perte par apprentissage profond utilisées. Ces techniques ne sont pas spécialisées à la compression d'en-têtes car il n'existe pas, à notre connaissance, de méthode spécifique à ce domaine.

Tableau 4.3 Résumé des méthodes de compression sans perte par apprentissage profond.

Nom	**Type de données**	**Description**	**Année**
CMIX [237]	Exécutables binaire, texte, images	Basé sur une combinaison de modèles utilisant l'Arithmetic Coding (AC) [238]. Il vise à optimiser le taux de compression au prix d'une utilisation CPU/mémoire élevée.	2021
NNCP [239]	Texte	Basé sur un modèle Transformer [209] XL, permet d'obtenir de meilleurs taux de compression au prix d'un coût de calcul plus élevé.	2021
LSTM-Compress [240]	Texte	Ce projet utilise le même code de LSTM et de prétraitement que CMIX [237]. Tous les autres modèles de CMIX [237] sont supprimés, de sorte que la compression soit effectuée en utilisant uniquement le LSTM.	2019
DecMac [241]	Texte	Combine un LSTM à trois couches utilisant l'AC [238] pour une compression sans perte.	2019
Latent variables bit-back coding [242]	Image	Utilise le Bits-Back avec Asymmetric Numeral Systems (BB-ANS), un schéma permettant d'effectuer une compression sans perte avec des modèles de variables latentes.	2019
Bitswap [243]	Image	Un nouveau schéma de compression qui généralise le Bits-Back avec Asymmetric Numeral Systems (BB-ANS) en utilisant des modèles de variables latentes hiérarchiques avec une structure de chaîne de Markov.	2019
DeepZip [244]	Texte, génomiques	Combine des RNN et l'AC [238] pour la compression sans perte de données synthétiques variées.	2018
Fast Text Compression [245]	Texte	Commpression avec un réseau de neurones par maximimisation de l'entropie au niveau bit.	2000
Sequential Neural Text [246]	Texte	Combiner les réseaux neuronaux prédictifs et les techniques de codage statistique pour compresser les fichiers texte.	1996

La plupart des modèles présentés ici peuvent fonctionner sur tous types de données. Cependant, certaines ont été conçues ou évaluées sur une catégorie de données spécifiques répertoriées dans la colonne « Type de données » du Tableau 4.3.

4.2.5 Méthodes par apprentissage profond

La possibilité de réaliser une compression sans connaissance préalable de la structure des protocoles à compresser peut être rendue possible par les approches par apprentissage profond. Les approches de compression sans perte par apprentissage profond ont commencé à faire leurs preuves pour la compression de données et à surpasser les approches traditionnelles telles que : Gzip [2], BSC [3], JPEG2000, PNG, etc. Parmi ces approches, nous trouvons : LSTM-Compress [240], NNCP [239] et DecMac [241] et plusieurs autres approches [245, 246]. La plupart sont spécifiques à une catégorie de données (par exemple, le texte [241] ou les images [242, 243]). Certaines de ces méthodes, telles que [242, 243], utilisent une approche de codage en « bit-back » et peuvent introduire un « overhead » importante pour la compression de petites quantités de données. D'autres approches plus universelles sont apparues introduisant une surcharge moins importante comme DeepZip [244] basée sur l'AC [238], une technique dérivée du codage de Huffman.

4.2.6 Limites des approches actuelles

Les méthodes de compression traditionnelles ne peuvent pas fonctionner sur plusieurs protocoles sans adaptation préalable. Dans le cas de SCSH [234], une adaptation est nécessaire pour chaque protocole à déployer. Les autres méthodes telles que 6LoWPAN [232] sont spécifiques à un protocole en particulier (IPv6). Les approches actuelles ne peuvent donc pas gérer l'arrivée de nouveaux protocoles ou la variété des protocoles présents dans l'IoT. Enfin, il n'est pas possible pour toutes les approches de travailler sur des topologies variées ou dynamiques.

Les techniques de compression sans perte par apprentissage profond offrent la possibilité de compresser des types de données variées. La phase d'apprentissage leurs permet d'appréhender finement la structure des données à compresser afin d'obtenir des performances supérieures aux approches traditionnelles. Néanmoins, il n'existe pas, à notre connaissance, d'approche permettant de réaliser de la compression d'en-tête de trafic réseau. De plus, Une limite importante des approches par apprentissage profond provient de la puissance de calcul et de la mémoire nécessaire pour les faire fonctionner. Les

2. `https://www.gzip.org`
3. `http://libbsc.com`

ressources présentes dans la majorité des équipements réseaux ne sont pas suffisantes pour utiliser directement ce type d'approche, une adaptation est nécessaire.

4.3 Présentation

Dans cette section, nous définissions une formalisation de notre approche ainsi que le fonctionnement générale de l'AC. La formalisation posera les bases du fonctionnement de notre approche DCH pour la compression universelle d'en-tête de trafic réseau.

4.3.1 Notations

TABLEAU 4.4 Principales notations utilisées pour les chapitres traitant de la compression d'en-têtes réseau.

Nom	Brève description
n	Le nombre de paquets (et d'en-têtes)
H_k	En-tête du numéro de paquet $k \in [0, n-1]$
H_k^c	En-tête compressé du numéro de paquet k
$H_{k,i}$	Bit i de H_k
L	Longueur de l'en-tête en bits
l	Nombrer de bits extraits de H_k (fenêtre glissante)
l_C	Longueur du contexte
C_k	Contexte utilisé pour H_k compression, $C_k = \{H_{k-l_C}, \ldots, H_{k-1}\}$
M^i	Masques utilisés pour le bit i, $M^i = \{m^i_{l_C}, \ldots, M^i_0\}$
m^i_j	Masque appliqué sur H_{k-j} dans le contexte C_k
δ^i_k	Bits extraient depuis C_k et H_k après application des masques M^i

L'ensemble des notations utilisées dans ce chapitre est résumé dans le Tableau 4.4. L'objectif de la compression d'en-têtes est de prendre en entrée un ensemble de n en-têtes de paquets, notés H_k, où $k \in [0, n-1]$ et de produire en sortie un ensemble d'en-têtes compressés, notés H_k^c. Les en-têtes sont considérés comme étant de longueur constants, notée L, et H_k sera donc représenté par une séquence de L bits $H_{k,i}$ avec $i \in [0, \ldots, L-1]$ et $i \in \mathbb{N}$.

La compression de l'en-tête H_k est réalisée à l'aide d'un contexte, noté C_k. Plusieurs approches de compression d'en-tête telles que Robust Header Compression (RoHC) [229, 231] utilisant également un contexte pour améliorer l'efficacité. Différents types d'informations peuvent être incluses dans le contexte : par exemple, SCSH [234] est basé sur la définition de certaines *règles*.

Le contexte doit être partagé entre l'expéditeur (pour la compression) et le destinataire (pour le processus de décompression), ce qui signifie que les deux entités doivent stocker les données qui composent le contexte.

Notre objectif est de définir un algorithme de compression qui puisse être mis en œuvre sur des appareils IoT. En raison de la capacité limitée de ces appareils, nous voulons limiter le contexte aux informations disponibles. L'introduction de toute donnée supplémentaire afin d'améliorer l'efficacité de la compression entraînerait une surcharge de transmission et de stockage.

Nous définissons donc un contexte comme un ensemble d'en-têtes de paquets qui précèdent l'en-tête de paquet compressé. La compression d'un en-tête H_k en H_k^c est un processus continu bit par bit, mais le même contexte C_k est utilisé tout au long de la compression de H_k. Le nombre d'en-têtes contenus dans un contexte est noté l_C avec $l_C \in \mathbb{N}$. La formule (4.3) représente le concept de contexte.

$$C_k = \{H_{k-l_C}, \ldots, H_{k-1}\} \text{ avec } k, l_C \in \mathbb{N} \tag{4.3}$$

Pour des raisons de cohérence, H_k est défini pour $k \in \mathbb{Z}$, les en-têtes H_k pour $k < 0$ sont considérés comme nuls (tous les bits à 0).

Nous supposons également que chaque bit d'un en-tête est fortement corrélé avec certains bits précédents dans le même en-tête, noté $H_{k,i-j}, \ldots, H_{k,i-1}$ avec $j \in [0, l]$ et $l \in \mathbb{N}$. L'opération de compression prend donc en entrée le contexte C_k et les bits $\{H_{k,i-l}, \ldots, H_{k,i-1}\}$.

4.3.2 Aperçu général

L'objectif de cette section est de donner un aperçu général de DCH. Le cœur de notre proposition est l'utilisation de l'apprentissage profond avec AC pour la compression des en-têtes, que nous décrirons brièvement dans 4.3.2. En tant que technique de codage entropique, AC est basée sur les probabilités d'occurrence des symboles. Conformément à notre hypothèse selon laquelle chaque bit d'un en-tête est corrélé à son contexte (tel que défini précédemment), nous décrirons dans 4.3.2 comment nous déterminons ces probabilités. Fondamentalement, cela se fait à l'aide de masques binaires.

L'une des principales contributions de ce chapitre est l'utilisation des « Occlusion Maps » [8] sur un modèle d'apprentissage profond pour construire ces masques binaires et les probabilités associées, que nous décrirons en détail dans la section suivante.

Une autre contribution importante est la mise en œuvre de DCH dans un environnement réel pour son évaluation, que nous décrirons dans la section 4.5.

Arithmetic coding

L'idée de base sur laquelle repose AC peut être illustrée par la Figure 4.1. Étant donné un ensemble de symboles (`A`, `B` et `C` dans cet exemple), l'objectif est de coder un message (*eg* `ABC`) sous la forme d'un nombre réel dans l'intervalle $[l, h]$ (*eg* $l = 0.0, h = 1.0$).

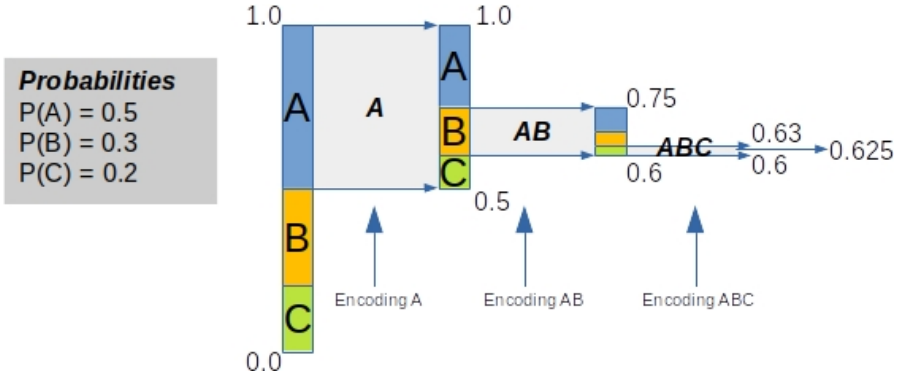

FIGURE 4.1 Codage du message ABC par AC.

Sur la base des probabilités d'occurrence, n'importe quel nombre de l'intervalle $[0.5, 1.0]$ peut être utilisé pour coder le message `A`. En répétant ce processus deux fois dans chaque nouvel intervalle, la Figure 4.1 montre que le message `ABC` peut être codé sous la forme d'un nombre quelconque dans l'intervalle $[0.6, 0.63]$.

Remarquez que nous avons utilisé les mêmes probabilités à chaque étape, mais ce processus est *adaptatif*, avec des probabilités différentes à chaque étape, tant que ces probabilités sont connues.

L'étape suivante consiste à choisir un nombre dans cet intervalle qui peut être encodé avec la séquence la plus courte. En supposant que l'intervalle initial est $[0.0, 1.0]$, nous n'avons besoin de représenter que la partie fractionnaire de ce nombre. En utilisant la base 2, la séquence la plus courte pour notre exemple est 101, ce qui correspond à $0.101_2 = 0.625_{10}[0.6, 0.63]$.

Bien entendu, pour compresser de longues séquences, plusieurs questions doivent être abordées, comme décrit par exemple dans [4]. Toutefois, le concept sous-jacent demeure : sur la base des probabilités d'occurrence, AC prend un flux de symboles (qui peut être un message ou un en-tête de paquet) en entrée et produit un flux de sortie compressé sans perte.

4. `https://marknelson.us/posts/2014/10/19/data-compression-with-arithmetic-coding.html`

Dans le présent document, les symboles sont des bits, de sorte que nous avons besoin d'une seule probabilité à chaque étape : la probabilité que la valeur du bit suivant soit 1.

Calcul des probabilités

L'une des hypothèses fondamentales de notre proposition est que chaque bit $H_{k,i}$ de l'en-tête à compresser est corrélé avec un nombre limité de bits dans le contexte et dans les bits précédents de $H_{k,i}$. Par conséquent, nous utiliserons notre connaissance de ces bits pour déterminer la probabilité de chaque bit de H_k. Ces bits sont issus du contexte C_k et des bits précédents du même en-tête : $\{H_{k,i-j}, \ldots, H_{k,i-1}\}$ avec $j \in [0, l]$ et $l \in \mathbb{N}$. l représente la taille de la fenêtre glissante appliquée à l'en-tête. Cette fenêtre est liée au fonctionnement de AC.

Pour mettre en évidence les bits les plus pertinents, nous utiliserons des masques binaires. Pour une position de bit donnée i dans un en-tête, nous définirons un masque M^i qui sélectionne, dans le contexte et dans les bits précédents du même en-tête, les bits avec lesquels la valeur du bit i est la plus corrélée. La construction de ces masques est une tâche complexe, et nous décrirons notre proposition dans la section 4.4.

Le masque M^i peut être considéré comme un ensemble de masques : $M^i = \{m^i_{l_C}, \ldots, m^i_0\}$; l'application du masque M^i au contexte C_k (calcul de $M^i \wedge \{C_k, \{H_{k,i-j}, \ldots, H_{k,i-1}\}\}$) pourrait être mise en œuvre comme l'application du masque m^i_p à l'en-tête $\{H_{k-p}, \ldots, H_k\}$ pour chaque $p \in \{l_C, \ldots, 0\}$. La formule (4.4) illustre ce concept.

$$\begin{aligned} &M^i \wedge \{C_k, \{H_{k,i-l}, \ldots, H_{k,i-1}\}\} \\ &= (m^i_{l_C} \wedge H_{k-l_C}, \ldots, m^i_0 \wedge \{H_{k,i-l}, \ldots, H_{k,i-1}\}) \end{aligned} \tag{4.4}$$

Il faut rappeler ici que les bits sont traités séquentiellement (compressés chez l'expéditeur et décompressés chez le destinataire). Par conséquent, lorsque le bit i de l'en-tête k ($H_{k,i}$) est traité, seules les valeurs des bits $H_{k,j}$ avec $j \in \{0, \ldots, i-1\}$ sont connues et peuvent donc être utilisées pour calculer les probabilités par corrélation. Il s'ensuit que le masque m^i_0 ne peut avoir des valeurs non nulles que pour les positions $< i$. Bien entendu, chaque bit des en-têtes précédents est connu, de sorte que tout bit de m^i_j peut être non nul pour $p \in \{1, \ldots, l_C - 1\}$.

En supposant que le masque M^i a un poids constant (le nombre de bits non nuls) w, nous pouvons représenter la sortie de cette opération de masque sur le contexte comme un ensemble de w bits. Appelons cet ensemble δ^i_k représenté dans la formule (4.5).

$$\delta^i_k = \{H_{k-p,i} | m^i_{k-p,i} = 1, p \in [0, l_C], i \in [0, L-1]\} \tag{4.5}$$

En supposant que le masque M^i ait été défini avec précision pour sélectionner les bits du contexte et des fenêtres glissantes contenant $\{H_{k,i-j}, \ldots, H_{k,i-1}\}$, qui ont la plus forte corrélation avec le bit i de l'en-tête. La valeur réelle de ces bits est conservée dans δ_k^i. Par conséquent, cette valeur peut être utilisée pour déterminer une probabilité pour le bit i illustrée par la formule (4.6).

$$P(H_{k,i}) = T(\delta_k^i) \tag{4.6}$$

L'une des principales contributions de ce travail, décrite dans la section 4.5, est consacrée à la recherche de l'ensemble des masques M^i (pour $i \in \{0, \ldots, L-1\}$) qui sélectionnent les bits les plus appropriés du contexte, et à la construction d'une table T qui détermine les probabilités.

4.3.3 Fonctionnement de DCH

Nous pouvons maintenant décrire le cadre général de notre proposition de DCH, qui vise à définir une solution complète pour la compression sans perte des en-têtes pour les applications IoT. DCH comporte deux étapes de fonctionnement :

— Une étape « hors ligne » : l'objectif est d'apprendre un modèle d'apprentissage profond pour la compression des en-têtes de réseau.

— Une étape « en ligne » : l'objectif est d'interpréter et de transformer le modèle appris dans la phase « hors ligne » et de le déployer sur les capteurs.

Dans la phase « hors ligne », un modèle d'apprentissage profond apprend à prédire la probabilité $P(H_{k,i})$ à partir de données collectées sur le réseau où le système de compression doit être déployé. L'apprentissage a lieu sur une infrastructure informatique distincte. Pendant la phase d'apprentissage, le modèle d'apprentissage profond apprend les masques M^i, pour chaque bit à la position i, à appliquer au contexte C_k et aux bits appartenant à la fenêtre glissante de taille l ($\{H_{k,i-j}, \ldots, H_{k,i-1}\}$).

Dans l'étape « en ligne », le modèle d'apprentissage profond est interprété et transformé en une table, appelée T, qui est mise en œuvre sur les appareils IoT. Cette table donne la probabilité $P(H_{k,i})$ pour chaque paquet à compresser selon la formule (4.6). Les modèles d'apprentissage profond ont la particularité d'être difficiles à expliquer. L'interprétation, étudiée dans la section 4.5.2, nous permettra d'extraire les δ_k^i bits qui auront le plus d'impact sur la probabilité $P(H_{k,i})$. Grâce aux δ_k^i bits extraits, la table T peut être construite puis implémentée sur un capteur IoT pour effectuer la compression et la décompression directement sur celui-ci. Il s'agit d'un élément fondamental de notre technique.

Après le déploiement sur l'appareil IoT, le processus de compression se déroulera comme suit :

— L'expéditeur détermine la probabilité $P_{k,i}$ de l'occurrence de chaque bit $H_{k,i}$ de l'en-tête H_k à l'aide de la table T ;
— Ces probabilités sont utilisées pour construire une séquence binaire $H^c_{k,i}$ par AC ;
— Cette séquence binaire est l'en-tête H^c transmis.

La compression est effectuée à partir de la couche réseau incluse dans le modèle, comme c'est le cas pour la plupart des approches actuelles [232, 234]. En pratique, d'autres informations doivent être ajoutées pour permettre la décompression, comme un identifiant de flux (pour la synchronisation du contexte) et un champ indiquant la taille finale de l'en-tête du paquet à décompresser. En effet, si nous ne connaissons pas la taille finale, nous ne pouvons pas savoir quand arrêter la décompression.

4.4 Deep Compression Header (DCH)

Dans cette section, nous introduisons les jeux de données utilisés pour l'apprentissage et l'évaluation de DCH. Ensuite, nous abordons les différents traitements appliqués sur nos données avant l'apprentissage. Enfin, nous présentons l'architecture de DCH et les différents éléments qui la composent.

4.4.1 Jeux de données

Quatre jeux de données ont été utilisés pour notre modélisation. Nous avons choisi ces jeux de données car ils nous donnent accès à une variété de protocoles. Ils nous permettront donc de vérifier la généricité de notre approche. Le Tableau 4.5 présente les jeux de données utilisés et leurs protocoles.

Tableau 4.5 Protocoles présent dans chaque jeu de données.

Jeu de données	**Couche Réseau (L3)**	**Couche Transport (L4)**	**Application**
DARPA [213]	IPv4	TCP	HTTP
			SMTP
		UDP	DNS
MQTTSet [220]	IPv6	TCP	MQTT
LoRaWAN [215]	LoRaWAN	-	-
CoAP	IPv6	UDP	CoAP

MQTTSet [220] contient uniquement des paquets IPv4. Une modification manuelle

est effectuée pour obtenir des paquets IPv6. Le jeu de données CoAP est collecté à l'aide du simulateur Contiki-NG [5]. Les données de chaque paquet sont simulées pas une série de valeurs constantes, de différentes natures, transmis par chaque émetteur.

Les ensembles de données DARPA [213] et LoRaWAN [215] contiennent des paquets avec des données car elles sont dérivées de l'activité des utilisateurs. Les jeux de données MQTTSet [220] et CoAP, qui sont obtenus via des simulateurs, ne contiennent pas de données réelles mais des valeurs fixes.

Pour chaque ensemble de données, 10 appareils sont sélectionnés. 5 appareils sont utilisés pour l'apprentissage du modèle et les 5 autres pour l'évaluation. Pendant l'évaluation, des appareils sont ajoutés au réseau pour voir l'effet de l'ajout de nouveaux appareils sur la compression. Les paquets des 5 appareils d'entraînement sont sous-échantillonnés et 15 000 paquets avec des valeurs différentes sont conservés. Les paquets sont divisés en trois groupes : données d'entraînement (80%), données de validation (10%) et données de test (10%). Les données d'entraînement et de validation constituent l'ensemble des données d'apprentissage. La modélisation est effectuée sur les données d'entraînements et de validation sur 7 époques afin d'éviter un temps d'apprentissage trop long. L'évaluation est effectuée sur les données de test. La section 4.4.4 aborde les temps d'apprentissage et de génération.

4.4.2 Traitement des données

La Figure 4.2 représente l'ensemble des étapes nécessaires à la compression et à la décompression des en-têtes de paquets réseau. Les étapes 1 et 2 montrent les étapes de traitement des données effectuées.

À l'étape 1, le contexte C est extrait, défini ici par l'en-tête des deux paquets précédents H_{k-2} et H_{k-1} appartenant à la même source identifiée par leurs adresses. Cela nous permet d'obtenir un bon compromis entre la quantité d'informations et le temps de calcul pour l'apprentissage et l'inférence.

Au cours de l'étape 2, une fenêtre glissante de taille l est appliquée au paquet à compresser. Nous choisissons de commencer la fenêtre au rang $i = 0$. Ainsi, la première fenêtre contient les bits $\{H_{k,0}, \ldots, H_{k,l-1}\}$. Dans la technique AC traditionnelle, les premiers bits de cette fenêtre ne sont pas compressés, ce qui entraîne une compression de taille $l = 16$ bits. Dans notre cas, nous appliquons un remplissage de taille $l = 16$ bits au début de l'en-tête à compresser. Ce remplissage nous permet de supprimer l'« overhead » introduite par AC sur l'en-tête à compresser.

5. `Contiki-NG:https://github.com/contiki-ng/contiki-ng`

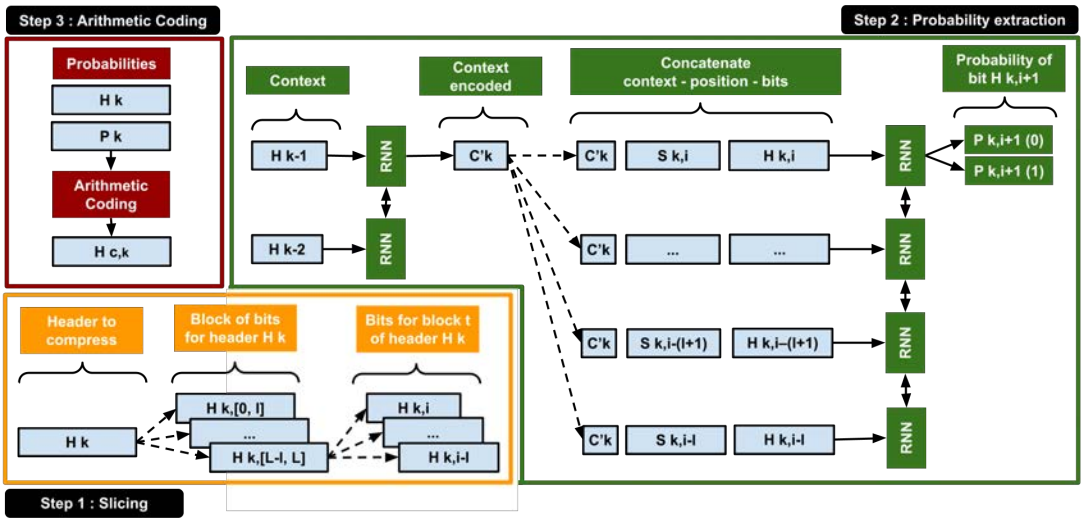

FIGURE 4.2 Architecture d'apprentissage profond pour la compression et la décompression d'en-têtes de paquets réseau.

Pour des raisons de temps de calcul, nous avons choisi de considérer les 100 premiers octets d'un paquet comme faisant partie de l'en-tête, soit $L = 100$ octets. Ces octets peuvent contenir des données utilisateur. DCH ne connaît pas la structure des protocoles contenus dans l'en-tête, ce choix est donc arbitraire. Ainsi, la compression est effectuée sans aucune notion préconçue sur la structure des données à compresser.

Une étude de sensibilité est réalisée dans la section 4.6 pour montrer l'effet des paramètres l_C et l sur le taux de compression. Selon cette étude, nous choisissons $l = 16$ bits et $l_C = 2$ en-têtes.

4.4.3 Architecture

L'architecture d'apprentissage profond est présentée à l'étape 3 de la Figure 4.2. La notation respecte celle présentée dans la partie 4.3.1. Son but est d'indiquer la probabilité d'obtenir la valeur du bit suivant, notée $P_{k,i+1}$. Les détails de l'architecture RNN (avec des cellules GRU [88]) utilisée sont présentés dans le Tableau 4.6.

L'objectif est d'effectuer une compression en utilisant des RNN et la technique de AC. Cette technique nous permet d'avoir un faible surcoût. L'état de l'art de cette approche par apprentissage profond, DeepZip [244], est générique mais n'est pas adapté à la compression des en-têtes de réseau. En effet, deux informations importantes sont manquantes :

TABLEAU 4.6 Architecture de l'étape 3 de la Figure 4.2. Les valeurs de paramètres obtenues sont basé sur $l = 16$, $l_C = 2$.

Contexte	**Paramètres**	**Dim. de sortie**
Input	0	$l_C \times (L-1)$
Bi-GRU	714240	256
Dense (LeakyReLU)	7710	C'
Paquet	**Paramètres**	**Dim. de sortie**
Input	0	$length(C') + length(sin)$ $+length(H_{k,i} \dots H_{k,i-l})$
Bi-GRU	14016	64
Dense (LeakyReLU)	195	3
Dense (Softmax)	8	2

— **La position du bit** : un bit dans $H_{k,i}$ a une position bien définie, notée i, dans la partie 4.3.1.

— **Le contexte** : défini dans la partie 4.3.1. Il s'agit d'un concept similaire aux approches telles que RoHC [229, 231].

Pour surmonter ces limitations, l'architecture se compose de deux parties. Une première partie prend en entrée le contexte C avec $l_C = 2$ en-têtes et $L = 800$ bits, la taille maximale d'un en-tête rencontré. Cette taille peut être ajustée en fonction des protocoles utilisés (si la taille est connue) ou fixée arbitrairement car DCH ne suppose aucune connaissance sur la structure des protocoles à compresser et donc sur la taille des en-têtes. Ainsi, le premier RNN récupère l'information du contexte C et l'encode sous la forme d'un vecteur, appelé C', dans une espace $\mathbb{R}^{30}$. L'annexe A montre la projection du contexte C' dans un espace $\mathbb{R}^2$ pour différents protocoles. Un second RNN est chargé d'apprendre la probabilité d'obtenir le bit suivant $P_{k,i+1}$ pour l'ensemble de l'en-tête. Il prend en entrée le vecteur C' ainsi qu'une information de position représentée par une sinusoïde à différentes fréquences, notée $s_{k,i}$. Cette information de position est inspirée de l'architecture Transformer [247] qui a fait ses preuves dans le traitement du langage naturel. L'apprentissage des deux parties se fait en une seule fois grâce à une infrastructure « cloud » [81].

4.4.4 Performances

Cette section résume les performances et l'infrastructure utilisée pour la génération et l'apprentissage des différents éléments de DCH. Les mesures de performances sont utilisées en prenant comme configuration $l_C = 2$ en-têtes et $l = 16$ bits. Après une analyse dimensionnelle détaillée dans la section4.6 nous avons montré que ce type de configuration est la plus performante. Le Tableau 4.7 résume les temps d'apprentissage et de compression

de DCH appliqué sur nos différents jeux de données en fonction d'un nombre de paquets donné. L'apprentissage, la compression, la décompression et les mesures sont effectuées sur l'architecture Osirim [81], la plateforme de calcul de l'IRIT, en utilisant des CPUs Intel Xeon Gold 6136 à 3GHz.

TABLEAU 4.7 Mesures des performances de compression pour chaque protocole présent dans chaque jeu de données.

Jeux de données	**Couche Réseau (L3)**	**Couche Transport (L4)**	**Application**	**Nombre de paquets**		**Temps de calcul**	
				Apprenti.	**Compres.**	**Apprenti.**	**Compres.**
DARPA [213]	IPv4	TCP	HTTP	13500	1500	19 h 36 min	19 min
			SMTP	13500	1500	47 h 50 min	21 min
		UDP	DNS	13500	1500	35 h	27 min
MQTTSet [220]	IPv6	UDP	MQTT	13500	1500	47 h 22 min	27 min
LoRaWAN [215]	LoRaWAN	-	-	13500	1500	2 h 41 min	7 min
CoAP	IPv6	UDP	CoAP	13500	1500	55 h 11 min	27 min

L'apprentissage ne nécessite pas l'utilisation de GPU car les données d'apprentissage ne peuvent pas être chargées en RAM. Un itérateur est utilisé afin de préparer les données. L'itérateur lit les données depuis le disque dur et découpe les en-têtes en y appliquant une fenêtre glissante de taille l. Cette opération prend un temps important qui rend l'utilisation de GPU inutile pour l'apprentissage.

Les différences dans les temps d'apprentissage et de génération s'expliquent par la distribution des tailles de paquets. Par exemple, LoRaWAN contient une plus grande quantité de paquets de petites tailles que HTTP. L'apprentissage, l'opération de compression et décompression sont donc plus rapides pour LoRaWAN.

La compression est parallélisée afin d'être plus rapide. Cela inclut la prédiction du modèle et la mise en oeuvre de l'AC. Seuls les temps de compression sont répertoriés, car l'opération de décompression suit la même dynamique et donc le délai est identique. Obtenir le temps de compression et de décompression d'un nombre de paquets données revient à multiplier par deux la durée de compression ou de décompression.

4.4.5 Conclusion

Nous avons expliqué le fonctionnement de notre architecture et son apprentissage. Cette architecture nous permet d'apprendre tous les types de protocoles et tous les types d'architectures (maillé, étoile, ...) sans avoir aucune connaissance préalable des en-têtes à compresser, contrairement aux approches de la littérature. Cependant, cette architecture ne peut pas être intégrée dans un appareil à faible consommation d'énergie. De plus, cette approche manque d'explicabilité.

4.5 Implémentation

4.5.1 Méthodes de compression

Pour notre mise en œuvre, nous avons choisi un capteur TTGO LoRa32 SX1276 OLED. Le capteur est programmé à l'aide de Arduino. La Figure 4.3 montre le capteur connecté à l'ordinateur en train d'effectuer une transmission.

L'utilisation d'une approche d'apprentissage profond pour l'inférence nécessite une puissance de calcul qui n'est pas disponible sur l'équipement IoT. Nous devons compresser le modèle. Il existe deux méthodes :

— **Distillation du modèle** : l'objectif est de comprimer le modèle original en réduisant le nombre de paramètres ou en utilisant un modèle avec moins de paramètres qui apprend à partir de la sortie du modèle original.

FIGURE 4.3 Capteur connecté à l'ordinateur qui effectue une transmission.

— **Interprétation du modèle** : l'objectif est de comprendre le fonctionnement des caractéristiques qui influencent les résultats du modèle et leur importance.

La distillation du modèle ne permet pas d'obtenir un modèle efficace qui puisse être intégré dans des appareils tels que les nôtres. Nous avons donc décidé d'utiliser les « Occlusion Maps » [8], une méthode d'interprétation du modèle.

4.5.2 Interprétation de modèle

« Occlusion Map » [8] est une méthode d'interprétation de modèle qui consiste à cacher chaque valeur à l'entrée du modèle, une à la fois, et à voir comment la sortie évolue. Cette méthode est présenté dans le chapitre 1 dans la section 1.6. L'objectif est de déterminer quelles sont les valeurs à l'entrée qui ont le plus d'influence sur la sortie du modèle. Nous avons choisi ce type de méthode car elle est plus intuitive à interpréter que les modèles de descente de gradient tels que « Saliency Map » [248]. Elle est également adaptée aux données tabulaires. Les « Occlusion Maps » [8] sont appliquées sur des données d'entraînement.

Nous notons m^o_n le masque chargé de masquer chaque valeur en position n. Dans notre cas, les valeurs d'entrées sont des bits notés $b_n = \{0, 1\}$, masquer une valeur revient à inverser sa valeur comme une opération XOR notée $\oplus$. Ensuite, nous examinons la différence entre la probabilité originale $P_{k,i}$, donnée par DCH, et la probabilité après application du masque $P^o_{k,n}$. La différence est notée $P^{diff}_{k,n}$. L'application des « Occlusion Map » [8] est indiquée dans la formule (4.7).

$$\begin{aligned}
&\forall i \in [0, \ldots, L-1], j \in [0, \ldots, l], n \in [0, \ldots, l_m] \\
&RNN((H_{k,i-j} \ldots H_k, C_k)) = P_{k,i} \\
&RNN((H_{k,i-j} \ldots H_k, C_k) \oplus m^o_n) = P^o_{k,n} \\
&P^{diff}_{k,n} = \|P_{k,i} - P^o_{k,n}\| \text{ et } l_m = l + (L-1) * l_C
\end{aligned} \tag{4.7}$$

Les bits l_{delta} ayant le plus grand $P^{diff}_{k,n}$ pour une position i fixe sont extraits pour former un ensemble de bits δ^i_k selon la formule (4.8).

$$\forall n \in [0, \ldots, l_m], max(P^{diff}_{k,n}) = \delta^i_k \tag{4.8}$$

La position n des bits extraits, pour une position fixe i, est utilisée pour définir le masque $m_{k,i}$ défini dans l'équation (4.5) utilisée pour extraire δ^i_k. Au maximum, seuls les bits qui expliquent 95% de la variabilité de $P^{diff}_{k,n}$ sont extraits. Ainsi, les bits contenus dans δ^i_k sont uniquement ceux qui ont un impact significatif sur $P_{k,i}$.

L'ensemble des opérations effectuées permet de construire la table T. Les probabilités $P_{k,i}$ sont réutilisées pour la sortie de la table T présentée dans l'équation (4.6).

4.5.3 Table de compression

Le nombre d'entrées dans la table T est conséquent et ne permet pas l'implémentation sur un capteur avec une faible quantité de mémoire. Cette partie se concentrera sur l'explication des opérations qui ont permis d'intégrer la table.

La table T est codée en dur dans le dispositif au sein d'une variable. Nous voulions donc (i) minimiser l'espace mémoire occupé par la variable stockant la table, (ii) minimiser l'espace mémoire occupé par le fichier contenant le code, (iii) réduire le temps d'exécution de la table.

Tout d'abord, nous avons converti les probabilités de sortie de notre table en entiers non signés de 8 bits afin de n'utiliser que 8 bits au lieu de 64 bits. Cela résout le problème (i). Nous avons également pu supprimer le « . » des nombres flottants, ce qui réduit la taille du fichier de code et résout donc le problème (ii).

Le tableau T est divisé en deux sous tableaux :

— Table T^{proba} : contient l'ensemble des probabilités $P_{k,i}$ de manière ordonnée. Elle contient $(l_C + 1) \times (L - 1)$ valeurs.

— Table T^{count} : contient le nombre de valeurs utilisées dans T^{proba} pour chaque position i. Par exemple, certains bits $H_{k,i}$ ont une valeur unique, donc $T^{count}_i = 1$. Il contient $(l_C + 1) \times (L - 1)$ valeurs.

La table T^{proba} est compressée. S'il existe une longue séquence de probabilités par

défaut, les valeurs sont regroupées en une seule valeur supérieure à 100. Par exemple, s'il y a 60 « $P_{k,i} = 0.5$ », la valeur saisie est 160. L'espace économisé par cette compression est examiné dans la section 4.6.3. La table T^{count} enregistre cette information en réduisant le nombre de valeurs pour une position donnée.

Ainsi, si nous voulons obtenir la probabilité d'un $P_{k,i}$ avec une taille de contexte donnée l_C, nous additionnons les valeurs de la table T^{count} jusqu'au rang $l_C \times i$, noté Q. Q nous donne le rang de la première valeur des probabilités en position i pour un contexte de taille l_C. Nous notons q, la valeur obtenue dans la table T^{count}, pour un l_C et un i donnés, notée $T^{count}_{l_C,i}$. q est le nombre de valeurs stockées dans T^{proba} associées à $P_{k,i}$. La formule (4.9) explique les valeurs q et Q.

$$\begin{array}{l} Q = \sum_{n=0}^{l_C} \sum_{i=0}^{L-1} T^{count}_{n,i} \\ \forall n \in [0, \ldots, l_C], q = T^{count}_{n,i} \end{array} \tag{4.9}$$

4.5.4 Conclusion

Nous avons vu le matériel utilisé pour embarquer notre modèle de compression d'en-tête. Tout d'abord, nous avons expliqué la transformation de modèle effectuée sur notre modèle d'apprentissage profond pour obtenir une table T grâce à la méthode des « Occlusion Map » [8]. Nous avons montré comment compresser cette table afin de l'intégrer dans un équipement type IoT. Dans la section suivante, nous verrons les limites d'une telle compression ainsi que ses avantages.

4.6 Résultats

4.6.1 Présentation

Deux mesures sont utilisées pour évaluer les performances de la compression : le taux de compression, noté CR et le gain d'espace, noté SS. Elles sont définies dans l'équation (4.10).

$$\begin{array}{l} \text{Le taux de compression CR} = H_{décompressé}/H_{compressé} \in \mathbb{R}^+ \\ \text{Le gain d'espace SS} = 1 - (H_{compressé}/H_{dcompressé}) \in [-\infty, 1] \end{array} \tag{4.10}$$

CR est considéré comme optimal s'il est proche de $+\infty$. Plus il est proche de 0, plus il est considéré comme mauvais. SS est considéré comme optimal lorsqu'il est égal à 1. Plus il est proche de $-\infty$, plus il est considéré comme mauvais. Les paquets examinés dans les données de test proviennent des mêmes appareils que ceux présents dans les données

d'apprentissage.

Les paquets ne sont en aucun cas transformés. Ainsi, les champs tels que CRC, Checksum, ... qui ont des valeurs très variables, ne sont pas supprimés. La suppression de ces champs peut apporter un gain significatif de compression.

Toutes les mesures sont comparées au modèle de référence. Ce modèle est construit à partir d'une table de fréquence de 8 bits construite grâce à une fenêtre glissante sur les en-têtes de paquets pour chaque protocole. La technique AC est ensuite appliquée pour compresser avec cette table. La performance de compression de DCH des paquets IPv6 est comparée à 6LoWPAN [232] dans la configuration la plus favorable. L'en-tête IPv6 est ainsi réduit à deux octets. Les performances ne sont pas comparées à SCSH [234] qui représente une limite de performance haute. En effet, SCSH [234] nécessite une bonne connaissance de la structure du paquet à compresser ce qui permet d'obtenir des performances optimales. En revanche, DCH ne nécessite aucune connaissance de la structure du protocole. De plus, la configuration de SCSH [234] dépend du choix du cadre d'application, ce qui fait varier fortement les performances de compression.

4.6.2 Étude de sensibilité

Taille de contexte

L'objectif est d'évaluer le gain d'espace obtenu en fonction de la taille du contexte, notée l_C.

La Figure 4.4 montre les distributions des gains d'espace en fonction de l_C, pour chaque protocole étudié sur les données de test et sur les équipements présents dans les données d'apprentissage. Lors de l'analyse, $l = 16$ bits. DCH est supérieur au modèle de référence quelle que soit la taille de contexte utilisée. Les performances semblent être légèrement affectées par la présence du contexte. Dans le cas de MQTT et de CoAP, les performances de compression sont supérieures à celles de 6LoWPAN [232]. Cette différence est plus importante pour CoAP que pour MQTT.

La Figure 4.5 les gains d'espace cumulés pour chaque position de bit en fonction de l_C, pour chaque protocole étudié sur les données de test et sur les équipements présents dans les données d'entraînement. La meilleure performance est obtenue au niveau des bits dans les positions les plus basses, c'est-à-dire au niveau de l'en-tête IP et TCP/UDP car il y a beaucoup de champs fixes. Les performances avec contexte sont meilleures pour les champs dont les valeurs peuvent être dérivées du contexte, tels que : le numéro de séquence, les champs d'adresse (en cas d'inversion), Time to Live (TTL), etc. D'autres champs ont un impact négatif sur les performances de compression. La charge utile peut avoir

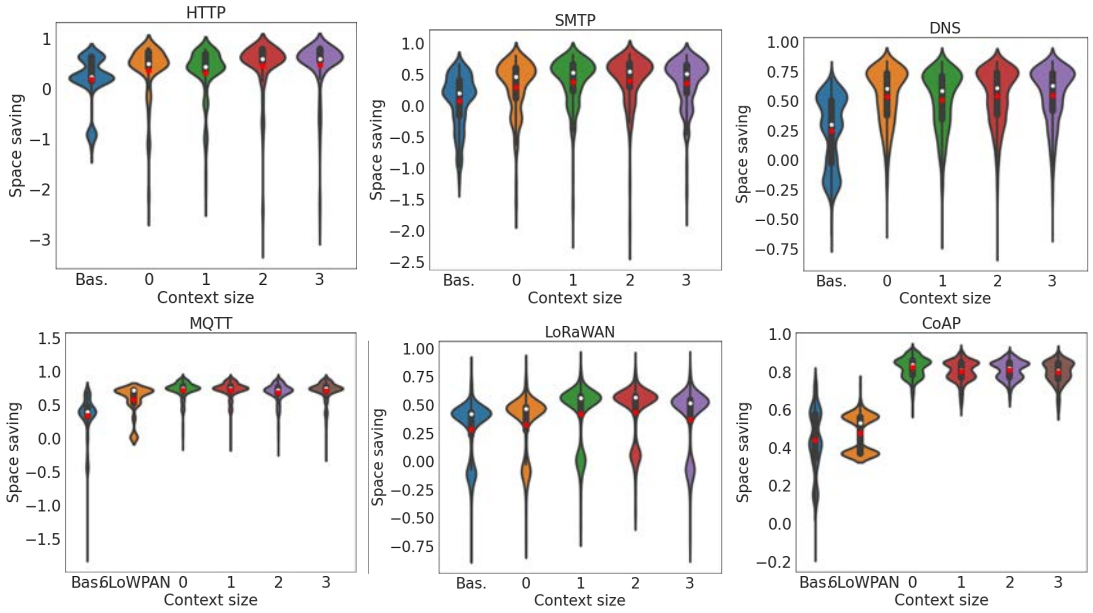

FIGURE 4.4 Distributions des gains d'espace en fonction de l_C, pour chaque protocole étudié sur les données de test et sur les équipements présents dans les données d'entraînement. Les points blancs et les points rouges représentent respectivement la médiane et la moyenne pour chaque distribution.

une très grande variabilité, surtout si elle est partiellement chiffrée, comme c'est le cas avec LoRaWAN. Ces champs à fortes incertitudes, difficile à compresser, entraînant dans certains rares cas une taille de l'en-tête compressé supérieure à l'en-tête non compressé. C'est ce qui explique l'apparition de valeurs négatives visibles sur la Figure 4.4 et la Figure 4.5. Cela explique aussi la différence entre MQTT et CoAP avec 6LoWPAN [232]. La charge utile de CoAP est similaire pour certains paquets, contrairement à MQTT où la variabilité est plus importante. La compression de l'en-tête permet à elle seule d'obtenir de bien meilleures performances que 6LoWPAN [232] dans le cas de MQTT. D'autres champs tels que la somme de contrôle et le CRC ont un effet similaire car ils nécessitent une vue de l'ensemble du paquet. Dans notre cas, nous n'avons qu'une vue locale par le biais d'une fenêtre glissante. L'augmentation de la taille du contexte n'améliore pas les performances pour tous les protocoles et tous les champs. Cela est dû au manque d'informations pertinentes supplémentaires. Dans le cas de LoRaWAN, cela augmente la dimension du vecteur d'entrée associée au contexte dans l'entrée de DCH. Si ces informations sont inutiles, elles sont associées à du bruit et augmentent la difficulté d'apprentissage, ce qui dégrade les performances. Nous pensons que $l_C = 2$ en-têtes est le meilleur compromis.

Taille de la fenêtre

L'objectif est d'évaluer le gain d'espace obtenu en fonction de la taille de la fenêtre glissante appliquée sur l'en-tête à compresser, notée l.

La Figure 4.6 montre les distributions des gains d'espace en fonction de l, appliquée sur l'en-tête pour chaque protocole étudié sur les données de test et sur les équipements présents dans les données d'entraînement. Pour l'analyse, $l_C = 2$ en-têtes. DCH est supérieur au modèle de référence quelle que soit la taille de la fenêtre utilisée. L'économie d'espace est affectée par la variation de la taille de la fenêtre. Contrairement à une implantation traditionnelle de l'AC, nous n'avons pas d'« overhead » lors de compression car nous utilisons du « padding » [6]. Ainsi, l'augmentation de la taille de la fenêtre n'augmente pas la surcharge et ne dégrade pas les performances.

La Figure 4.7 montre les gains d'espace cumulés pour chaque position de bit en fonction de l, sur les données de test et sur les équipements présents dans les données d'apprentissage. Nous constatons que l'effet de la taille de la fenêtre dépend de la taille des champs. En général, plus la taille des champs est grande, plus l'impact et le gain de performance de la taille de la fenêtre sont importants. Pour DNS et SMTP, nous pouvons constater que l'écart de performance entre les différentes tailles de fenêtre augmente lors de la compression de l'en-tête. Ces protocoles ont des champs de taille variable et importante. Inversement, l'utilisation d'une petite taille de fenêtre sur des champs de grande taille (c'est-à-dire l'adresse IPv6) a un impact significatif sur la taille de la fenêtre. Nous pensons que $l = 16$ bits est le meilleur compromis.

Nouvelles équipements

L'objectif est d'évaluer l'impact sur le gain d'espace lors de l'ajout d'équipements absents dans les données d'entraînement. Cinq équipements sont utilisés pour l'évaluation, absents des données d'apprentissage.

La Figure 4.8 montre les distributions des gains d'espace pour chaque protocole étudié sur des données d'équipements absents des données d'apprentissage. Pour l'analyse, $l_C = 2$ en-têtes et $l = 16$ bits.

La Figure 4.9 montre les gains d'espace cumulés pour chaque position de bit sur des données provenant d'équipements absents des données d'apprentissage. Les résultats sont similaires pour les autres protocoles. DCH est supérieur au modèle de référence sur les données provenant de l'équipement d'entraînement. En revanche, sur les données provenant

6. L'opération est similaire à celle pratiquée lors du chapitre 2 de la thèse sauf qu'elle est ici appliquée au début du paquet et non à la fin.

de l'équipement de test, DCH obtient des performances inférieures au modèle de référence. Le modèle gagne en performance grâce à sa complexité mais perd en généralisation. DCH perd en performance pour les champs dont la valeur est fortement influencée par le changement d'équipement. Pour les protocoles utilisant IPv6, les champs d'adresse ont un impact négatif sur les performances. Ce comportement est similaire pour les protocoles qui n'utilisent pas IP, tels que LoRaWAN.

4.6.3 Implantation

Nombre de bits

L'objectif est d'évaluer le gain d'espace obtenu en fonction du nombre de bits, noté l_{delta}, utilisée pour construire la table, notée T, à intégrer dans l'équipement IoT.

La Figure 4.10 montre les distributions des gains d'espace pour chaque taille de l_{delta} utilisée pour chaque protocole étudié sur les données de test. l_{delta}, évoqué dans la section 4.5.2, représente le nombre de bits extraits pour construire la table T. Pour l'analyse, $l_C = 2$ en-têtes et $l = 16$ bits. DCH est supérieur au modèle de référence, mais cela dépend de la taille l_{delta} utilisée. Les performances augmentent avec le nombre de bits l_{delta} utilisé. Passé un certain seuil, les performances se stabilisent et l'augmentation du nombre de bits l_{delta} n'augmente pas les performances. En général, le nombre minimum moyen de bits pertinents pour la compression est de 6 ($l_{delta} = 6$ bits). Le seuil de 95% utilisé après les « Occlusion Maps » [8] permet de se concentrer uniquement sur les valeurs importantes. De plus, comme précédemment, les performances de compression sont meilleures sur les en-têtes que sur les données, ce qui explique les résultats très proches de CoAP et MQTT avec 6LoWPAN [232].

Précision

L'objectif est d'évaluer le gain d'espace obtenu en fonction du nombre de chiffres utilisés pour encoder la probabilité. La probabilité est stockée sous forme d'un entier et une simple division permet de trouver sa valeur en flottant. Par exemple, le nombre 9 représente la probabilité 0.9 encodée sur 1 chiffre. Le nombre 22 représente la probabilité 0.22 encodée sur 2 chiffres.

La Figure 4.11 montre les distributions des gains d'espace pour chaque nombre de chiffres utilisés pour encoder la probabilité du bit $H_{k,i}$, sur les données de test et sur les équipements présents dans les données d'apprentissage. Pour l'analyse, $l_C = 2$ en-têtes et $l = 16$ bits. Le nombre de chiffre utilisé pour encoder les probabilités n'a pas d'effet sur les performances de la compression. Si la probabilité est codée sur un seul chiffre, le gain de

compression diminue en cas d'erreur. Au contraire, il augmente de manière significative. Le nombre de chiffres n'a donc pas d'effet sur la performance de compression de chaque paquet, mais sur la variabilité cumulative du gain d'espace pour chaque position de bit. C'est une source importante de gain pour le stockage.

Consommation mémoire

L'objectif est d'évaluer l'empreinte mémoire sur l'équipement de la table, notée T, nécessaire à la compression, en fonction de la taille de l'en-tête, notée L.

La Figure 4.12 montre les taux de compression obtenus après compression des tables pour chaque valeur de l_{delta} utilisée avec $l_C = 2$ en-têtes et $l = 16$ bits. En pratique, les tables T ont une taille maximale, notée l_T^{MAX} qui est limitée au nombre de valeurs possibles, donné par l'équation (4.11).

$$l_T^{MAX} = 2^{l_{delta}} \times (l_C + 1) \tag{4.11}$$

Cependant, certains champs ont des valeurs fixes qui réduisent le nombre de possibilités. Nous avons également proposé une méthode de compression pour l'implantation de nos tables sur le capteur. Nous observons une réduction non négligeable de 70% de la taille des tables, en moyenne pour tous les protocoles étudiés, pour $l_{delta} = 6$ bits. Cette réduction permet d'embarquer la table T sur un équipement IoT avec une faible mémoire.

4.7 Conclusion

Dans ce chapitre, nous proposons DCH une approche pour la compression d'en-têtes universelle, sans perte, par apprentissage profond. La compression s'effectue séquentiellement par AC. Nous avons montré la capacité de DCH à pouvoir travailler sur différents types de protocoles, sans a priori sur leurs structures.

Afin de pouvoir réaliser la compression directement sur l'équipement IoT, nous effectuons la transformation de modèle à l'aide des « Occlusion Maps » [8]. Cette transformation nous permet d'obtenir des tables, notées T, déployables sur des équipements IoT pour réaliser de la compression d'en-têtes. Pour rendre ces tables embarquables sur la majorité des équipements IoT, nous avons proposé une méthodes de compression de celles-ci.

Les résultats montrent la versatilité de notre approche ainsi que sa capacité à surpasser les approches traditionnelles telles que 6LoWPAN. Ainsi, DCH peut être utilisé pour différents cadres applicatifs, quel que soit le type de trafic et de protocoles.

Néanmoins, DCH présente certaines limites. Dans le cas de communication multiple, il est nécessaire d'utiliser les bonnes informations de contexte pour chaque communication. Cela peut passer par l'utilisation d'un identifiant attribué aléatoirement pour chaque flux ce qui peut réduire les performances de compression. De plus, malgré nos expérimentations, il n'est pas possible de déterminer finement les limites de DCH face aux différentes structures de champs qui peuvent exister au sein des protocoles.

Dans le prochain chapitre, nous analyserons DCH, en détails, afin de comprendre l'impact des différents types de champs, présents dans un en-tête, sur les performances de compression. Nous en déduirons également les limites de notre approche.

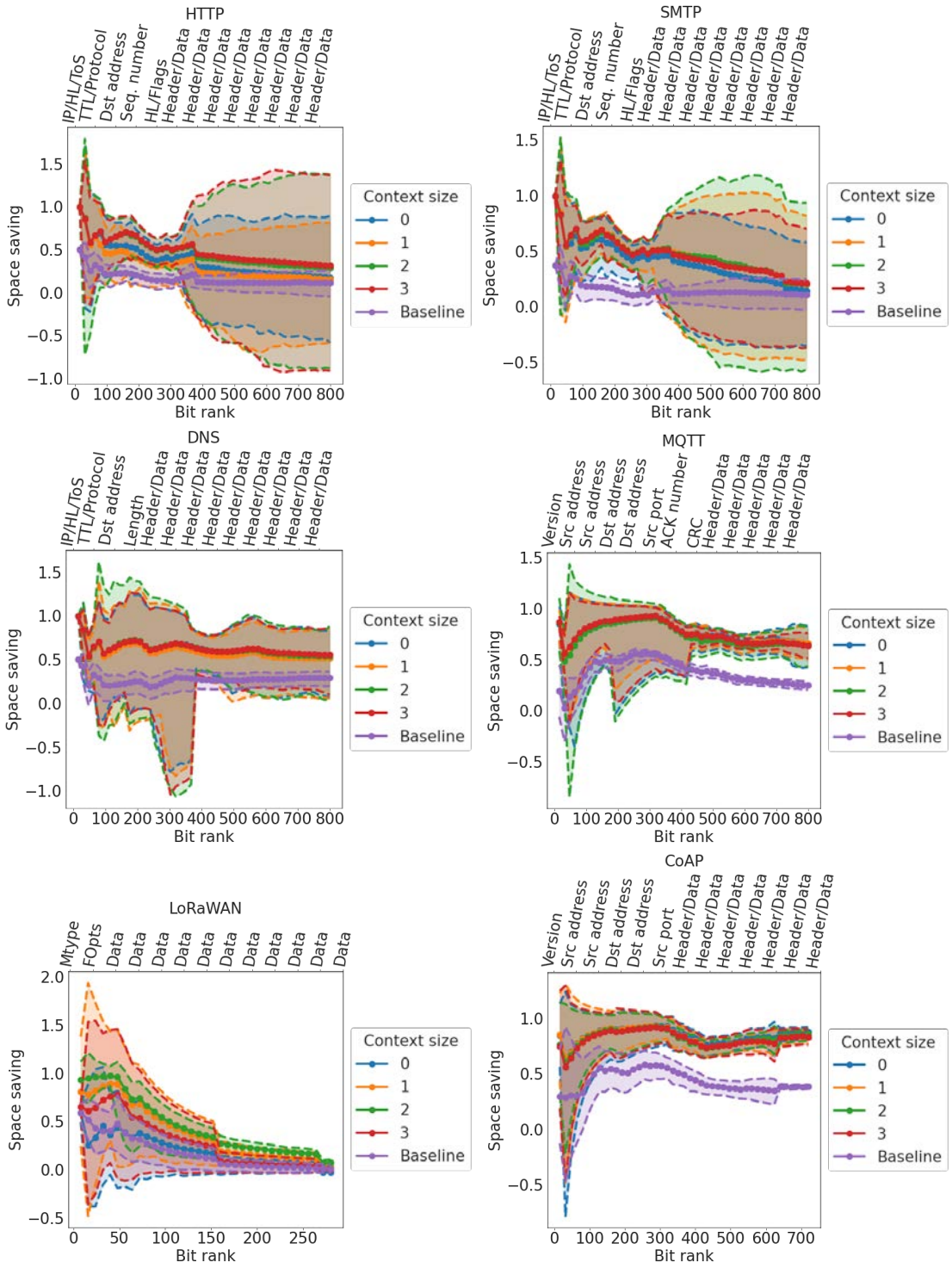

FIGURE 4.5 Gains d'espace cumulés pour chaque position de bit en fonction de l_C, pour chaque protocole étudié sur les données de test et sur les équipements présents dans les données d'entraînement.

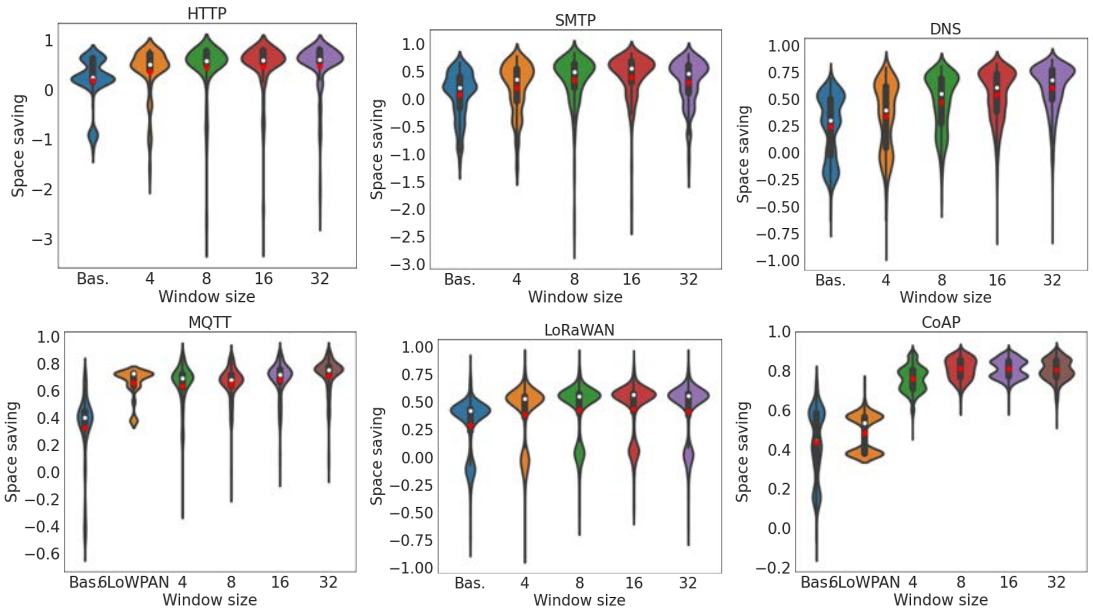

FIGURE 4.6 Distributions des gains d'espace en fonction de l, appliquées sur l'en-tête pour chaque protocole étudié sur les données de test et sur les équipements présents dans les données d'entraînement. Les points blancs et les points rouges représentent respectivement la médiane et la moyenne de chaque distribution.

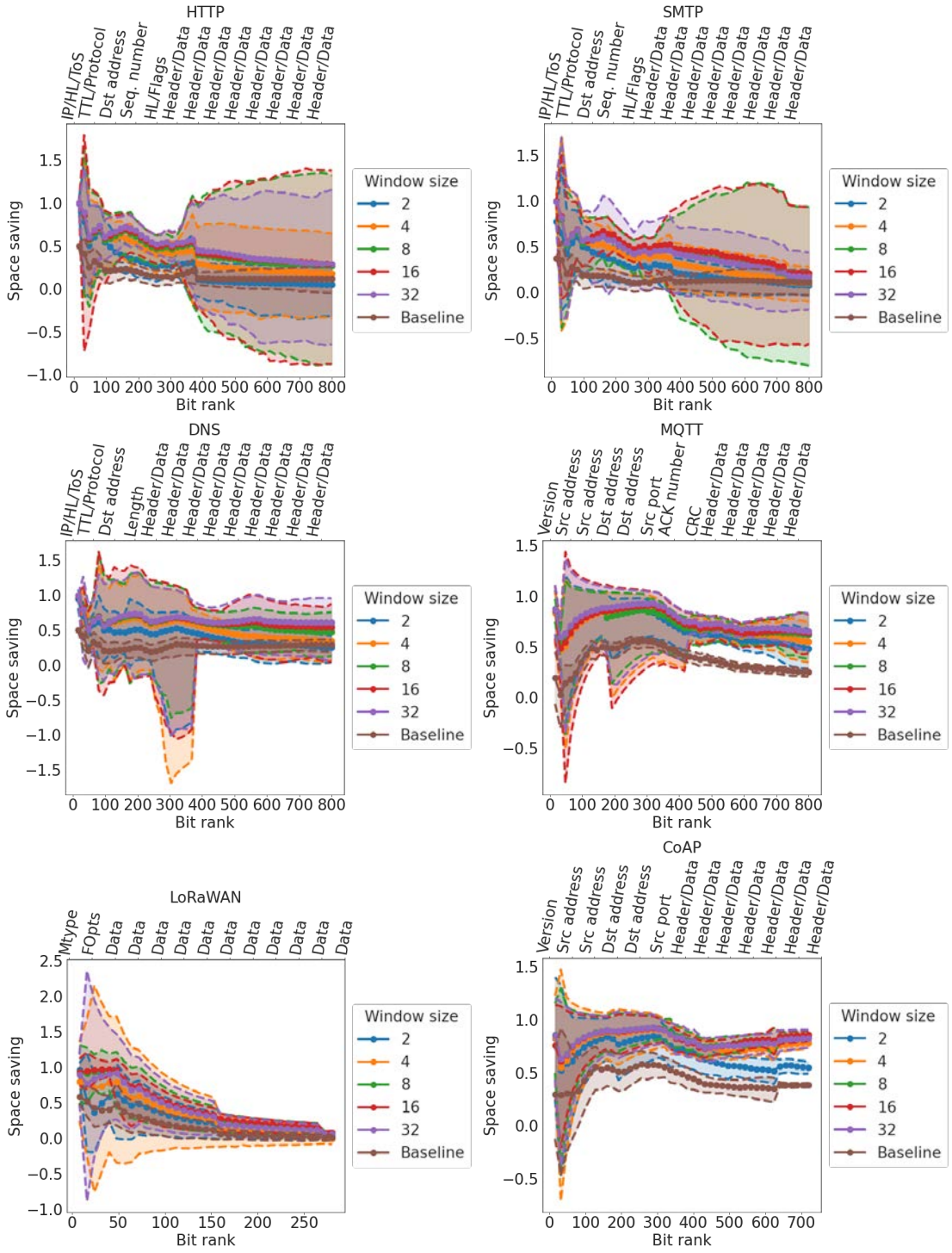

FIGURE 4.7 Gains d'espace cumulés pour chaque position de bit en fonction de l, pour chaque protocole étudié sur les données de test et sur les équipements présents dans les données d'apprentissage.

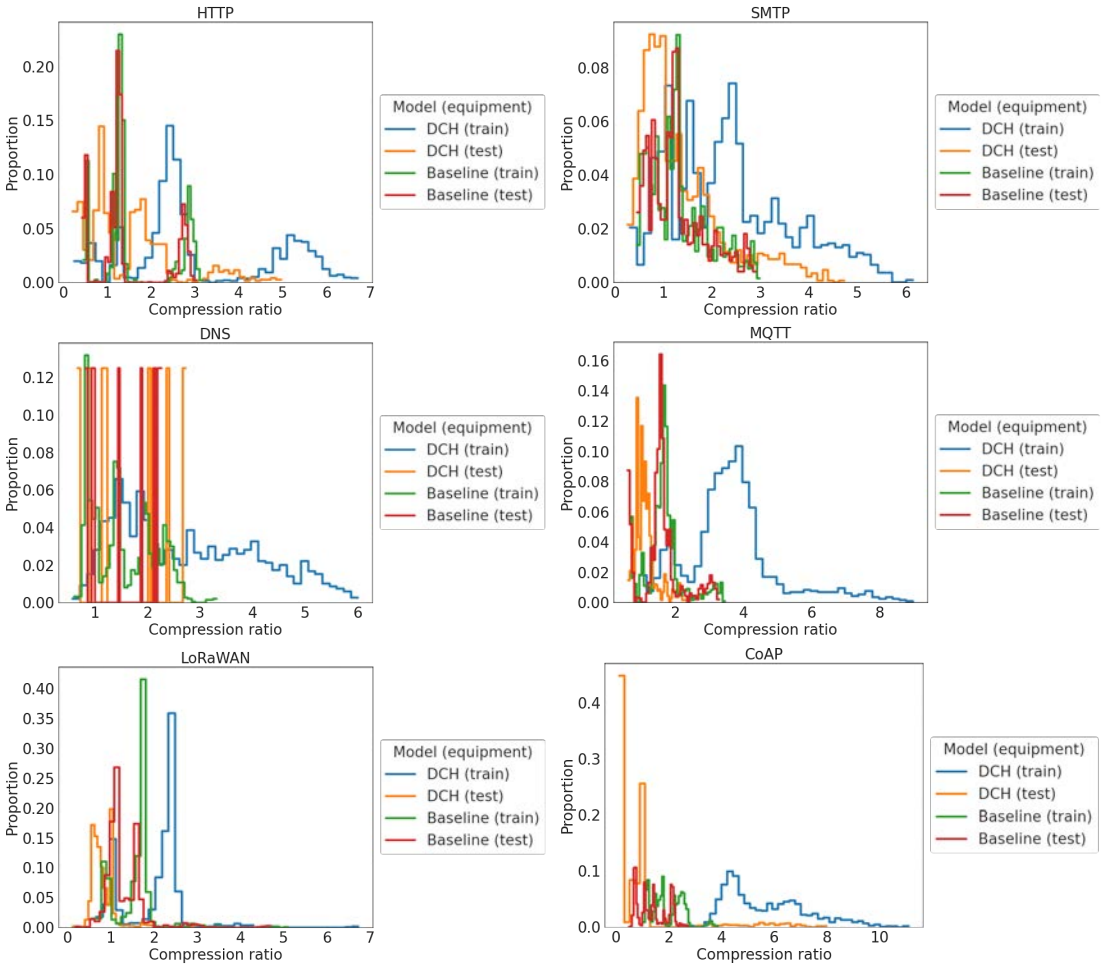

FIGURE 4.8 Distributions des gains d'espace pour chaque protocole étudié sur des données d'équipements absents des données d'apprentissage.

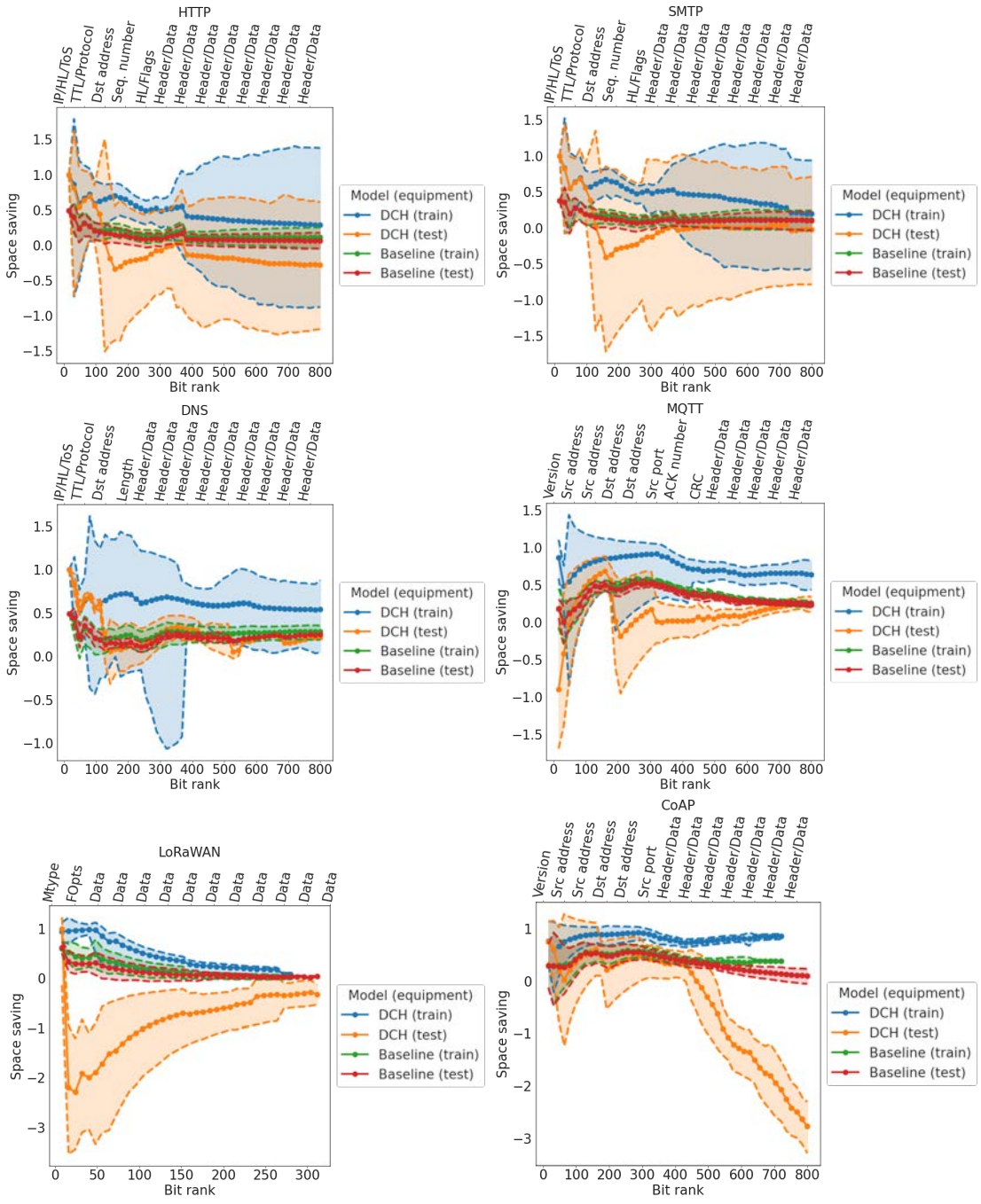

FIGURE 4.9 Gains d'espace cumulés pour chaque position de bit sur des données provenant d'équipements absents des données d'apprentissage.

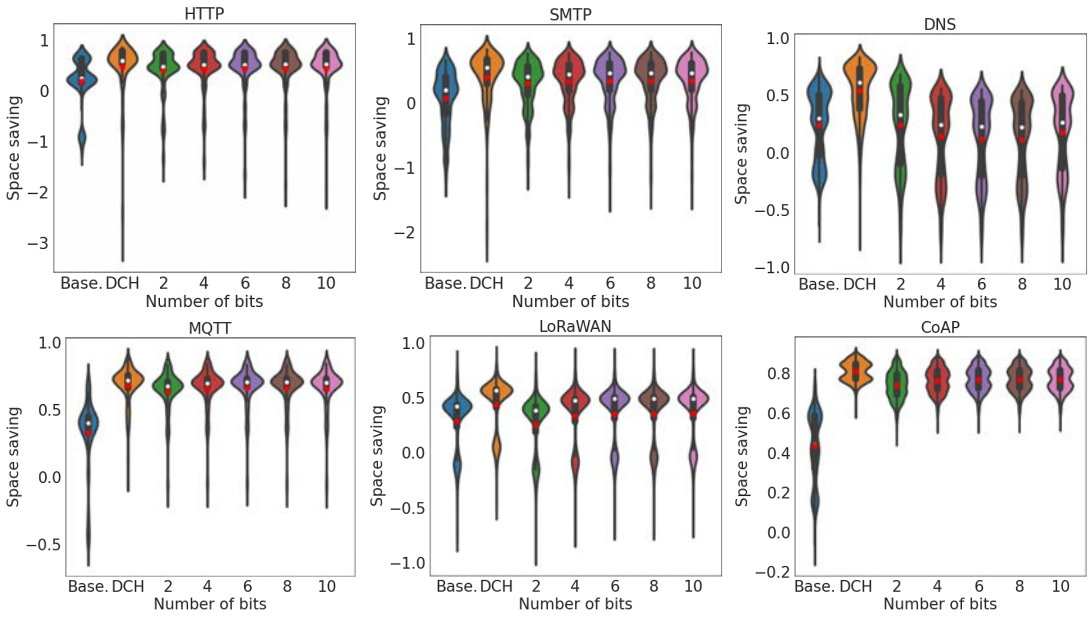

FIGURE 4.10 Distributions de gains d'espace pour chaque taille de l_{delta} utilisée pour chaque protocole étudié sur les données de test. Les points blancs et les points rouges représentent respectivement la médiane et la moyenne de chaque distribution.

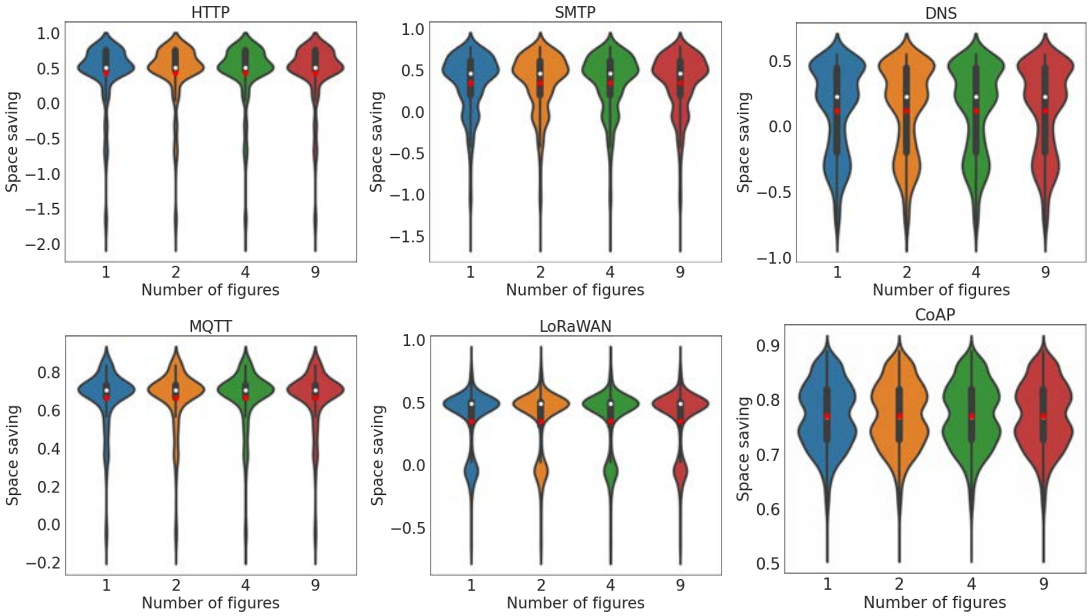

FIGURE 4.11 Distributions des gains d'espace pour chaque nombre de chiffres utilisés pour encoder la probabilité du bit $H_{k,i}$, sur les données de test et sur les équipements présents dans les données d'apprentissage. Les points blancs et les points rouges représentent respectivement la médiane et la moyenne de chaque distribution.

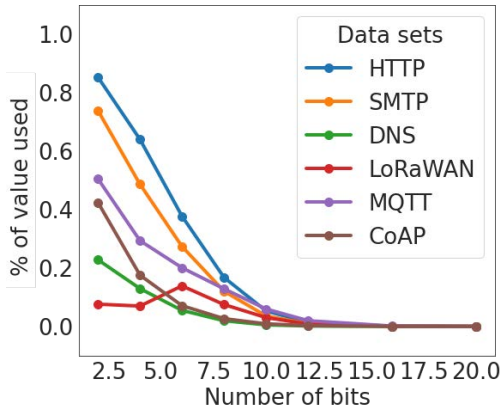

FIGURE 4.12 Taux de compression obtenus après compression des tables pour chaque valeur de l_{delta} utilisée.

Chapitre 5

Étude de sensibilité des performances de DCH

Sommaire

5.1 Introduction

Dans ce chapitre, nous effectuons une étude approfondie de l'approche que nous avons développée pour effectuer de la compression universelle d'en-têtes de trafics réseau, nommée DCH.

Notre objectif est d'expliquer les performances de DCH en fonction des différentes structures que nous retrouvons dans les champs des en-têtes de protocole à compresser. Ces structures sont associées à des champs au sein d'un en-tête. Afin de bien comprendre le comportement de DCH sur les différents types de champs que l'on peut rencontrer dans un en-tête, nous avons défini les six types suivants :

- **Checksum :** la valeur attribuée résulte d'un calcul fondé sur d'autres valeurs présentes dans l'en-tête. Dans ce chapitre, nous avons implanté l'algorithme de la RFC 1071 [249] utilisé par exemple pour les champs « checksum » de TCP et IP.
- **Fixe :** il s'agit ici d'un champ constant tel que le champ version d'IP, certains paramètres du protocoles LoRaWAN.
- **Fixe par flux :** une valeur aléatoire est associée à chaque flux. Elle reste identique pour tous les paquets d'un même flux. Par exemple, les champs port de TCP/UDP.
- **Inversion :** ce type permet de représenter des couples de champs qui sont régulièrement inversés tels que : les champs d'adresses IP, les numéros de port TCP/UDP.
- **Compteur :** un champ de type numéro de séquence tel que celui de TCP ou le champ « FCnt » de LoRaWAN. Pour une meilleure compréhension, les incréments seront fixés par pas de 1.
- **Aléatoire :** la partie donnée ou les valeurs de champs ne pouvant pas être déterminées, par aucun moyen.

Ces types couvrent l'essentiel des champs présents. Par exemple dans IPv4, IPv6, UDP, TCP, LoRaWAN, MQTT, CoAP, HTTP, SMTP, DNS. L'évaluation sur ces six types de champs nous permet une évaluation suffisamment exhaustive pour comprendre l'impact des différents protocoles sur la performance de DCH.

Les métriques utilisées durant notre étude sont : le taux de compression et le gain d'espace présenté dans l'équation (5.1). Nous utiliserons une de ces deux métriques en fonction de la pertinence de la visualisation.

$$\begin{aligned} &\text{Le taux de compression CR} = H_{décompressé}/H_{compressé} \in \mathbb{R}^+ \\ &\text{Le gain d'espace SS} = 1 - (H_{compressé}/H_{décompressé}) = 1 - (1/CR) \end{aligned} \tag{5.1}$$

CR est considéré comme optimal lorsqu'il tant vers $+\infty$. Plus il est proche de 0, plus

celui-ci est considéré comme mauvais. Pour les visualisations, CR $\in]0, L]$ avec L la taille de l'en-tête. G est considéré comme optimal lorsqu'il est égal à 1. Plus il tend vers $-\infty$ et plus il est considéré comme mauvais.

Dans un premier temps, nous présentons le jeu de données développé pour notre étude. Ensuite, nous étudions l'effet des différents paramètres de DCH conjointement avec le type de champs sur les performances de la compression. Enfin, nous répétons cette étude sur un jeu de données issu de la combinaison des différents types de champs étudiés. Pour finir, nous analysons la table de probabilités, notée T, résultant de l'interprétation de DCH, après son apprentissage.

5.2 Jeux de données simulés

Nous définissons un jeu de données simulé pour chaque type de champs étudié. Chaque jeu de données comprend un ensemble de vecteurs de 32 bits associé à un type de champs. L'objectif de ce jeu de données est multiple. Sa faible dimension nous permet de vérifier notre analyse sur des exemples simples et maîtrisés. Il nous permet également de comprendre l'impact des divers types de champs et leurs interactions. Enfin, il permet tester notre code. Les jeux de données sont séparés en trois parties en suivant un processus classique d'apprentissage automatique : données d'entraînement (80 %), données de validation (10 %) et données de test (10 %).Nous considérons que chaque vecteur, assimilable à un paquet avec un champ, fait partie d'un flux de trois paquets, donc ici $l_C = 2$ paquets.

Le détail de construction pour chaque type de jeu de données est le suivant :

— **Checksum :** les 24 premiers bits sont générés aléatoirement. Les 8 derniers bits sont le résultat du calcul checksum appliqué sur les 24 premiers bits.

— **Fixe :** une série aléatoire de bits est utilisée et répliquée pour construire l'ensemble des données. Ce type de jeu de données peut s'assimiler au champ « version » de IP.

— **Fixe par flux :** une série aléatoire de bits est utilisée pour construire les bits de chaque flux. La série de bits est différente pour chaque flux. Ce type de jeu de données peut s'assimiler au port source ou destination de TCP ou UDP.

— **Inversion :** pour chaque flux, le premier vecteur comprend 32 bits générés aléatoirement. Ensuite, les 16 premiers bits et les 16 derniers sont inversés successivement au cours du flux. Ce type de jeu de données peut s'assimiler aux champs d'adresses de IP.

— **Compteur :** pour chaque flux, le premier vecteur est initialisé à une valeur

aléatoire. Les vecteurs successifs au cours du flux ont pour valeur leur prédécesseur incrémenté de 1. Ce type de jeu de données peut s'assimiler au champ « FCnt » de LoRaWAN.

— **Aléatoire :** le jeu de données est généré aléatoirement.

A l'exception du jeu de données comprenant des champs « fixe », une graine de génération différente est utilisée pour générer les données de test. L'objectif est de réduire le risque que le modèle apprenne des récurrences apparaissant dans les données lors de la génération ce qui fausserait les résultats. De plus, avoir un contrôle sur la graine de génération nous permet de différencier les données d'entraînement et de test.

5.3 Évaluation champ par champ

5.3.1 Présentation

L'objectif est d'étudier l'impact du type de champs sur les performances de compression. La Figure 5.1 montre les performances obtenues pour chaque jeu de données sur les données de test.

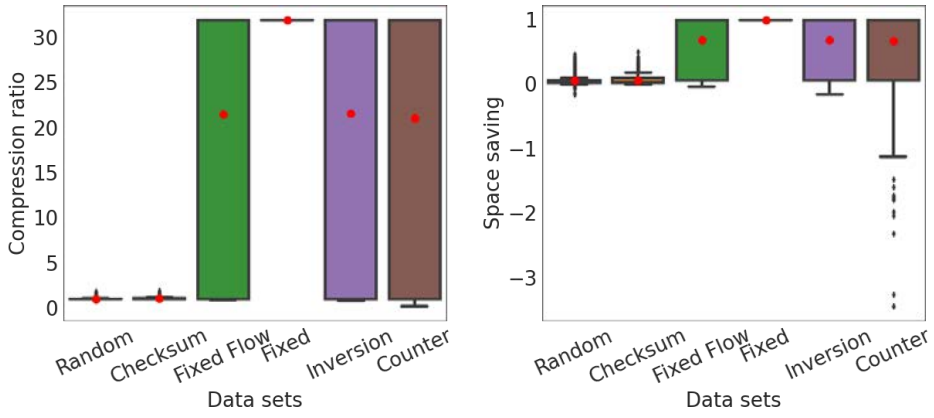

FIGURE 5.1 Boîtes à moustaches du taux de compression (à gauche) et du gain d'espace (à droite) pour chaque type de jeu de données étudié sur les données de test. Les points rouges représentent les moyennes associées à chaque distribution

Le jeu de données « fixe » obtient un gain d'espace de 1, celui-ci est parfaitement compressé. À l'inverse, le jeu de données « aléatoire » n'est pas compressé ce qui est cohérent. En effet, aucun motif n'est distinguable d'un jeu de données générées aléatoirement. Les jeux de données « fixe par flux », « inversion » et « compteur » possèdent un taux de compression avec une variabilité importante. Cette variabilité peut s'expli-

quer par l'absence de contexte lors de la compression du premier paquet de chaque flux. Le contexte donne une information importante pour la compression d'un en-tête. Sans contexte, ces jeux de données deviennent équivalents à une série de bits aléatoire. Cet écart de performance dans la compression entre les premiers paquets des flux et les autres paquets du flux vont fortement impacter la moyenne mais faiblement la médiane. Les flux défini dans notre jeu de données ne comprennent que 3 paquets (pour rappel $l_C = 2$ paquets). Nous avons donc 1/3 des données avec un gain d'espace de 1 et 2/3 avec un gain d'espace de 0. Cette situation explique l'écart important entre les deux mesures.

Le jeu de données « checksum » n'est pas compressé : il est assimilé à de l'aléatoire. Cela s'explique par le choix de la taille de la fenêtre, notée l, en fonction de la taille du champ sur lequel est encodé le « checksum ». Nous détaillerons cela dans la partie 5.3.3.

Afin d'obtenir une compréhension plus fine, nous avons analysé l'impact des paramètres de DCH sur les performances de compression pour chaque jeu de données. Les performances obtenues sont comparées à un modèle de référence (« baseline ») construit en utilisant une table de fréquences après application d'une fenêtre glissante de taille 8 sur les données d'apprentissage. Cette table de compression construite est utilisée pour la compression en utilisant la méthode de l'AC.

5.3.2 Taille du contexte

Dans cette section, notre objectif est d'évaluer l'impact de la taille du contexte, notée l_C, en fonction du type de champ, sur les performances de compression. La Figure 5.2 présente la distribution du gain d'espace en fonction de l_C, pour chaque type de jeu de données sur les données de test.

La Figure 5.3 montre les gains d'espace cumulés obtenus pour chaque position de bit en fonction de l_C pour chaque type de jeu de données sur les données de test.

Le contexte peut être défini en fonction de l'équipement lors de la communication. Cela peut correspondre aux paquets précédemment échangés avec celui-ci, c'est le choix qui a été fait pour le développement de DCH. Le contexte peut être affiné et également dépendre des éléments présents dans cette liste non exhaustive :

— Flux : le flux est défini par le tuple : addresse IP source, addresse IP destination, port source et port destination. Les paquets précédemment envoyés et appartenant au même flux que le paquet à compresser peuvent être utilisés.
— Application : le contexte est défini en fonction de l'application. Cela peut correspondre aux derniers paquets précédemment échangés en lien avec les données de l'application transportée.
— Direction : lors de communications impliquant des équipements IoT, il peut être

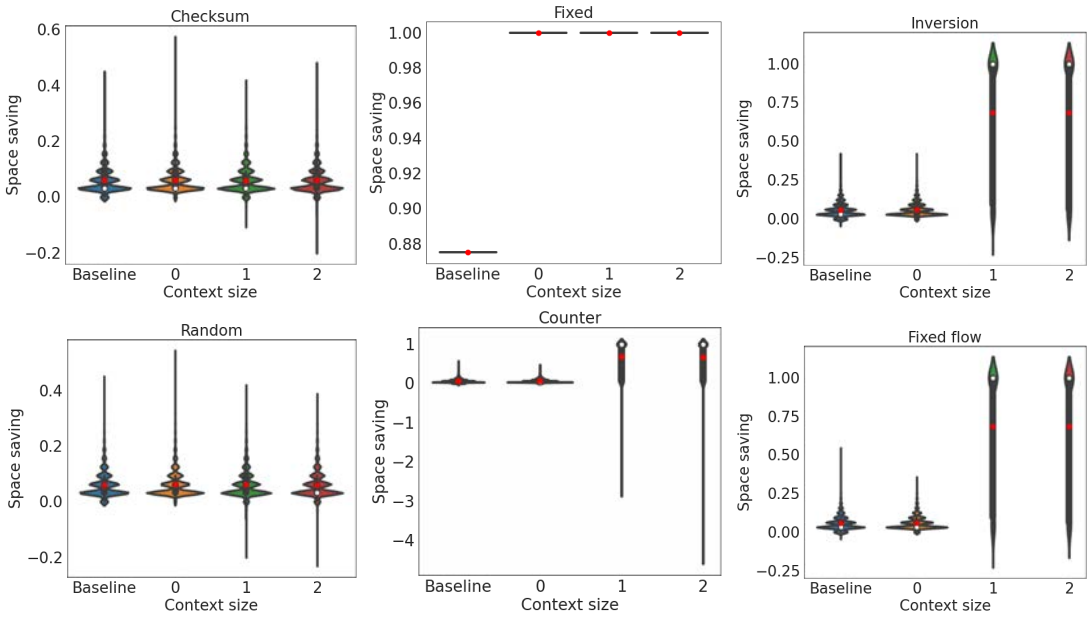

FIGURE 5.2 Distribution du gain d'espace en fonction de l_C, pour chaque type de jeu de données sur les données de test. Les points blancs et les points rouges représentent respectivement la médiane et la moyenne de chaque distribution.

envisageable de considérer des communications unidirectionnelles. Dans ce cas, le contexte sera les derniers paquets transmis par l'équipement IoT.

En bref, la définition du contexte dépend de la variabilité de celui-ci par rapport à l'en-tête à compresser. Plus le contexte comprend d'information exploitable pour la compression de l'en-tête et plus la compression sera performante. Cela doit être jugé au regard du type des champs présents dans l'en-tête à compresser et du lien entre les champs de l'en-tête à compresser et son contexte.

Dans la Figure 5.3, nous pouvons voir que l'impact du contexte est très important pour les jeux de données : « inversion », « compteur » et « fixe par flux ». Pour ces jeux de données, la valeur du champ est définie lors de l'envoi du premier paquet. Sans contexte, la compression est similaire à de l'aléatoire. Cette variabilité est visible sur la Figure 5.2. Les distributions sont très étirées et les premiers paquets de chaque flux détériorent les performances. Ces paquets sont minoritaires ce qui entraîne une chute de la moyenne mais pas de la médiane. En revanche, les jeux de données : « checksum », « fixe » et « aléatoire » n'ont pas besoin de contexte pour être compressés. La performance de compression ne change pas en fonction de la taille du contexte.

Pour conclure, il pourra être plus pertinent de travailler avec un contexte par équipement et application si les champs de l'en-tête du paquet à compresser ont une dépendance

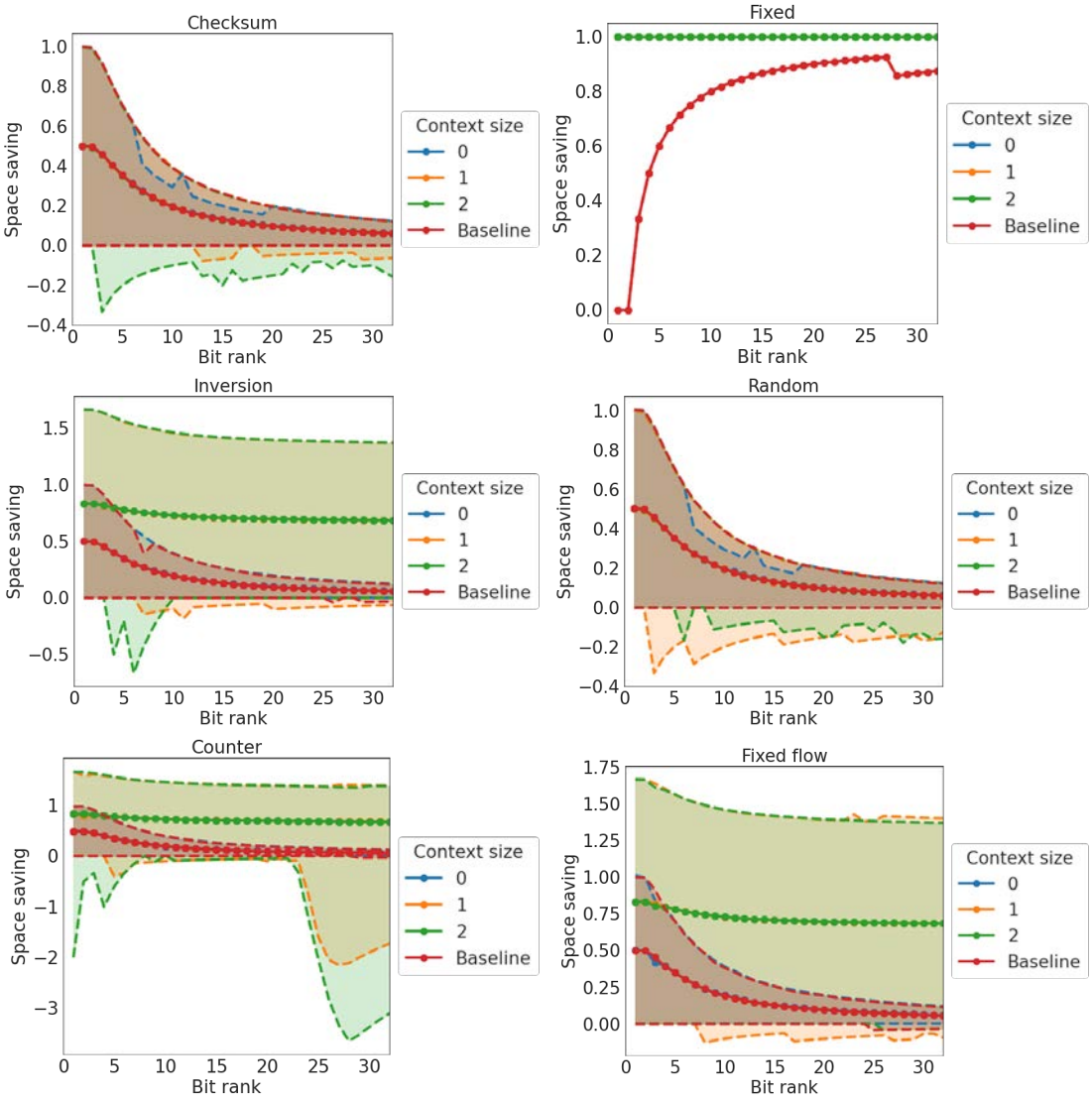

FIGURE 5.3 Gains d'espace cumulés pour chaque position de bit en fonction de l_C, pour chaque type de jeu de données sur les données de test.

importante entre les en-têtes des paquets générés avec la même application.

5.3.3 Taille de la fenêtre

L'objectif de cette section est d'étudier l'impact sur les performances de compression de la taille de la fenêtre glissante en nombre de bits, notée l, en fonction du type de champs. La Figure 5.4 illustre la distribution du gain d'espace en fonction de l, pour chaque type de jeu de données sur les données de test.

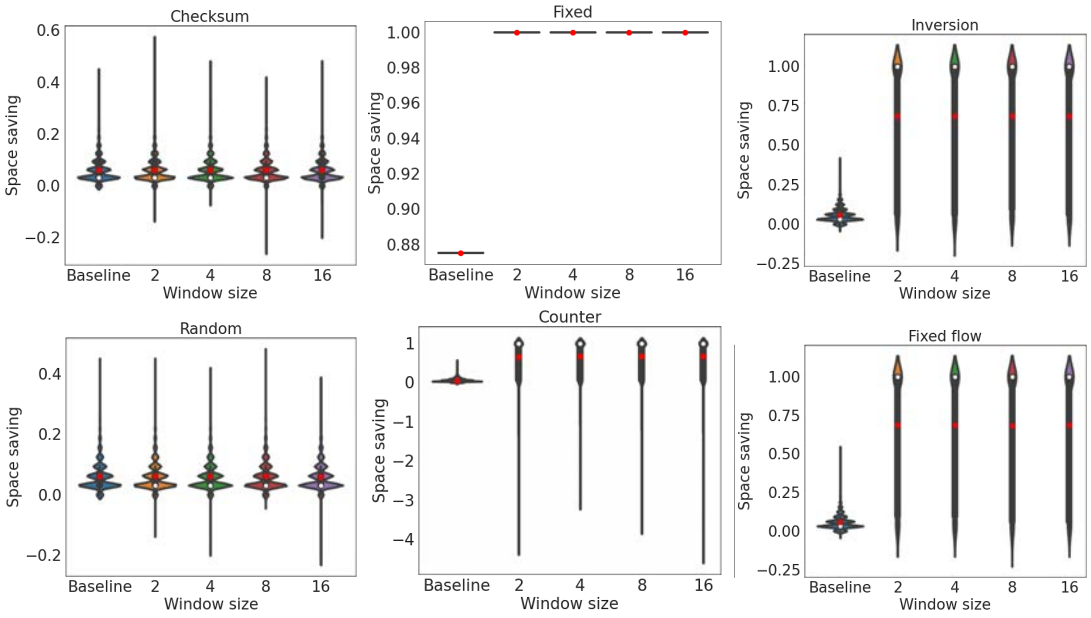

FIGURE 5.4 Distribution du gain d'espace en fonction de l, pour chaque type de jeu de données sur les données de test. Les points blancs et les points rouges représentent respectivement la médiane et la moyenne de chaque distribution.

La Figure 5.5 montre les gains d'espace cumulés obtenus pour chaque position de bit en fonction de l pour chaque type de jeu de données, sur les données de test.

La taille de la fenêtre impacte de façon différente la compression. L'impact sur la compression va dépendre du type du champ et de la taille de celui-ci. Pour les champs de type « compteur » et « checksum » la taille de la fenêtre a un impact important. En effet, pour ces champs, la compression d'un bit à une position donnée peut être améliorée en connaissant les bits précédents. Ainsi, la taille de la fenêtre doit être choisie en fonction des bits précédents qui conditionnent la valeur du bit à compresser. Cela se confirme en observant le gain d'espace sur la Figure 5.5. A l'opposé, les champs « fixe » ou « inversion », dont la valeur ne dépend pas de la valeur interne au champ, la modification de la taille de la fenêtre n'a pas d'impact sur les paramètres de compression.

La Figure 5.6 illustre le gain d'espace cumulé pour chaque position de bit en fonction de l pour le jeu de données « checksum » construit comme un champ de 16 bits sur les données de test. Ce graphe montre un gain de performance lorsque la taille de la fenêtre arrive à couvrir la totalité des bits utilisés pour construire le champ « checksum » (16 bits, ceux précédant le champ). Les 16 premiers bits ne sont pas compressés car ils sont générés aléatoirement. Pour rappel, le jeu de données est construit sous la forme d'un ensemble de vecteurs, assimilable à des paquets, d'une taille de 32 bits. Une partie est

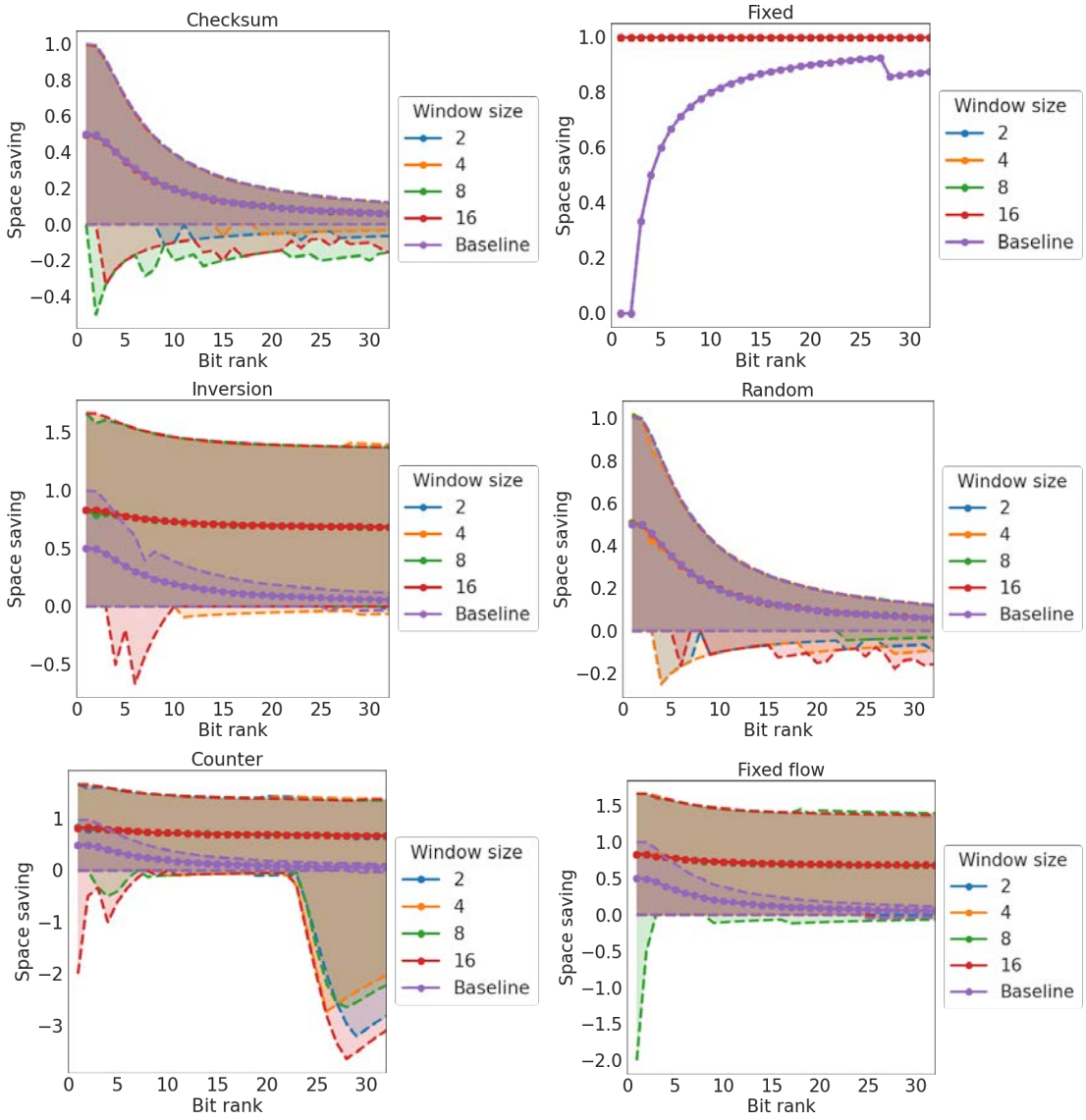

FIGURE 5.5 Gain d'espace cumulé pour chaque position de bit en fonction de l, pour chaque type de jeu de données sur les données de test.

générée aléatoirement (ici les 16 premiers bits) et l'autre partie est le résultat du calcul « checksum » appliqué sur les premiers bits.

Pour conclure, l'impact de la fenêtre est fortement lié à la taille du champ qui est à compresser. Dans le cas d'un paquet comprenant un ensemble de champs, il pourrait être envisageable d'utiliser la taille moyenne de l'ensemble des champs de l'en-tête comme taille de fenêtre.

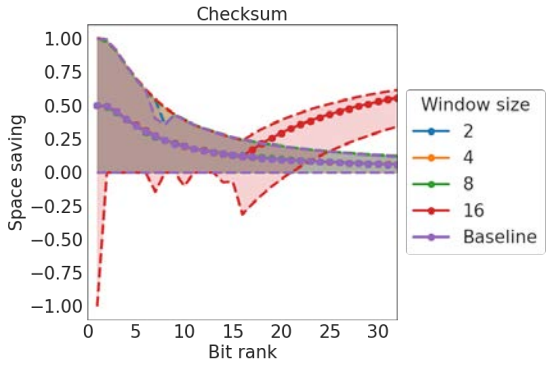

FIGURE 5.6 Gains d'espace cumulés pour chaque position de bit en fonction de l pour le jeu de données « checksum » construit comme un champ de 16 bits sur les données de test.

5.3.4 « Occlusion Map »

Après avoir analysé l'impact des paramètres dimensionnant comme la taille du contexte et de la fenêtre glissante, nous allons, dans cette section, nous intéresser à l'impact des bits en entrée de DCH lors de la compression. Cette analyse nous permettra de comprendre quelles informations le modèle utilise en entrée pour déterminer la probabilité $P(H_{k,i})$.

Pour chaque type de jeu de données, nous allons chercher à comprendre la prise de décision de DCH pour la compression d'un bit à la position i au sein de l'en-tête. Pour cela, nous utilisons des « Occlusion Map » [8]. Les « Occlusion Map » [8] vont consister à masquer une partie de l'ensemble les bits en entrées de DCH pour la prédiction d'un bit en position i. La variation de la probabilité en sortie du modèle, après application du masque, nous informera sur l'intensité de l'impact d'un bit pour la prédiction du bit à la position i. La Figure 5.12 montre les « Occlusion Maps » [8] de DCH (avec $l_C = 1$ paquet et $l = 8$ bits) sur un paquet et son contexte extrait des données de test de chaque de jeu de données.

La série de bits à compresser et le contexte associé sont représentés en deux dimensions où chaque ligne représente un octet. Un paquet se lit de haut en bas et de gauche à droite. Le cadre vert représente la fenêtre glissante de taille l appliquée lors de la compression. Le cadre rouge se focalise sur le bit à compresser dont la probabilité $P_{k,i}$ est estimée par DCH. Pour obtenir chaque matrice, chaque bit de chaque en-tête est inversé (un bit à 1 devient 0 et vice-versa). Pour chaque inversion, la différence entre la probabilité obtenue avec DCH, notée $P_{k,i}$, et la nouvelle probabilité obtenue avec DCH, notée $P^o_{k,n}$ est estimée. Cette mesure est associée à chaque bit inversé. Plus le bit inversé à de l'importance dans la prise de décision du modèle et plus la nouvelle probabilité obtenue $P_{k,i}$ sera écartée de

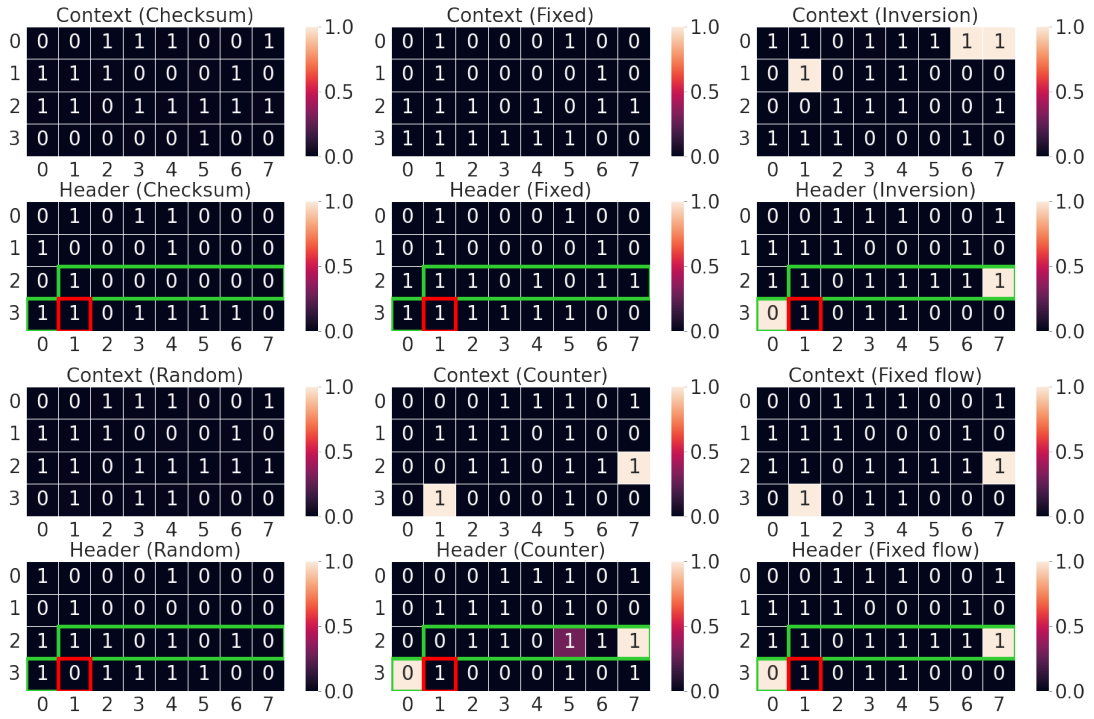

FIGURE 5.7 « Occlusion Map » [8] de DCH sur un paquet et son contexte extrait des données de test de chaque type de jeu de données.

$P^o_{k,n}$. La différence entre $P_{k,i}$ et $P^o_{k,n}$, notée $P^{diff}_{k,n} = \|P_{k,i} - P^o_{k,n}\|$ sera donc importante. Sur la Figure 5.12 la différence, $P^{diff}_{k,n}$ est associée à chaque bit avec une couleur. Ainsi, plus la couleur est chaude (rouge), plus $P^{diff}_{k,n}$ est élevée et plus l'intérêt porté par le modèle sur le bit est important. Ces figures nous montrent donc l'importance relative que DCH donne à chaque bit du contexte et de la fenêtre.

Nous pouvons voir sur la Figure 5.12 que seulement quelques bits sont utilisés par DCH pour obtenir la probabilité $P_{k,i}$. Cette dépendance très disparate permet de justifier les performances supérieures de DCH par rapport au modèle naïf utilisé ici (« Baseline »). DCH est capable d'aller chercher l'information là où il y a un intérêt pour la compression. Pour les jeux de données « inversion », « fixe par flux » et « compteur », le modèle va récupérer de l'information dans le contexte. L'information contenue dans le contexte permet de déduire la valeur à compresser, l'absence de contexte entraîne donc une chute de performance comme constaté dans la partie 5.3.2. Dans le cas de « inversion » et « flux fixe », nous pouvons voir que le modèle va récupérer dans le contexte les bits similaires au bit à compresser. Pour les jeux de données « checksum » et « fixe », le contexte n'a pas d'importance, DCH se focalise uniquement sur le contenu de la fenêtre. Dans le cas de

« checksum », DCH se focalise sur la partie des données qui est recouverte par la fenêtre glissante. La fenêtre ne recouvre pas la totalité des données, aucun intérêt n'est donc porté sur le contenu de la fenêtre. Cela entraîne une chute des performances observées dans la partie 5.3.3. Pour le jeu de données « aléatoire », aucune focalisation significative est observée ce qui est cohérent avec l'absence de compression observée lors de l'étude des performances dans la partie 5.3.2 et 5.3.3.

Nous avons analysé l'impact individuel de chaque bit, ainsi que les paramètres dimensionnant de DCH pour chaque jeu de données. Nous allons désormais essayer de combiner les différents champs étudiés afin de s'approcher du comportement du modèle en situation réelle.

5.4 Évaluation sur des champs combinés

L'objectif est d'étudier l'impact de la combinaison des champs étudiés précédemment afin de s'approcher au mieux du comportement du modèle en situation réelle. En effet, les protocoles sont construits à partir de différents champs dont ceux étudiés précédemment. Les combiner revient à imiter le comportement d'un protocole et donc s'approcher du fonctionnement de DCH sur des protocoles classiques tels que : LoRaWAN, CoAP, MQTT, etc. Dans un premier temps, nous allons définir les champs de l'en-tête. Dans un second temps, une analyse des performances et du comportement du modèle est effectuée.

5.4.1 Construction de l'en-tête

Le jeu de données utilisé est construit à partir des différents champs étudiés précédemment. La Figure 5.8 montre le position des différents champs pour la construction du jeu de données.

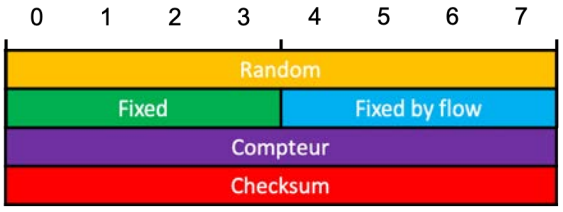

FIGURE 5.8 Schéma de l'en-tête construit sur 32 bits en combinant plusieurs champs différents.

Le jeu de données comprend un ensemble de flux de trois paquets. Ce choix est fait par souci de cohérence vis à vis des expérimentations précédentes, donc ici $l_C = 2$ paquets.

Comme précédemment, nous pouvons évaluer l'impact de la taille du contexte jusqu'à une taille de 2. Les champs « compteur » et « aléatoire » sont inversés par rapport aux paquets précédents du même flux. En combinant l'ensemble des types de champs précédent, nous essayons de nous rapprocher au mieux de la construction d'un en-tête réel tel que : LoRaWAN, CoAP, MQTT, etc. Nous appellerons le jeu de données construit : « combinaison ». La Figure 5.9 illustre la construction du jeu de données et des paquets de chaque flux.

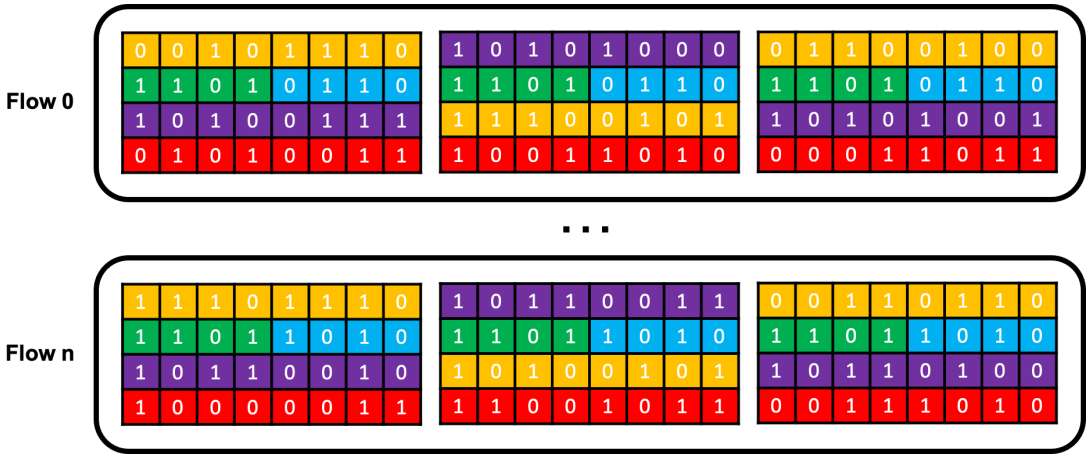

FIGURE 5.9 Exemple d'un ensemble de flux n avec $n \in \mathbb{N}$ comprenant trois paquets générés à partir de la combinaison de différents champs.

Afin d'éviter l'apparition de récurrence pour les champs aléatoires, une graine de génération différente est utilisée pour générer les données de test.

5.4.2 Taille du contexte

Dans cette section, nous analysons les performances de DCH en fonction des différentes tailles de contexte. La Figure 5.10 illustre les performances obtenues en fonction des différentes tailles de contexte pour le jeu de données « combinaison ». Le modèle DCH avec $l = 16$ bits est utilisé, fixé de façon arbitraire.

Contrairement à l'étude sur les autres jeux de données, les performances augmentent significativement avec la taille du contexte. Cette progression s'explique par la taille du flux choisi ainsi que par le type de champ utilisé pour l'inversion.

Lorsqu'aucun contexte est utilisé, seul le champ « fixe » est compressé. Lorsque $l_C = 1$ paquet, DCH arrive à compresser les champs « fixe » et « fixe par flux ». Les champs « compteur » et « aléatoire » sont inversés, DCH n'a aucune information sur leur position. Si l'inversion s'effectue entre un champ fixe et un champ variable (« compteur »,

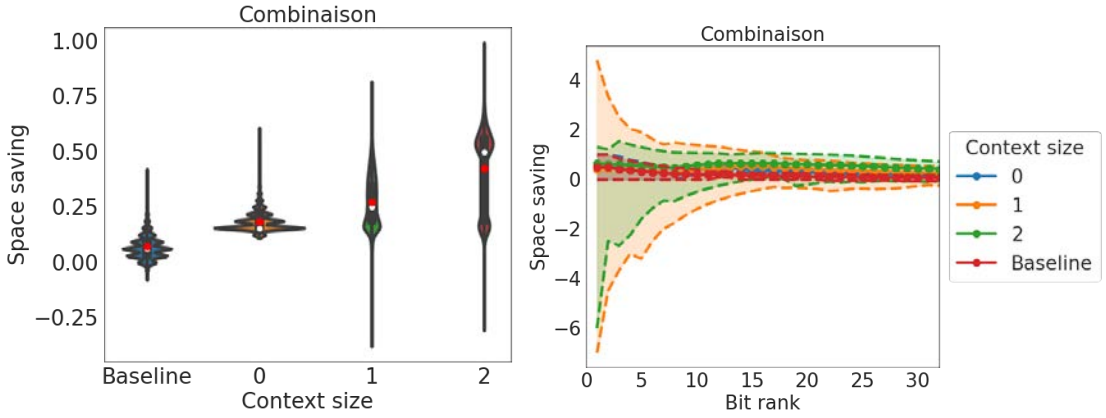

FIGURE 5.10 Distributions des gains d'espace (à gauche) et des gains d'espace cumulés pour chaque position de bit en fonction de l_C (à droite) pour chaque type de jeu de données sur les données de test. Les points blancs et les points rouges représentent respectivement la médiane et la moyenne de chaque distribution.

« aléatoire », « fixe par flux »), DCH peut déduire la position du champ variable en se servant du champ « fixe » présent dans le contexte. Dans ce cas, l'augmentation de la taille du contexte n'a pas d'impact sur les performances de compression. Néanmoins, nous construisons des flux comprenant 3 paquets. Si la taille des flux est supérieur à l'augmentation de la taille du contexte cela n'a pas d'impact sur les performances de compression. Quelle que soit la taille du contexte, les performances de DCH restent supérieures au modèle de référence. Après avoir abordé la taille du contexte, nous allons étudier le prochain paramètre dimensionnant qui est l.

5.4.3 Taille de la fenêtre

Dans cette section, nous analysons les performances de DCH en fonction des différentes tailles de fenêtre, notée l, pour le jeu de données construit à partir d'une combinaison de champs. La Figure 5.11 illustre les performances obtenues en fonction des différentes tailles de fenêtre pour le jeu de données « combinaison ». Le modèle DCH avec $l_C = 2$ paquets est utilisé.

Les tailles des fenêtres ont un impact qui va dépendre de la taille du champ comme étudié précédemment. Le taux de compression augmente sur les champs dont la taille utilisée est inférieure ou égale à la taille de la fenêtre exploitée par DCH. Par exemple, lors de la compression du dernier bit d'un champ et lorsque la taille du champ est inférieur ou égale à la taille de la fenêtre, DCH peut utiliser l'ensemble des informations du champ pour déduire la probabilité du bit à compresser. Nous pouvons observer cette augmentation

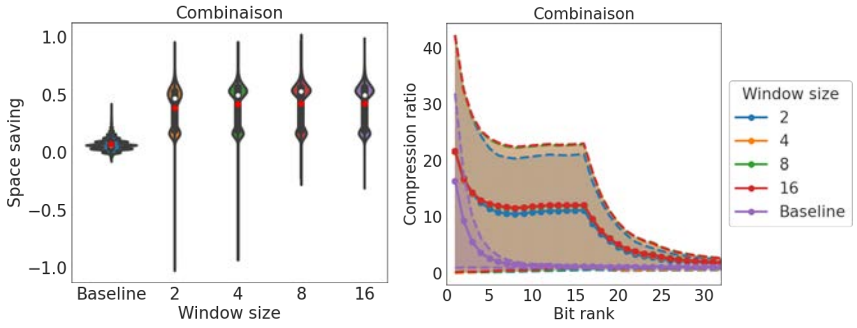

FIGURE 5.11 Distribution des gains d'espace (à gauche) et des gains d'espace cumulés pour chaque position de bit en fonction de l (à droite) pour le jeu de données « combinaison » sur les données de test. Les points blancs et les points rouges représentent respectivement la médiane et la moyenne de chaque distribution.

sur les champs « fixe » et « fixe par flux ». Cela explique les performances légèrement supérieures pour $l = 8$ bits et $l = 16$ bits.

5.4.4 « Occlusion Map »

Dans cette partie, notre objectif est d'étudier l'impact des bits en entrées de DCH pour chaque champ du jeu de données utilisé par combinaison de différents champs. Pour cela, nous utilisons des « Occlusion Map » [8]. La Figure 5.12 montre les « Occlusion Map » de DCH (avec $l_C = 1$ paquet et $l = 8$ bits) sur un paquet et son contexte extrait des données de test pour chaque type de champs.

Comme étudié dans la partie 5.3.4, DCH ne porte pas d'attention particulière pour les champs « fixe », « checksum », et « aléatoire ». Les champs « checksum », et « aléatoire » ne comprennent pas de motifs et sont perçus comme du bruit. Le champ « fixe » est compressé grâce au vecteur de position, le contexte et la fenêtre glissante ne contiennent pas d'information utile pour la compression de ce champ ce qui explique l'absence de couleur. Le champ « fixe par flux » a également un comportement similaire à celui étudié précédemment dans la partie 5.3.4. DCH se focalise sur le contexte pour aller chercher l'information. En revanche, le champ « compteur » n'a pas le même comportement sur DCH que celui étudié dans la partie 5.3.4 à cause de l'inversion. En effet, Le champs « compteur » est inversé avec un champ variable (champ « aléatoire »). De plus, les paramètres de DCH ne supporte un contexte qu'avec un paquet (pour rappel $l_C = 1$ paquet). DCH ne peut donc pas déterminer la position de l'en-tête à compresser au sein du flux. DCH ne peut donc pas déterminer la position du champ « compteur » et du champ « aléatoire », les champs inversés sont perçus comme de l'aléatoire.

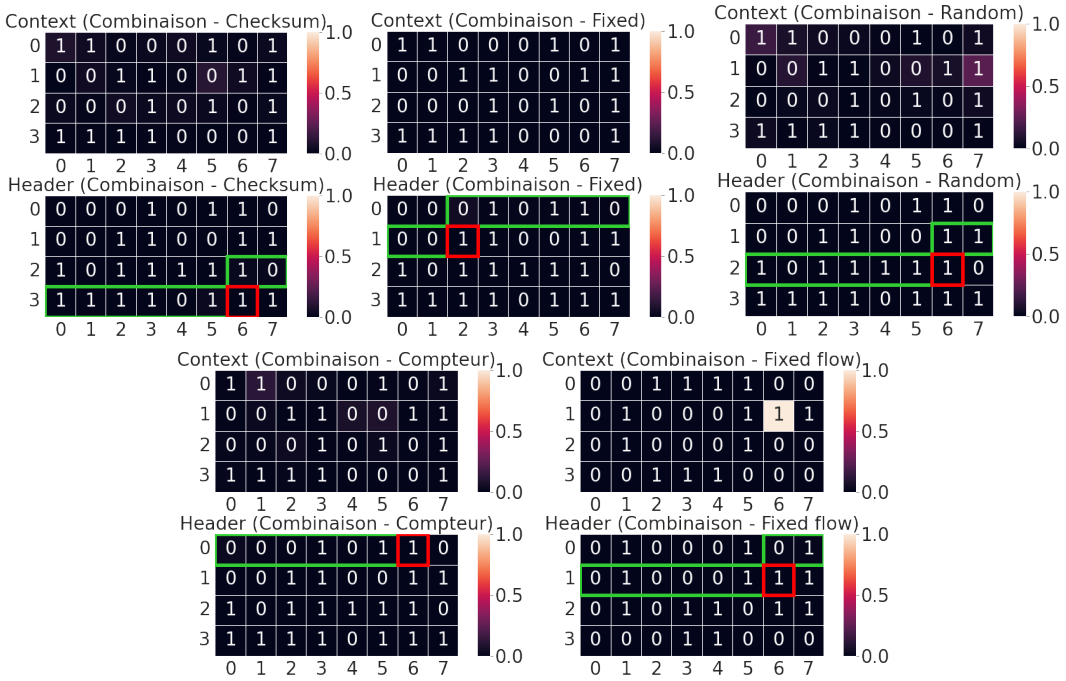

FIGURE 5.12 Carte d'intérêt de DCH sur un paquet et son contexte extrait des données de test du jeu de données créer par combinaison de différents champs.

Néanmoins, le problème d'inversion avec un champ variable peut être contourné en ajoutant comme information la position de l'en-tête du paquet à compresser au sein du flux. Cette ajout peut s'effectuer en concaténant une information de position du paquet dans le flux à chaque bit en entrée du modèle. Par exemple, elle peut se matérialiser par des sinusoïdes comme celles utilisées pour informer de la position des bits contenus dans la fenêtre glissante, par rapport à l'en-tête.

5.4.5 Table de probabilités

Présentation

Après application des « Occlusion Map » [8], nous obtenons une table T dont le processus d'extraction est détaillé dans la partie 4.5. Dans cette section, nous allons étudier les paramètres de cette table en fonction du jeu de données étudié tel que :

— le nombre d'entrées de la table ;
— les performances de la table lors de la compression ;
— les performances de la table en fonction du nombre de décimales utilisées pour encoder la probabilité $P_{k,i}$ dans la table.

La construction de la table T s'effectue en utilisant un modèle DCH avec $l_C = 1$ paquet et $l = 8$ bits. Le choix de la taille de contexte s'effectue pour des raisons de performances vis à vis de l'équipement choisi pour l'implantation de la table. La taille de la fenêtre est choisie de façon arbitraire car les champs étudiés ne sont pas significativement impactés par celle-ci comme nous l'avons montré.

Taille

Dans cette section, nous allons étudier le nombre d'entrées de la table, noté l_T, pour chaque jeu de données étudié. Le nombre d'entrées de la table nous permet de déterminer la taille de celui-ci. Plus le nombre d'entrées est importante et plus l'espace mémoire nécessaire sur l'équipement IoT pour implémenter la table T est importante. La Figure 5.13 montre la proportion d'entrées de la table T utilisées en fonction du nombre d'entrées possible. Le modèle DCH avec les paramètres suivants $l_C = 1$ paquet et $l = 8$ bits est utilisé.

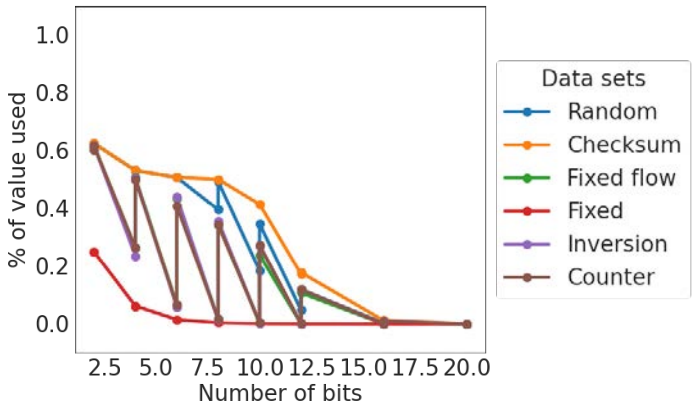

FIGURE 5.13 Proportion d'entrées utilisées par la table T par rapport au nombre total d'entrées possible, noté l_T^{MAX}, pour chaque jeu de données étudié.

Pour rappel, le nombre maximal d'entrées possibles dans la table T, noté l_T^{MAX} est calculé en suivant la formule (5.2).

$$l_T^{MAX} = 2^{l_{delta}} \times (l_C + 1) \tag{5.2}$$

l_{delta} représente le nombre de bits utilisés en entrée de la table T. $2^{l_{delta}}$ représente donc l'ensemble des combinaisons de bits exploitables. l_C étant la taille du contexte, il existe l_C configurations de taille de contexte. Le nombre total d'entrées possibles l_T^{MAX} est la combinaison du nombre d'entrées possibles $2^{l_{delta}}$ avec le nombre de configurations

de contexte possible l_C. Dans le cas où le contexte ne contient pas d'en-tête, les bits extraits utilisés en entrée de la table T proviennent uniquement du contenu de la fenêtre.

Ainsi, pour chaque jeu de données et pour chaque nombre de bits extraits, noté l_{delta}, par les « Occlusion Maps » [8], nous obtenons la proportion d'entrées utilisées dans T par rapport à l_T^{MAX}. Le jeu de données « fixe » est considéré comme le nombre minimal d'entrées possibles car il n'existe qu'une entrée par position i dans la table T donc $l_T = l_C \times L$. Par opposition, le jeu de données « aléatoire » (ou « checksum ») est considéré comme le nombre d'entrée maximal possible dans la table T car le nombre de possibilités est de $(2^{l_{delta}})$. Le nombre de combinaisons de bits de taille l_{delta} rencontrée va dépendre de la taille du jeu de données utilisé. La taille de la table T pour les autres jeux de données évoluent entre les tailles maximales et minimales définies précédemment.

En utilisant un seuil à 95% sur les valeurs extraites avec les « Occlusion Map » [8], dont la procédure est détaillée partie 4.5, nous ne sélectionnons que les bits ayant l'impact le plus significatif. Ainsi, l'augmentation du nombre d'entrées n'a pas d'impact sur la taille de la table, celle-ci reste constante et devient de plus en plus creuse.

Performances

Dans cette section, nous avons étudié les performances de compression en utilisant la table T extraite pour chaque jeu de données en fonction du nombre de bits utilisés en entrée de la table. L'objectif est de déterminer le nombre de bits optimal, noté l_{delta}, à utiliser pour la construction de la table T pour chaque jeu de données. Ce nombre est déterminé à partir du moment ou le gain d'espace ne croît plus. La Figure 5.14 montre les performances obtenues de la table T pour chaque jeu de données en fonction du nombre de bits utilisés en entrée de la table.

Les mesures sont effectuées en compressant uniquement les en-têtes avec un contexte. La plupart des jeux de données sont équivalents à de l'aléatoire lors de la compression d'en-tête sans contexte tel que : « inversion », « fixe par flux » et « compteur ». Supprimer ces paquets évitent d'introduire du bruit qui réduit les performances de la compression. À partir des résultats obtenus, nous définissons le nombre de bits à utiliser pour construire la table à 4 bits, donc $l_{delta} = 4$ bits. L'utilisation d'un nombre de bits supérieurs n'augmentent pas les performances.

Nombre de décimales

Dans cette section, nous avons étudié les performances de compression en utilisant la table T extraite pour chaque jeu de données, avec $l_{delta} = 4$ bits en se basant sur les

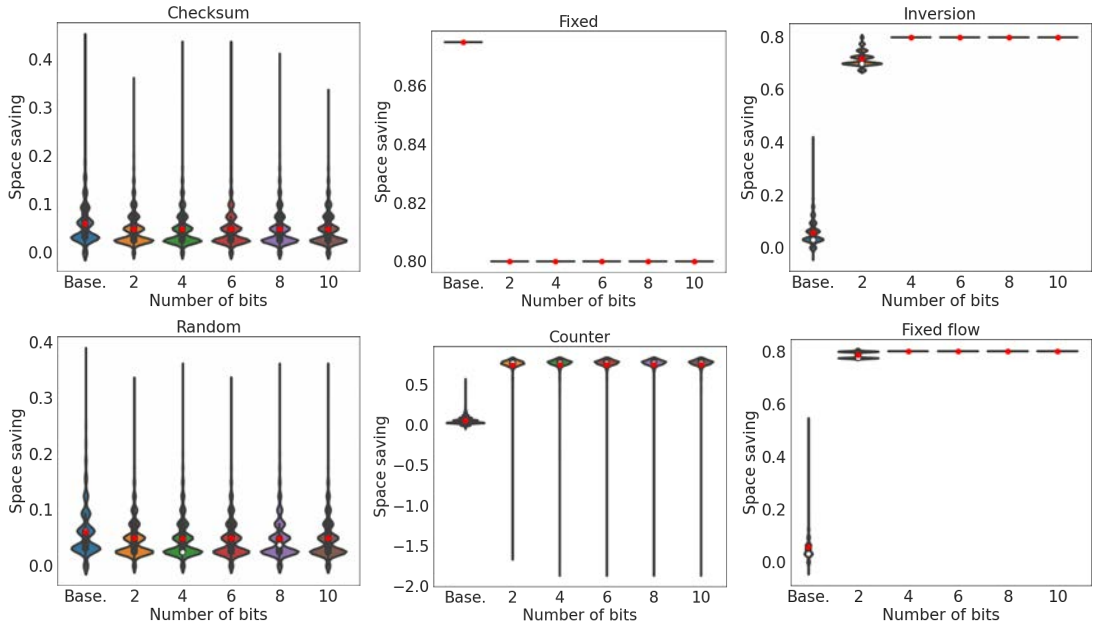

FIGURE 5.14 Distribution du gain d'espace en fonction de la taille de l_{delta} utilisée pour chaque jeu de données étudié sur les données de test. Les points blancs et les points rouges représentent respectivement la médiane et la moyenne de chaque distribution.

résultats précédents. L'objectif est de déterminer le nombre de chiffres optimal à utiliser pour encoder les probabilités dans la table T pour chaque jeu de données. Le nombre de chiffres utilisé permettra de déterminer la taille de la mémoire nécessaire pour embarquer la table sur l'équipement IoT. La Figure 5.15 montre les performances obtenues de la table T, pour chaque jeu de données, en fonction du nombre de chiffres utilisés pour encoder les probabilités de la table sur les données de test.

Le nombre de chiffres utilisé a un impact sur les performances de la table T. Pour les jeux de données « checksum » et « aléatoire », l'utilisation d'un chiffre accentue l'absence de compression en ne permettant l'utilisation que d'une probabilité à 0.5. Pour le jeu de données « fixe », cela n'a pas d'impact car la probabilité peut être 0 ou 1, stockée sur un chiffre, car celle-ci est facilement déterminé par DCH. Pour les jeux de données « inversion » et « compteur », l'impact n'est pas significatif sur les performances. Les paquets sans contexte sont classifiés comme les paquets du jeu de données « aléatoire ». Pour les paquets possédant un contexte, les données sont perçues comme un champ fixe. Ainsi, une légère baisse de performance peut s'observer sur les performances de compression lorsque le nombre de chiffres utilisés pour encoder la probabilité est de 1. Cela n'a pas d'impact lorsque le nombre de chiffres est supérieur.

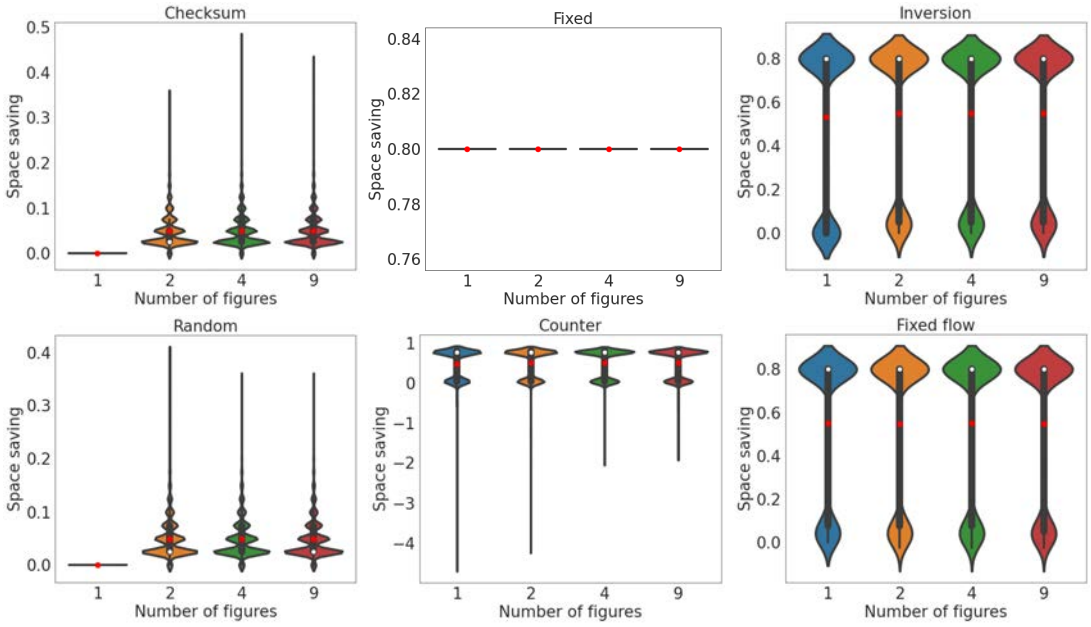

FIGURE 5.15 Distribution du gains d'espace en fonction du nombre de chiffres utilisés pour encoder la probabilité de la table T pour chaque jeu de données sur les données de test. Les points blancs et les points rouges représentent respectivement la médiane et la moyenne de chaque distribution.

Plus le nombre de chiffres utilisés est important et plus l'espace mémoire nécessaire pour embarquer la table T sur l'équipement IoT devra être important.

Combinaison des champs

Dans cette section, nous allons analyser la taille, les performances globales et les performances en fonction du nombre de chiffres utilisé pour encoder les probabilités dans la table T.

La Figure 5.16 montre la proportion d'entrée utilisés par la table T par rapport au nombre total d'entrées possibles, noté l_T^{MAX}, pour le jeu de données « combinaison ». La taille de la table T en fonction de l_{delta} pour le jeu de données « combinaison » évolue entre la taille de la table pour le jeu de données « fixe » et le jeu de données similaire à de l'aléatoire, « checksum ». Cette observation est cohérente avec ce qui est observé dans la partie 5.4.5.

La Figure 5.17 montre la distribution du gain d'espace en fonction de la taille de l_{delta} et du nombre de chiffres utilisés pour encoder la probabilité de la table T pour le jeu de données « combinaison » sur les données de test. Le nombre de bits optimal à

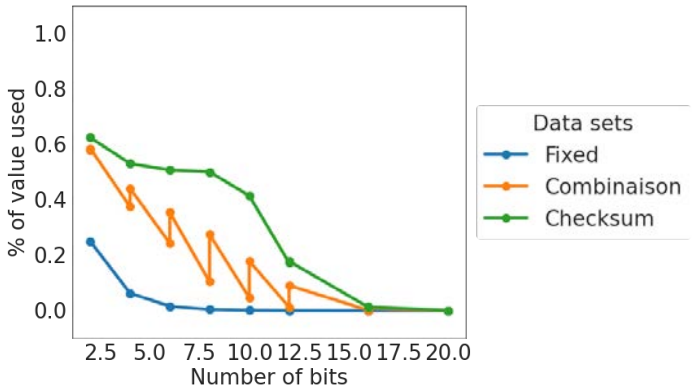

FIGURE 5.16 Proportion d'entrées utilisées par la table T par rapport au nombre total d'entrées possibles, noté l_T^{MAX}, pour le jeu de données « combinaison ».

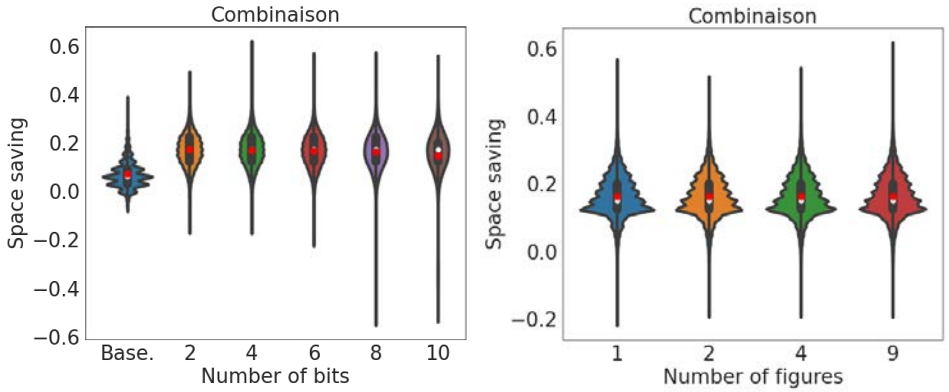

FIGURE 5.17 Distribution du gain d'espace en fonction de la taille de l_{delta} (à gauche) et du nombre de chiffres utilisés pour encoder la probabilité de la table T (à droite) pour le jeu de données « combinaison » sur les données de test. Les points blancs et les points rouges représentent respectivement la médiane et la moyenne de chaque distribution.

utiliser est de 4 au minimum comme observé lors de l'étude sur les champs individuels dans la partie 5.4.5. Cette contrainte est nécessaire pour les champs « inversion » et « fixe par flux ». Ces champs nécessitent de l'information supplémentaire pour déterminer la probabilité du bit étudié. Augmenter le nombre de bits l_{delta} sélectionné n'augmente pas les performances.

5.4.6 Conclusion

Dans cette partie, nous avons vu l'impact des différents types des champs d'en-tête les performances de DCH. Nous avons également pu voir que différents paramètres sont à

prendre en compte tels que la taille du contexte, notée l_C ou la taille de la fenêtre, notée l. Le choix de ces paramètres est à faire conjointement avec le type de champ contenu dans l'en-tête à compresser. Par exemple, la taille du champ et sa dépendance au contenu de l'en tête, comme c'est le cas pour le champs de type « checksum », vont influencer la taille de la fenêtre. De la même manière, la dépendance d'un champ aux autres paquets va influencer la taille du contexte, c'est le cas pour le champ de type « inversion ».

L'application des « Occlusion Map » [8] nous a permis de montrer la capacité de DCH à sélectionner les bits les plus importants pour réaliser la compression. Cette sélection, automatique et apprise, est la force de DCH et lui permet même dans le cas où les champs sont combinés d'offrir de bonnes performances de compression.

Enfin, nous avons abordé l'impact des tables générées sur la mémoire des équipements IoT sur lesquels elles seront embarquées. L'impact sur les mémoires de ces tables dépend de leur taille. Leurs tailles varient en fonction du type de champ, du nombre de bits sélectionnés (noté l_{delta}) et du nombre de décimales utilisées pour encoder ces probabilités. l_{delta} et le type de champ sont les variables qui impactent le plus les performances de compression.

Pour finir, lors de la combinaison des différents champs, nous avons pu voir qu'il était important de prendre en compte la taille du flux. Cette information offre une piste d'amélioration future pour notre approche DCH.

5.5 Synthèse

Le Tableau 5.1 synthétise le gain d'espace moyen pour différents modèles de compression avec différents paramètres appliqués sur chaque jeu de données associé à chaque type de champs.

TABLEAU 5.1 Gain d'espace moyen pour différents modèles de compression avec différents paramètres appliqué sur chaque jeu de données associé à chaque type de champs. Les meilleures performances pour chaque jeu de données sont en **gras**.

Modèles	**Check. 2 octets**	**Check.**	**Fixe**	**Inv.**	**Aléa.**	**Compt.**	**Fixe par flux**	**Comb.**
DeepZip [244] (no padding $l = 16$)	0.50	0.061	0.5	0.061	0.062	0.062	0.062	0.062
DeepZip [244] ($l = 16$)	0.411	**0.062**	0.968	0.062	**0.062**	0.062	0.060	0.101
DCH ($l_C = 0, l = 16$)	0.560	0.061	1.0	0.061	0.061	0.060	0.060	0.185
DCH ($l_C = 1, l = 16$)	0.559	0.059	1.0	0.685	0.060	**0.679**	0.684	0.272
DCH ($l_C = 2, l = 16$)	**0.560**	0.060	**1.0**	**0.685**	0.060	0.673	**0.685**	**0.425**
DCH (no padding $l_C = 1, l = 8$)	0.5	0.060	0.5	0.353	0.060	0.351	0.353	0.068
DCH ($l_C = 1, l = 8$)	0.059	0.061	1.0	0.685	0.0605	0.676	0.684	0.273
T ($l_C = 1, l = 8, l_{delta} = 2$)	0.048	0.049	0.800	0.494	0.049	0.516	0.540	0.165
T ($l_C = 1, l = 8, l_{delta} = 4$)	0.048	0.049	0.800	0.549	0.048	0.519	0.548	0.164
T ($l_C = 1, l = 8, l_{delta} = 6$)	0.048	0.049	0.800	0.549	0.049	0.519	0.548	0.162
T ($l_C = 1, l = 8, l_{delta} = 8$)	0.047	0.049	0.800	0.549	0.048	0.519	0.548	0.159
Référence ($l = 8$)	0.060	0.060	0.875	0.058	0.060	0.052	0.057	0.071

L'effet du « padding » est visible sur le jeu de données « fixe » en comparant les performances de DeepZip [244] sans et avec « padding ». Dans le cas d'absence de « padding », le gain d'espace ne peut pas dépasser 0.5 car $l = 16$ bits et le jeu de données comprend des en-têtes de taille $L = 32$ bits. Les bits situés dans la fenêtre ne peuvent pas être compressés, sans « padding » les premiers bits contenus dans la fenêtre sont donc des bits d'en-tête. En revanche, l'ajout de « padding » rend la totalité des bits de l'en-tête compressible car les premiers bits situés dans l'en-tête sont des bits de bourrage, les premiers bits de l'en-tête peuvent donc être compressés.

L'impact de la taille de la fenêtre varie en fonction de la taille des champs présents dans l'en-tête à compresser. Comme abordé dans la section 5.3.3, la taille de fenêtre doit permettre d'accéder à l'information nécessaire située dans l'en-tête pour pouvoir compresser un champ donné. Par exemple, dans le cas où le champ « checksum » est codé sur 2 octets, DCH avec les paramètres $l_C = 1$ paquet et $l = 16$ bits offrent de meilleures performances que DCH avec les paramètres $l_C = 1$ paquet et $l = 8$ bits (avec « padding »). Pour résumer, dans le cas des jeux de données représentant un champ, les meilleures performances sont obtenues avec $l = 8$ bits à l'exception du jeu de données « checksum » où le calcul checksum est codé sur 2 octets. Pour le jeu de données « combinaison », les meilleures performances sont obtenues avec $l = 16$ bits.

L'ajout de l'information de position est une nouveauté introduite dans DCH. Par rapport à DeepZip [244], cette information permet un gain de performance pour les jeux de données « combinaison » et « checksum ». En effet, connaître la position des champs et la position du calcul du checksum parmi les bits à compresser permet une compression plus fine et donc un gain de performance.

La construction des tables T, effectuées en appliquant des « Occlusion Map » [8] sur DCH, fait diminuer les performances entre 10% et 20% environ. Cette chute dépend de deux choses : d'une part, le nombre de bits sélectionnés pour construire la table, nommée l_{delta}, et d'autres part le seuil appliqué pour sélectionner les bits les plus pertinents (ici 95%). Le seuil appliqué permet de sélectionner les bits les plus pertinents pour la prédiction d'un bit en position i. Néanmoins, rien ne garantit que ce seuil est pertinent, un seuil à 99% pourrait permettre de sélectionner plus de bits et améliorer les performances de compression dans le cas du jeu de données « fixe ». De plus, l'augmentation de l_{delta} n'affecte pas les performances car le seuil appliqué permet d'éviter l'ajout de bits non significatifs dans la prise de décision du modèle. Les meilleures performances sont obtenues lorsque $l_{delta} = 4$ bits.

5.6 Conclusion

Dans ce chapitre, nous analysons, en détails, le comportement de DCH sur différents types de jeux de données simulés. Ainsi, nous avons pu montrer la capacité de DCH à pouvoir travailler sur différents types de champs présents dans la majorité des protocoles réseaux existants.

Parmi les limites de DCH, nous trouvons l'utilisation d'une fenêtre glissante pour la prédiction d'un bit à la position i. Dans certain cas, l'information permettant la compression de ce bit se situe en dehors des limites de la fenêtre glissante appliquée sur l'en-tête. D'autres types d'architectures pourraient donc être envisagées comme Temporal Convolutional Networks (TCN), des réseaux convolutifs adaptés aux séries temporelles. Ces architectures permettraient de capturer les informations de structures avec un faible coût de calcul.

Néanmoins, des pistes d'améliorations sont envisageables. La première piste d'amélioration serait l'utilisation d'informations liées aux flux telles que la position du paquet dans le flux. Cette information peut être utile pour les champs de type « inversion » où la position du paquet dans le flux peut donner une information sur la position du champ inversé. D'autres pistes d'amélioration sont envisagées comme l'ajout d'informations permettant d'aider à la compression. Ces informations pourraient venir d'expert métier et permettraient, au préalable, d'identifier les différents types de structures présentes dans l'en-tête. Cette expertise permettrait au modèle de plus facilement compresser chaque champ de l'en-tête sans avoir à préciser la structure de chacune d'entrées elles. Cela peut s'instancier en ajoutant une dimension supplémentaire en entrée du RNN. Cette information pourrait également servir à supprimer les champs de type « checksum » ou « aléatoire » qui réduisent les performances de compression.

Conclusions et Perspectives

Conclusion

Durant cette thèse, nous avons cherché à améliorer le fonctionnement des infrastructures réseau des « Smart Cities » en abordant trois aspects : la classification (SPPNet), la génération (NeCSTGen) et la compression (DCH) de trafic réseau. Ces propositions permettent d'améliorer la QoS, le dimensionnement, d'alléger la charge de trafic et la consommation d'énergie de ces infrastructures. Pour chacun de ces cadres applicatifs, nous avons proposé une architecture d'apprentissage profond capable de faire face à la variété et à la volumétrie des données générées au sein des « Smart Cities » et, de façon plus générale, à tout type de réseau.

Au travers de ces différentes applications, nous avons montré la versatilité des architectures d'apprentissage profond. Elles permettent de gagner en adaptabilité en travaillant sur des données réseaux issues de sources multiples tout en répondant à des objectifs variés. De plus, elles peuvent traiter une grande volumétrie de données en offrant une extraction automatique des caractéristiques nécessaires pour répondre à l'objectif visé. Nous pouvons donc considérer ces méthodes comme suffisamment matures et comme un outil indispensable pour répondre aux enjeux présent et futur des réseaux de communication.

La capacité d'adaptation de l'apprentissage profond permet à cette approche de ne pas être figée dans le temps comme les outils mathématiques plus classiques. Elles permettent un gain de temps et humain : l'expertise humaine est moins sollicitée par rapport aux approches classiques. Au cours de cette thèse, nous avons pu voir que cette capacité se matérialise, dans un premier temps, par une définition claire de l'objectif visé mesurable au travers d'une ou plusieurs métriques. Puis, par une collecte de données, en quantité importante, et leurs traitements grâce à l'intervention d'une expertise humaine. Ensuite, par la définition d'une architecture d'apprentissage profond. Enfin, par l'apprentissage de celle-ci et son déploiement. C'est grâce à la collecte des données et un re-apprentissage régulier de ce type d'architecture que ces outils peuvent être au plus proche du fonction-

nement actuel des réseaux. Ainsi, grâce à l'apprentissage profond, nous pouvons faire face à l'évolution permanente des réseaux, dans des cadres applicatifs variés, tout en offrant des performances parfois supérieures aux approches classiques.

Il reste aujourd'hui de nombreuses pistes d'amélioration à explorer sur lesquelles l'apprentissage profond peut apporter un gain de performance non-négligeable dans le domaine des réseaux (routage, détection d'anomalie, prédiction de charge de trafic, etc.). Chaque domaine d'application nécessite une expertise humaine pour permettre une adaptation spécifique de l'architecture, du traitement des données et des choix de paramètres du modèle. Cela rend coûteux en puissance de calcul l'utilisation de ces outils. De plus, il n'existe aucune garantie sur la performance de ces outils par rapport aux approches classiques. Enfin, le déploiement de ces outils au sein des villes peut être problématique. L'intégration de l'apprentissage profond au plus près de l'utilisateur nécessite une adaptation de l'architecture pour réduire ses besoins en temps de calculs. En revanche, un déploiement dans le cloud ne nécessitera pas d'adaptation, mais entraînera de la latence, car il sera nécessaire d'acheminer les données des utilisateurs jusqu'au serveur ou le modèle est déployé.

Perspectives

L'adaptabilité de ces systèmes font qu'ils restent génériques. Ils se retrouvent parfois moins performants que des systèmes très spécialisés conçus à l'aide d'une expertise humaine. C'est une limite à laquelle nous avons pu nous confronter, durant cette thèse, lors de la compression d'en-tête. Rendre les approches par apprentissage profond plus spécialisées tout en conservant le gain de temps offert par celles-ci est une piste d'exploration intéressante pour le domaine des réseaux. Elle peut se mettre en œuvre par un traitement des données plus fin mais également en cherchant à intégrer cette information directement lors de la définition de l'architecture d'apprentissage profond. Par exemple, dans le cadre de la compression d'en-têtes, en précisant le type de champ lors de la compression (champs fixe, variable, checksum, etc.).

L'apprentissage profond se base sur l'existence de données. Un jeu de données de mauvaise qualité dont les valeurs seraient manquantes et où certains événements seraient sous représentés par rapport à la réalité endommagerait les performances du modèle appris. Par exemple, pour la génération de trafic réseau, la capacité à traiter la variabilité humaine et les événements rares sont des pistes importantes d'améliorations pour notre approche NeCSTGen. En effet, certains trafics dépendent fortement de la variabilité humaine telle que la voix, la navigation internet ou le chat. D'autres trafics, au contraire,

peuvent présenter des événements rares comme une absence de communication pendant une longue durée. La classification de trafic réseau basée sur les statistiques est également confrontée à ces limites. La collecte de jeu de données réseau s'effectue le plus souvent sur un type de réseau mais ne permet pas de capturer la diversité des configurations et typologies de réseau sur lesquelles un type de trafic peut circuler (voix, navigation internet, etc.). L'existence de jeu de données de références avec un traitement spécifique ou le développement de nouvelles architectures et méthodes d'apprentissage sont des pistes d'améliorations.

Des applications transverses peuvent être abordées à travers nos travaux, circonscrits au domaine des réseaux, mais également plus largement. Dans le cas de classification ou de la compression d'en-têtes, les communications satellitaires peuvent offrir un cadre d'application pertinent. Ces communications peuvent être soumises à des contraintes d'énergie, c'est le cas des nano-satellites, mais aussi à des trafics multiples nécessitant une classification. Nos travaux pourraient donc offrir des perspectives intéressantes pour résoudre des problématiques similaires dans d'autres cadres réseau. Dans un cadre plus large, la génération de trafic peut être vue comme la génération de longues séquences de séries temporelles multivariées. La cohérence globale et local offerte par NeCSTGen peut être envisagée pour la génération de signaux comme la consommation d'énergie, les sons urbains, la pollution, entre autres. Ces domaines, liés au « Smart Cities » ou non, peuvent ainsi utiliser NeCSTGen dans l'objectif, par exemple, d'augmenter la taille des bases de données d'apprentissage. La compression d'en-têtes peut également être appliquée dans un cadre plus large de compression sans perte de données avec une taille finie et avec un contexte. Par exemple, pour la compression de séquences d'images. Les images successives peuvent être assimilées à un contexte et l'image à un paquet réseau. Ces différents champs d'applications, transverses, illustrent la capacité de l'IA à pouvoir impacter de façon significative les « Smart Cities » dans tous ces aspects.

Publications

Conférences internationales

— F. Meslet-Millet, S. Mouysset, and E. Chaput, "SPPNet: An Approach For Real-Time Encrypted Traffic Classification Using Deep Learning," in *2021 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, 2021 [250]

— F. Meslet-Millet, S. Mouysset, and E. Chaput, "NeCSTGen: An Approach For Realistic Network Traffic Generation Using Deep Learning," in *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, pp. 3108–3113, 2022 [251]

— F. Meslet-Millet, S. Mouysset, and E. Chaput, "DCH: A deep learning approach to universal header compression for the internet of things," in *Proceedings of the International Conference on Modeling Analysis and Simulation of Wireless and Mobile Systems on International Conference on Modeling Analysis and Simulation of Wireless and Mobile Systems, MSWiM 2023, Montreal, Quebec, Canada, October 30, 2023*, ACM, 2023 [252]

Conférences nationales

— F. Meslet-Millet, S. Mouysset, and E. Chaput, "NeCSTGen : Génération de Trafic Réseau Réaliste par Apprentissage Profond," in *CoRes 2023 - 8èmes Rencontres Francophones sur la Conception de protocoles, l'évaluation de performances et l'expérimentation de Réseaux de communication*, (Cargèse, France), May 2023 [253]

Bibliographie

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[2] C. Zhang, P. Patras, and H. Haddadi, "Deep Learning in Mobile and Wireless Networking : A Survey," Jan. 2019. arXiv :1803.04311 [cs].

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[4] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[5] M. Schuster and K. Paliwal, "Bidirectional recurrent neural networks," *Signal Processing, IEEE Transactions on*, vol. 45, pp. 2673 – 2681, 12 1997.

[6] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM : Visual Explanations from Deep Networks via Gradient-Based Localization," in *ICCV*, (Venice), pp. 618–626, IEEE, Oct. 2017.

[7] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3." RFC 8446, Aug. 2018.

[8] K. Heinrich, P. Zschech, T. Skouti, J. Griebenow, and S. Riechert, "Demystifying the Black Box : A Classification Scheme for Interpretation and Visualization of Deep Intelligent Systems," p. 11.

[9] S. Rezaei and X. Liu, "Deep Learning for Encrypted Traffic Classification : An Overview," *IEEE Commun. Mag.*, vol. 57, pp. 76–81, May 2019.

[10] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "beta-VAE : Learning basic visual concepts with a constrained variational framework," in *International Conference on Learning Representations*, 2017.

[11] O. A. Adeleke, N. Bastin, and D. Gurkan, "Network traffic generation : A survey and methodology," *ACM Comput. Surv.*, vol. 55, jan 2022.

[12] S. Molnar, P. Megyesi, and G. Szabo, "How to validate traffic generators ?," in *2013 IEEE ICC*, (Budapest, Hungary), IEEE, June 2013.

[13] M. Tömösközi, P. Seeling, P. Ekler, and F. Fitzek, "Performance evaluation and implementation of IP and robust header compression schemes for TCP and UDP traffic in static and dynamic wireless contexts," *Computer Science and Information Systems*, vol. 14, no. 2, pp. 283–308, 2017.

[14] M. Tomoskozi, M. Reisslein, and F. H. P. Fitzek, "Packet Header Compression : A Principle-Based Survey of Standards and Recent Research Studies," *IEEE Commun. Surv. Tutorials*, vol. 24, no. 1, pp. 698–740, 2022.

[15] L. J. CNRS, "Des villes toujours plus grosses." `https://lejournal.cnrs.fr/articles/des-villes-toujours-plus-grosses`, 2017. [Online ; accessed 24-January-2023].

[16] B. Mondiale, "Développement urbain." `https://www.banquemondiale.org/fr/topic/urbandevelopment/overview#:~:text=Vue\%20d'ensemble,-Contexte&text=Aujourd'hui\%2C\%2056\%20\%25\%20de,monde\%20vivront\%20en\%20milieu\%20urbain`, 2017. [Online ; accessed 24-January-2023].

[17] I. Fun MOOC, "Défis technologiques des villes intelligentes participatives." `https://www.fun-mooc.fr/fr/cours/defis-technologiques-des-villes-intelligentes-participatives/`, 2020. [Online ; accessed 24-January-2023].

[18] *Smart Cities and Artificial Intelligence*. Elsevier, 2020.

[19] P. Domingos, "A few useful things to know about machine learning," *Commun. ACM*, vol. 55, p. 78–87, oct 2012.

[20] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–44, 05 2015.

[21] M. A. Alsheikh, D. Niyato, S. Lin, H.-p. Tan, and Z. Han, "Mobile big data analytics using deep learning and apache spark," *IEEE Network*, vol. 30, no. 3, pp. 22–29, 2016.

[22] I. W. Tsang, J. T. Kwok, and P.-M. Cheung, "Core vector machines : Fast svm training on very large data sets," *J. Mach. Learn. Res.*, vol. 6, p. 363–392, dec 2005.

[23] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. The MIT Press, 11 2005.

[24] P. Georgiev, S. Bhattacharya, N. D. Lane, and C. Mascolo, "Low-resource multi-task audio sensing for mobile and embedded devices via shared deep neural network representations," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 1, sep 2017.

[25] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, "Geometric Deep Learning on Graphs and Manifolds Using Mixture Model CNNs," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (Honolulu, HI), pp. 5425–5434, IEEE, July 2017.

[26] B. Bru, “Brigitte Le Roux, Henry Rouanet, "Geometric Data Analysis from Correspondence Analysis to Structured Data Analysis", Dordrecht-Boston-London, Kluwer Academic Publisher, 2004,” *Mathématiques et sciences humaines*, Sept. 2005.

[27] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, “PointNet++ : Deep Hierarchical Feature Learning on Point Sets in a Metric Space,”

[28] T. N. Kipf and M. Welling, “SEMI-SUPERVISED CLASSIFICATION WITH GRAPH CONVOLUTIONAL NETWORKS,” 2017.

[29] D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba, “Network dissection : Quantifying interpretability of deep visual representations,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pp. 3319–3327, IEEE Computer Society, 2017.

[30] M. Wu, M. C. Hughes, S. Parbhoo, M. Zazzi, V. Roth, and F. Doshi-Velez, “Beyond sparsity : Tree regularization of deep models for interpretability,” in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI’18/IAAI’18/EAAI’18, AAAI Press, 2018.

[31] S. Chakraborty, R. Tomsett, R. Raghavendra, D. Harborne, M. Alzantot, F. Cerutti, M. Srivastava, A. Preece, S. Julier, R. M. Rao, T. D. Kelley, D. Braines, M. Sensoy, C. J. Willis, and P. Gurram, “Interpretability of deep learning models : A survey of results,” in *2017 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computed, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, pp. 1–6, 2017.

[32] L. Chen, D. Yang, D. Zhang, C. Wang, J. Li, and T.-M.-T. Nguyen, “Deep mobile traffic forecast and complementary base station clustering for c-ran optimization,” *J. Netw. Comput. Appl.*, vol. 121, p. 59–69, nov 2018.

[33] X. Wang, Z. Zhou, Z. Yang, Y. Liu, and C. Peng, “Spatio-temporal analysis and prediction of cellular traffic in metropolis,” in *2017 IEEE 25th International Conference on Network Protocols (ICNP)*, pp. 1–10, 2017.

[34] L. Pierucci and D. Micheli, “A neural network for quality of experience estimation in mobile communications,” *IEEE MultiMedia*, vol. 23, no. 4, pp. 42–49, 2016.

[35] Y. L. Gwon and H. T. Kung, “Inferring origin flow patterns in Wi-Fi with deep learning,” in *11th International Conference on Autonomic Computing (ICAC 14)*, (Philadelphia, PA), pp. 73–83, USENIX Association, June 2014.

[36] L. Nie, D. Jiang, S. Yu, and H. Song, “Network traffic prediction based on deep belief network in wireless mesh backbone networks,” in *2017 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1–5, 2017.

[37] F. J. Ordóñez and D. Roggen, “Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition,” *Sensors*, vol. 16, no. 1, 2016.

[38] S. Liu and J. Du, "Poster : Mobiear-building an environment-independent acoustic sensing platform for the deaf using deep learning," MobiSys '16 Companion, (New York, NY, USA), p. 50, Association for Computing Machinery, 2016.

[39] L. Sicong, Z. Zimu, D. Junzhao, S. Longfei, J. Han, and X. Wang, "Ubiear : Bringing location-independent sound awareness to the hard-of-hearing people with smartphones," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 1, jun 2017.

[40] V. Jindal, "Integrating mobile and cloud for ppg signal selection to monitor heart rate during intensive physical exercise," in *2016 IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, pp. 36–37, 2016.

[41] Z. Lin, M. Yin, S. A. Feygin, M. S. Transportation, J.-F. Paiement, and A. P. Cee, "Deep generative models of urban mobility," 2017.

[42] X. Ouyang, C. Zhang, P. Zhou, and H. Jiang, "Deepspace : An online deep learning framework for mobile big data to understand human mobility patterns," 10 2016.

[43] H. Yang, Z. Li, and Z. Liu, "Neural networks for manet aodv : An optimization approach," *Cluster Computing*, vol. 20, p. 3369–3377, dec 2017.

[44] X. Song, H. Kanasugi, and R. Shibasaki, "Deeptransport : Prediction and simulation of human mobility and transportation mode at a citywide level," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, p. 2618–2624, AAAI Press, 2016.

[45] J. Zhang, Y. Zheng, and D. Qi, "Deep spatio-temporal residual networks for citywide crowd flows prediction," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, p. 1655–1661, AAAI Press, 2017.

[46] I. Saffar, M. L. A. Morel, K. D. Singh, and C. Viho, "Deep learning based speed profiling for mobile users in 5g cellular networks," in *2019 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–7, 2019.

[47] M. L. A. Morel, I. Saffar, K. D. Singh, and C. Viho, "Multi-task deep learning based environment and mobility detection for user behavior modeling," in *2019 International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOPT)*, pp. 1–7, 2019.

[48] X. Wang, L. Gao, and S. Mao, "Phasefi : Phase fingerprinting for indoor localization with a deep learning approach," in *2015 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, 2015.

[49] X. Wang, L. Gao, and S. Mao, "Csi phase fingerprinting for indoor localization with a deep learning approach," *IEEE Internet of Things Journal*, vol. 3, pp. 1–1, 12 2016.

[50] X. Wang, L. Gao, S. Mao, and S. Pandey, "Deepfi : Deep learning for indoor fingerprinting using channel state information," 03 2015.

[51] X. Wang, X. Wang, and S. Mao, "Cifi : Deep convolutional neural networks for indoor localization with 5 ghz wi-fi," pp. 1–6, 05 2017.

[52] X. Wang, L. Gao, and S. Mao, "Biloc : Bi-modal deep learning for indoor localization with commodity 5ghz wifi," *IEEE Access*, vol. 5, pp. 4209–4220, 2017.

[53] I. Saffar, M. L. Alberi Morel, M. Amara, K. D. Singh, and C. Viho, "Mobile user environment detection using deep learning based multi-output classification," in *2019 12th IFIP Wireless and Mobile Networking Conference (WMNC)*, pp. 16–23, 2019.

[54] I. Saffar, M. L. A. Morel, K. D. Singh, and C. Viho, "Semi-supervised deep learning-based methods for indoor outdoor detection," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pp. 1–7, 2019.

[55] P.-J. Chuang and Y.-J. Jiang, "Effective neural network-based node localisation scheme for wireless sensor networks," *Wireless Sensor Systems, IET*, vol. 4, pp. 97–103, 06 2014.

[56] M. Bernaś and B. Płaczek, "Fully connected neural networks ensemble with signal strength clustering for indoor localization in wireless sensor networks," *International Journal of Distributed Sensor Networks*, vol. 2015, pp. 1–10, 12 2015.

[57] A. Payal, C. Rai, and B. Reddy, "Analysis of some feedforward artificial neural network training algorithms for developing localization framework in wireless sensor networks," *Wireless Personal Communications*, vol. 82, 06 2015.

[58] Y. Dong, Z. Li, R. Wang, and K. Zhang, "Range-based localization in underwater wireless sensor networks using deep neural network : Poster abstract," in *Proceedings of the 16th ACM/IEEE International Conference on Information Processing in Sensor Networks*, IPSN '17, (New York, NY, USA), p. 321–322, Association for Computing Machinery, 2017.

[59] X. Yan, H. Cheng, Y. Zhao, W. Yu, H. Huang, and X. Zheng, "Real-time identification of smoldering and flaming combustion phases in forest using a wireless sensor network-based multi-sensor system and artificial neural network," *Sensors (Basel, Switzerland)*, vol. 16, 2016.

[60] B. Mao, Z. M. Fadlullah, F. Tang, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, "Routing or computing ? the paradigm shift towards intelligent computer network packet transmission based on deep learning," *IEEE Transactions on Computers*, vol. 66, no. 11, pp. 1946–1960, 2017.

[61] L. Liu, Y. Cheng, L. Cai, S. Zhou, and Z. Niu, "Deep learning based optimization in wireless network," in *2017 IEEE International Conference on Communications (ICC)*, pp. 1–6, 2017.

[62] S. Subramanian and A. Banerjee, "Poster : Deep learning enabled m2m gateway for network optimization," in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services Companion*, MobiSys '16 Companion, (New York, NY, USA), p. 144, Association for Computing Machinery, 2016.

[63] Y. He, C. Liang, F. R. Yu, N. Zhao, and H. Yin, "Optimization of cache-enabled opportunistic interference alignment wireless networks : A big data deep reinforcement learning approach," in *2017 IEEE International Conference on Communications (ICC)*, pp. 1–6, 2017.

[64] V. L. L. Thing, "Ieee 802.11 network anomaly detection and attack classification : A deep learning approach," in *2017 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1–6, 2017.

[65] T. Luo and S. G. Nagarajan, "Distributed anomaly detection using autoencoder neural networks in wsn for iot," in *2018 IEEE International Conference on Communications (ICC)*, pp. 1–6, 2018.

[66] M. Yousefi-Azar, V. Varadharajan, L. Hamey, and U. Tupakula, "Autoencoder-based feature learning for cyber security applications," in *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 3854–3861, 2017.

[67] M. E. Aminanto and K. Kim, "Detecting impersonation attack in wifi networks using deep learning approach," in *Web Information System and Application Conference*, 2016.

[68] Q. Feng, Y. Zhang, C. Li, Z. Dou, and J. Wang, "Anomaly detection of spectrum in wireless communication via deep auto-encoders," *J. Supercomput.*, vol. 73, p. 3161–3178, jul 2017.

[69] M. Wijaya, K. Fukawa, and H. Suzuki, "Intercell-interference cancellation and neural network transmit power optimization for mimo channels," pp. 1–5, 09 2015.

[70] T. O'Shea, T. Erpek, and T. C. Clancy, "Deep learning based mimo communications," *ArXiv*, vol. abs/1707.07980, 2017.

[71] M. Borgerding, P. Schniter, and S. Rangan, "Amp-inspired deep networks for sparse linear inverse problems," *IEEE Transactions on Signal Processing*, vol. PP, pp. 1–1, 05 2017.

[72] T. Fujihashi, T. Koike-Akino, T. Watanabe, and P. V. Orlik, "Nonlinear equalization with deep learning for multi-purpose visual mimo communications," in *2018 IEEE International Conference on Communications (ICC)*, pp. 1–6, 2018.

[73] S. Rajendran, W. Meert, D. Giustiniano, V. Lenders, and S. Pollin, "Deep learning models for wireless signal classification with distributed low-cost spectrum sensors," *IEEE Transactions on Cognitive Communications and Networking*, vol. 4, no. 3, pp. 433–445, 2018.

[74] R. Gonzalez, A. Garcia-Duran, F. Manco, M. Niepert, and P. Vallina, "Network data monetization using net2vec," in *Proceedings of the SIGCOMM Posters and Demos*, SIGCOMM Posters and Demos '17, (New York, NY, USA), p. 37–39, Association for Computing Machinery, 2017.

[75] N. Kaminski, I. Macaluso, E. Pascale, A. Nag, J. Brady, M. Kelly, K. Nolan, W. Guibene, and L. Doyle, "A neural-network-based realization of in-network computation for the internet of things," pp. 1–6, 05 2017.

[76] L. Xiao, Y. Li, G. Han, H. Dai, and H. V. Poor, "A secure mobile crowdsensing game with deep reinforcement learning," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 1, pp. 35–47, 2018.

[77] N. C. Luong, Z. Xiong, P. Wang, and D. Niyato, "Optimal auction for edge computing resource management in mobile blockchain networks : A deep learning approach," in *2018 IEEE International Conference on Communications (ICC)*, pp. 1–6, 2018.

[78] A. Gulati, G. S. Aujla, R. Chaudhary, N. Kumar, and M. S. Obaidat, "Deep learning-based content centric data dissemination scheme for internet of vehicles," in *2018 IEEE International Conference on Communications (ICC)*, pp. 1–6, 2018.

[79] J. Mao, X. Chen, K. W. Nixon, C. Krieger, and Y. Chen, "Modnn : Local distributed mobile computing system for deep neural network," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pp. 1396–1401, 2017.

[80] R. Buyya, S. N. Srirama, G. Casale, R. Calheiros, Y. Simmhan, B. Varghese, E. Gelenbe, B. Javadi, L. M. Vaquero, M. A. S. Netto, A. N. Toosi, M. A. Rodriguez, I. M. Llorente, S. D. C. D. Vimercati, P. Samarati, D. Milojicic, C. Varela, R. Bahsoon, M. D. D. Assuncao, O. Rana, W. Zhou, H. Jin, W. Gentzsch, A. Y. Zomaya, and H. Shen, "A manifesto for future generation cloud computing : Research directions for the next decade," *ACM Comput. Surv.*, vol. 51, nov 2018.

[81] I. de Recherche en Informatique de Toulouse, "Open Services for Indexing and Research Information in Multimedia contents." `https://osirim.irit.fr/`. [Online ; accessed 24-January-2023].

[82] Y. Zhang and B. Wallace, "A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification," Apr. 2016.

[83] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification : a review," *Data Mining and Knowledge Discovery*, vol. 33, July 2019.

[84] Y. Bengio, "Learning deep architectures for ai," *Foundations*, vol. 2, pp. 1–55, 01 2009.

[85] T. Moon, "The expectation-maximization algorithm," *IEEE Signal Processing Magazine*, vol. 13, no. 6, pp. 47–60, 1996.

[86] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[87] F. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget : continual prediction with lstm," in *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*, vol. 2, pp. 850–855 vol.2, 1999.

[88] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling," Dec. 2014.

[89] A. F. Agarap, "Deep Learning using Rectified Linear Units (ReLU)," *arXiv e-prints*, p. arXiv :1803.08375, Mar. 2018.

[90] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier Nonlinearities Improve Neural Network Acoustic Models,"

[91] M. Lotfollahi, M. Jafari Siavoshani, R. Shirali Hossein Zade, and M. Saberian, "Deep packet : a novel approach for encrypted traffic classification using deep learning," *Soft Comput*, vol. 24, pp. 1999–2012, Feb. 2020.

[92] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret, "Network traffic classifier with convolutional and recurrent neural networks for internet of things," *IEEE Access*, vol. 5, pp. 18042–18050, 2017.

[93] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescape, "Traffic classification of mobile apps through multi-classification," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, pp. 1–6, 2017.

[94] R. T. El-Maghraby, N. M. Abd Elazim, and A. M. Bahaa-Eldin, "A survey on deep packet inspection," in *2017 12th International Conference on Computer Engineering and Systems (ICCES)*, pp. 188–197, 2017.

[95] S. H. Yeganeh, M. Eftekhar, Y. Ganjali, R. Keralapura, and A. Nucci, "CUTE : Traffic Classification Using TErms," in *21st ICCCN*, (Munich, Germany), pp. 1–9, IEEE, July 2012.

[96] S. Sen, O. Spatscheck, and D. Wang, "Accurate, scalable in-network identification of p2p traffic using application signatures," in *13th WWW '04*, (New York, NY, USA), p. 512, ACM Press, 2004.

[97] S. Huang, K. Chen, C. Liu, A. Liang, and H. Guan, "A statistical-feature-based approach to internet traffic classification using machine learning," in *2009 International Conference on Ultra Modern Telecommunications Workshops*, pp. 1–6, 2009.

[98] D. Zuev and A. Moore, "Traffic classification using a statistical approach," vol. 3431, pp. 321–324, 03 2005.

[99] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, "Mobile encrypted traffic classification using deep learning," in *2018 Network Traffic Measurement and Analysis Conference (TMA)*, pp. 1–8, 2018.

[100] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang, "End-to-end encrypted traffic classification with one-dimensional convolution neural networks," in *ISI*, (Beijing, China), pp. 43–48, IEEE, July 2017.

[101] W. Wang, Y. Sheng, J. Wang, X. Zeng, X. Ye, Y. Huang, and M. Zhu, "Hast-ids : Learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection," *IEEE Access*, vol. 6, pp. 1792–1806, 2018.

[102] G. Cheng and S. Wang, "Traffic classification based on port connection pattern," in *2011 International Conference on Computer Science and Service System (CSSS)*, pp. 914–917, 2011.

[103] IANA, "Internet Assigned Numbers Authority." `https://www.iana.org/`, 2023. [Online ; accessed 21-February-2023].

[104] R. Bar Yanai, M. Langberg, D. Peleg, and L. Roditty, "Realtime Classification for Encrypted Traffic," in *Experimental Algorithms*, vol. 6049, pp. 373–385, Berlin, Heidelberg : Springer Berlin Heidelberg, 2010. Series Title : Lecture Notes in Computer Science.

[105] M. Perera Jayasuriya Kuranage, K. Piamrat, and S. Hamma, "Network Traffic Classification Using Machine Learning for Software Defined Networks," in *2nd International Conference on Machine Learning for Networking (MLN)* (S. Boumerdassi, É. Renault, and P. Mühlethaler, eds.), vol. LNCS-12081 of *Machine Learning for Networking*, (Paris, France), pp. 28–39, Springer International Publishing, Dec. 2019.

[106] D. McGaughey, T. Semeniuk, R. Smith, and S. Knight, "A systematic approach of feature selection for encrypted network traffic classification," in *SysCon*, (Vancouver, BC), pp. 1–8, IEEE, Apr. 2018.

[107] O. Aouedi, K. Piamrat, and B. Parrein, "Performance evaluation of feature selection and tree-based algorithms for traffic classification," in *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*, pp. 1–6, 2021.

[108] Z. Chen, K. He, J. Li, and Y. Geng, "Seq2Img : A sequence-to-image based approach towards IP traffic classification using convolutional neural networks," in *Big Data*, (Boston, MA), pp. 1271–1276, IEEE, Dec. 2017.

[109] F. Pacheco, E. Exposito, and M. Gineste, "A framework to classify heterogeneous Internet traffic with Machine Learning and Deep Learning techniques for satellite communications," *Computer Networks*, vol. 173, p. 107213, May 2020.

[110] P. Wang, F. Ye, X. Chen, and Y. Qian, "Datanet : Deep Learning Based Encrypted Network Traffic Classification in SDN Home Gateway," *IEEE Access*, vol. 6, pp. 55380–55391, 2018.

[111] Z. Zou, J. Ge, H. Zheng, Y. Wu, C. Han, and Z. Yao, "Encrypted Traffic Classification with a Convolutional Long Short-Term Memory Neural Network," in *HPCC/SmartCity/DSS*, (Exeter, United Kingdom), pp. 329–334, IEEE, June 2018.

[112] C. Liu, L. He, G. Xiong, Z. Cao, and Z. Li, "FS-Net : A Flow Sequence Network For Encrypted Traffic Classification," in *INFOCOM 2019*, (Paris, France), pp. 1171–1179, IEEE, Apr. 2019.

[113] M. Song, J. Ran, and S. Li, "Encrypted Traffic Classification Based on Text Convolution Neural Networks," in *ICCSNT*, (Dalian, China), pp. 432–436, IEEE, Oct. 2019.

[114] O. Aouedi, K. Piamrat, G. Muller, and K. Singh, "Intrusion detection for softwarized networks with semi-supervised federated learning," in *ICC 2022 - IEEE International Conference on Communications*, pp. 5244–5249, 2022.

[115] S. Rezaei and X. Liu, "Multitask Learning for Network Traffic Classification," in *ICCCN*, (Honolulu, HI, USA), pp. 1–9, IEEE, Aug. 2020.

[116] O. Aouedi, K. Piamrat, and D. Bagadthey, "A semi-supervised stacked autoencoder approach for network traffic classification," in *2020 IEEE 28th International Conference on Network Protocols (ICNP)*, pp. 1–6, 2020.

[117] P. Li, Z. Chen, L. T. Yang, J. Gao, Q. Zhang, and M. J. Deen, "An improved stacked auto-encoder for network traffic flow classification," *IEEE Network*, vol. 32, no. 6, pp. 22–27, 2018.

[118] L. Vu, C. T. Bui, and Q. U. Nguyen, "A Deep Learning Based Method for Handling Imbalanced Problem in Network Traffic Classification," in *SoICT 2017*, (Nha Trang City Viet Nam), ACM, Dec. 2017.

[119] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng, "Malware traffic classification using convolutional neural network for representation learning," in *2017 International Conference on Information Networking (ICOIN)*, pp. 712–717, 2017.

[120] J. Höchst, L. Baumgärtner, M. Hollick, and B. Freisleben, "Unsupervised traffic flow classification using a neural autoencoder," in *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*, pp. 523–526, 2017.

[121] Z. Wang, "The applications of deep learning," 2015.

[122] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of Encrypted and VPN Traffic using Time-related Features :," in *2nd ICISSP*, (Rome, Italy), pp. 407–414, SCITEPRESS - Science and and Technology Publications, 2016.

[123] A. Habibi Lashkari, G. Draper Gil, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of Tor Traffic using Time based Features :," in *3rd ICISSP*, (Porto, Portugal), pp. 253–262, SCITEPRESS - Science and Technology Publications, 2017.

[124] Sangoma, "Asterisk the open source communications toolkit." `https://www.asterisk.org/`. [Online ; accessed 24-January-2023].

[125] R. R. S, R. R, M. Moharir, and S. G, "Scapy- a powerful interactive packet manipulation program," in *2018 ICNEWS*, 2018.

[126] S. Rezaei, B. Kroencke, and X. Liu, "Large-Scale Mobile App Identification Using Deep Learning," *IEEE Access*, vol. 8, pp. 348–362, 2020.

[127] T. Team, "Introducing tensorflow feature columns," 2017.

[128] "Iperf."

[129] B. Nicola, G. Stefano, G. Procissi, and S. Raffaello, "Brute : A high performance and extensibile traffic generator," 2005.

[130] P. Team, "Pacgen packet generator.," Retrieved February, 2023.

[131] B. Fink and R. Scottm, "Nuttcp," Retrieved February, 2023.

[132] G. Antichi, A. Di Pietro, D. Ficara, S. Giordano, G. Procissi, and F. Vitucci, "Bruno : A high performance traffic generator for network processor," in *2008 International Symposium on Performance Evaluation of Computer and Telecommunication Systems*, pp. 526–533, 2008.

[133] S. Zander, D. Kennedy, and G. J. Armitage, "Kute a high performance kernel-based udp traffic engine," 2005.

[134] NMap., "Nping - network packet generation tool/ping utiliy," Retrieved February, 2023.

[135] S. Communications, "Spirent testcenter—verifying network and cloud evolution - spirent," Retrieved February, 2023.

[136] K. Technologies, "Ixchariot - instant performance assessment of complex networks from pre- to post-deployment," Retrieved February, 2023.

[137] Q. Testbed, "Gensend and genrecv - a simple udp traffic generator application," Retrieved February, 2023.

[138] K. Technologies, "Ixnetwork - l2-3 network infrastructure performance testing that scales to business needs," Retrieved February, 2023.

[139] C. Inc., "Colasoft packet builder - colasoft," Retrieved February, 2023.

[140] B. Ballman and S. Krecher, "Ip-packet generator," Retrieved February, 2023.

[141] C. Inc., "Using test tcp (ttcp) to test throughput," Retrieved February, 2023.

[142] Z. Communications, "Lantraffic v2," Retrieved February, 2023.

[143] L. Liang, "Ipgen ip packets generator," Retrieved February, 2023.

[144] E. Acri, "Hexinject : The power of raw hex network access," Retrieved February, 2023.

[145] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "Moongen : A scriptable high-speed packet generator," in *Proceedings of the 2015 Internet Measurement Conference*, IMC '15, (New York, NY, USA), p. 275–287, Association for Computing Machinery, 2015.

[146] G. Antichi, M. Shahbaz, Y. Geng, N. Zilberman, G. Covington, M. Bruyere, N. McKeown, N. Feamster, B. Felderman, M. Blott, A. Moore, and P. Owezarski, "Osnt : Open source network tester," *Network, IEEE*, vol. 28, pp. 6–12, 10 2014.

[147] R. Olsson, "pktgen the linux packet generator," 07 2005.

[148] M. Fontanini, "Libtins : C++ packet sniffing and crafting library," Retrieved February, 2023.

[149] M. Vattinen, "epb - ethernet packet generator," Retrieved February, 2023.

[150] K. Wiles, "The dpdk pktgen application," Retrieved February, 2023.

[151] Cisco, "Trex : Realistic traffic generator," Retrieved February, 2023.

[152] "Ostinato."

[153] N. Jeff, "Nemesis packet injection tool suite," Retrieved February, 2023.

[154] J. Networks, "Wrap17 traffic generator," Retrieved February, 2023.

[155] J. Sommers, H. Kim, and P. Barford, "Harpoon : A flow-level traffic generator for router and network tests," *SIGMETRICS Perform. Eval. Rev.*, vol. 32, jun 2004.

[156] K. V. Vishwanath and A. Vahdat, "Swing : Realistic and responsive network traffic generation," *IEEE/ACM Transactions on Networking*, vol. 17, no. 3, 2009.

[157] M. C. Weigle, P. Adurthi, F. Hernández-Campos, K. Jeffay, and F. D. Smith, "Tmix : A tool for generating realistic tcp application workloads in ns-2," *SIGCOMM Comput. Commun. Rev.*, vol. 36, p. 65–76, jul 2006.

[158] C. Rolland, J. Ridoux, B. Baynat, and V. Borrel, "Using LiTGen, a realistic IP traffic model, to evaluate the impact of burstiness on performance," in *ICST*, (Marseille, France), ICST, 2008.

[159] C. Technologies, "Lanforge : Stateful ip traffic generators and network emulators," Retrieved February, 2023.

[160] S. Corporation, "Skaion traffic generation system (tgs)," Retrieved February, 2023.

[161] K. Technologies, "Breakingpoint ve - virtualized security resilience testing for enterprise-wide networks," Retrieved February, 2023.

[162] E. Inc., "Byteblower—making accurate ip testing quick and easy," Retrieved February, 2023.

[163] K. V. Katsaros, G. Xylomenos, and G. C. Polyzos, "Globetraff : A traffic workload generator for the performance evaluation of future internet architectures," in *2012 5th International Conference on New Technologies, Mobility and Security (NTMS)*, pp. 1–5, 2012.

[164] G. Communications, "Gl traffic generator : Simulation analysis network traffic characteristics," Retrieved February, 2023.

[165] P. Zach, M. Pokorný, and A. Motycka, "Design of software network traffic generator," 2013.

[166] S. Avallone, D. Emma, A. Pescapè, and G. Ventre, "Performance evaluation of an open distributed platform for realistic traffic generation," *Performance Evaluation*, vol. 60, no. 1, pp. 359–392, 2005. Performance Modeling and Evaluation of High-Performance Parallel and Distributed Systems.

[167] W.-c. Feng, A. Goel, A. Bezzaz, W.-c. Feng, and J. Walpole, "Tcpivo : A high-performance packet replay engine," in *Proceedings of the ACM SIGCOMM Workshop on Models, Methods and Tools for Reproducible Network Research*, MoMeTools '03, (New York, NY, USA), p. 57–64, Association for Computing Machinery, 2003.

[168] A. Yeow and C. Heng, "Bit-twist : Libpcap-based ethernet packet generator," Retrieved February, 2023.

[169] E. L. Helvey, "Trafgen : An efficient approach to statistically accurate artificial network traffic generation," Retrieved February, 2023.

[170] C.-Y. Ku, Y.-D. Lin, Y.-C. Lai, P.-H. Li, and K. C.-J. Lin, "Real traffic replay over WLAN with environment emulation," in *2012 IEEE WCNC*, (Paris, France), IEEE, Apr. 2012.

[171] D. Song, "Fragroute," Retrieved February, 2023.

[172] A. Ott, "Playcap packet replay," Retrieved February, 2023.

[173] Tao Ye, D. Veitch, G. Iannaccone, and S. Bhattacharyya, "Divide and Conquer : PC-Based Packet Trace Replay at OC-48 Speeds," in *TridentCom '08*, (Trento, Italy), IEEE, 2005.

[174] R. E. A. Khayari, M. Rücker, A. Lehmann, and A. Musovic, "ParaSynTG : A Parameterized Synthetic Trace Generator for Representation of WWW Traffic," 2008.

[175] B. Benchimol, "Voip traffic generator," Retrieved February, 2023.

[176] D. Barroso, "Yersinia traffic generator," Retrieved February, 2023.

[177] D. Mosberger and T. Jin, "Httperf—a tool for measuring web server performance," *SIGMETRICS Perform. Eval. Rev.*, vol. 26, p. 31–37, dec 1998.

[178] P. Barford and M. Crovella, "Generating representative web workloads for network and server performance evaluation," in *Proceedings of the 1998 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '98/PERFORMANCE '98, (New York, NY, USA), p. 151–160, Association for Computing Machinery, 1998.

[179] A. Abhari and M. Soraya, "Workload generation for youtube," *Multimedia Tools Appl.*, vol. 46, jan 2010.

[180] P. Siska, M. P. Stoecklin, A. Kind, and T. Braun, "A flow trace generator using graph-based traffic classification techniques," in *IWCMC '10*, (Caen, France), ACM Press, 2010.

[181] "Tg."

[182] D. Nagle, "Packet sender - free utility to for sending and receiving of network packets," Retrieved February, 2023.

[183] P. Team, "packeth," Retrieved February, 2023.

[184] U. Weber, "mausezahn," Retrieved February, 2023.

[185] C. Krasic, "Home page of mxtraf," Retrieved February, 2023.

[186] SolarWinds, "Network traffic generator—wan killer test," Retrieved February, 2023.

[187] Omnicor, "Network testing tools," Retrieved February, 2023.

[188] G. Kramer, "Generator of self-similar traffic," Retrieved February, 2023.

[189] A. Varet and N. Larrieu, "Realistic network traffic profile generation : Theory and practice," *Computer and Information Science*, vol. 7, 02 2014.

[190] Postel, "Tg tool," Retrieved February, 2023.

[191] E. C. D. C. Inc, "Stateful traffic generator," Retrieved February, 2023.

[192] J. Manner, "Jugi's traffic generator (jtg)," Retrieved February, 2023.

[193] "Mgen."

[194] A. Botta, A. Dainotti, and A. Pescapé, "Do you trust your software-based traffic generator ?," *IEEE Communications Magazine*, vol. 48, no. 9, pp. 158–165, 2010.

[195] H. Navidan, P. Fard Moshiri, M. Nabati, R. Shahbazian, S. A. Ghorashi, V. Shah-Mansouri, and D. Windridge, "Generative adversarial networks (gans) in networking : A comprehensive survey evaluation," *Computer Networks*, vol. 194, p. 108149, 05 2021.

[196] M. R. Shahid, G. Blanc, H. Jmila, Z. Zhang, and H. Debar, "Generative Deep Learning for Internet of Things Network Traffic Generation," in *2020 IEEE PRDC*, (Perth, Australia), IEEE, Dec. 2020.

[197] M. Ring, D. Schlör, D. Landes, and A. Hotho, "Flow-based network traffic generation using Generative Adversarial Networks," *Computers & Security*, vol. 82, May 2019.

[198] A. J. Kouam, A. C. Viana, and A. Tchana, "Zen : LSTM-based generation of individual spatiotemporal cellular traffic with interactions," Jan. 2023. arXiv :2301.02059 [cs].

[199] P. Wang, S. Li, F. Ye, Z. Wang, and M. Zhang, "Packetcgan : Exploratory study of class imbalance for encrypted traffic classification using cgan," in *ICC 2020*, 2020.

[200] P. Zingo and A. Novocin, "Can GAN-Generated Network Traffic be used to Train Traffic Anomaly Classifiers ?," in *2020 11th IEEE IEMCON*, (Vancouver, BC, Canada), IEEE, Nov. 2020.

[201] D. Li, D. Kotani, and Y. Okabe, "Improving Attack Detection Performance in NIDS Using GAN," in *2020 IEEE 44th COMPSAC*, (Madrid, Spain), IEEE, July 2020.

[202] R. Hasibi, M. Shokri, and M. D. T. Fooladi, "Augmentation scheme for dealing with imbalanced network traffic classification using deep learning," *CoRR*, vol. abs/1901.00204, 2019.

[203] C. Zhang, X. Ouyang, and P. Patras, "Zipnet-gan : Inferring fine-grained mobile traffic patterns via a generative adversarial neural network," in *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '17, (New York, NY, USA), p. 363–375, Association for Computing Machinery, 2017.

[204] "Packet-Level Adversarial Network Traffic Crafting using Sequence Generative Adversarial Networks," Mar. 2021. arXiv : 2103.04794.

[205] Y. Guo, G. Xiong, Z. Li, J. Shi, M. Cui, and G. Gou, "Combating imbalance in network traffic classification using gan based oversampling," in *2021 IFIP Networking Conference (IFIP Networking)*, pp. 1–9, 2021.

[206] B. Dowoo, Y. Jung, and C. Choi, "Pcapgan : Packet capture file generator by style-based generative adversarial networks," in *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pp. 1149–1154, 2019.

[207] A. Cheng, "Pac-gan : Packet generation of network traffic using generative adversarial networks," in *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pp. 0728–0734, 2019.

[208] M. Rigaki and S. Garcia, "Bringing a gan to a knife-fight : Adapting malware communication to avoid detection," in *2018 IEEE Security and Privacy Workshops (SPW)*, pp. 70–75, 2018.

[209] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.).

[210] T. . LibPCAP, "Tcpdump libpcap," Retrieved February, 2023.

[211] W. Foundation, "Wireshark," Retrieved February, 2023.

[212] H. Shi, H. Li, D. Zhang, C. Cheng, and X. Cao, "An efficient feature generation approach based on deep learning and feature selection techniques for traffic classification," *Computer Networks*, vol. 132, Feb. 2018.

[213] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das, "The 1999 darpa off-line intrusion detection evaluation," *Comput. Netw.*, vol. 34, Oct. 2000.

[214] D. Caputo, A. Ranieri, L. Verderame, A. Merlo, and L. Caviglione, "Google home pcap," 2021.

[215] L. Bhatia, M. Breza, R. Marfievici, and J. A. McCann, "Loed : The lorawan at the edge dataset," Sept. 2020.

[216] A. Ranieri, D. Caputo, L. Verderame, A. Merlo, and L. Caviglione, "Deep Adversarial Learning on Google Home devices," Feb. 2021.

[217] K. Park and W. Willinger, *Self-Similar Network Traffic and Performance Evaluation.* USA : John Wiley Sons, Inc., 1st ed., 2000.

[218] P. Huang, A. Feldmann, and W. Willinger, "A non-instrusive, wavelet-based approach to detecting network performance problems," in *ACM SIGCOMM*, (New York, NY, USA), Association for Computing Machinery, 2001.

[219] P. Huang, A. Feldmann, and W. Willinger, "A non-intrusive, wavelet-based approach to detecting network performance problems,"

[220] H. Hindy, C. Tachtatzis, R. Atkinson, E. Bayne, and X. Bellekens, "Mqtt-iot-ids2020 : Mqtt internet of things intrusion detection dataset," 2020.

[221] M. Tömösközi, P. Seeling, and F. H. P. Fitzek, "Performance evaluation and comparison of robust header compression (rohc) rohcv1 and rohcv2 for multimedia delivery," in *2013 IEEE Globecom Workshops (GC Wkshps)*, pp. 544–549, 2013.

[222] A. Auge and J. Aspas, "Tcp/ip over wireless links : performance evaluation," in *VTC '98. 48th IEEE Vehicular Technology Conference. Pathway to Global Wireless Revolution (Cat. No.98CH36151)*, vol. 3, pp. 1755–1759 vol.3, 1998.

[223] M. Tömösközi, P. Seeling, P. Ekler, and F. H. Fitzek, "Robust header compression version 2 power consumption on android devices via tunnelling," in *2017 IEEE International Conference on Communications Workshops (ICC Workshops)*, pp. 418–423, 2017.

[224] D. J. Farber, G. S. Delp, and T. M. Conte, "Thinwire protocol for connecting personal computers to the internet," *RFC*, vol. 914, pp. 1–22, 1984.

[225] V. Jacobson, "Compressing TCP/IP headers for low-speed serial links," *RFC*, vol. 1144, pp. 1–49, 1990.

[226] S. Mathur and M. S. Lewis, "Compressing IPX headers over WAN media (CIPX)," *RFC*, vol. 1553, pp. 1–23, 1993.

[227] S. L. Casner and V. Jacobson, "Compressing IP/UDP/RTP headers for low-speed serial links," *RFC*, vol. 2508, pp. 1–24, 1999.

[228] M. Degermark, B. Nordgren, and S. Pink, "IP header compression," *RFC*, vol. 2507, pp. 1–47, 1999.

[229] C. Bormann, C. Burmeister, M. Degermark, H. Fukushima, H. Hannu, L. Jonsson, R. Hakenberg, T. Koren, K. Le, Z. Liu, A. Martensson, A. Miyazaki, K. Svanbro, T. Wiebke, T. Yoshimura, and H. Zheng, "Robust header compression (ROHC) : framework and four profiles : Rtp, udp, esp, and uncompressed," *RFC*, vol. 3095, pp. 1–168, 2001.

[230] T. Koren, S. L. Casner, J. Geevarghese, B. Thompson, and P. Ruddy, "Enhanced compressed RTP (CRTP) for links with high delay, packet loss and reordering," *RFC*, vol. 3545, pp. 1–22, 2003.

[231] G. Pelletier and K. Sandlund, "Robust header compression version 2 (rohcv2) : Profiles for rtp, udp, ip, ESP and udp-lite," *RFC*, vol. 5225, pp. 1–124, 2008.

[232] J. W. Hui and P. Thubert, "Compression format for ipv6 datagrams over IEEE 802.15.4-based networks," *RFC*, vol. 6282, pp. 1–24, 2011.

[233] R. Peon and H. Ruellan, "HPACK : header compression for HTTP/2," *RFC*, vol. 7541, pp. 1–55, 2015.

[234] A. Minaburo, L. Toutain, C. Gomez, D. Barthel, and J. C. Zúñiga, "SCHC : generic framework for static context header compression and fragmentation," *RFC*, vol. 8724, pp. 1–71, 2020.

[235] G. Pelletier, K. Sandlund, L. Jonsson, and M. A. West, "Robust header compression (ROHC) : A profile for TCP/IP (ROHC-TCP)," *RFC*, vol. 4996, pp. 1–94, 2007.

[236] M. Belshe, R. Peon, and M. Thomson, "Hypertext Transfer Protocol Version 2 (HTTP/2)." RFC 7540, May 2015.

[237] B. Knoll, "CMIX version 19." `http://www.byronknoll.com/cmix.html`, 2021. [Online ; accessed 24-February-2023].

[238] I. H. WIllEN, R. M. Neal, and J. G. Cleary, "ARITHMETIC CODING FOR DATA COMPRESSION," *Communications of the ACM*, vol. 30, no. 6, p. 21, 1987.

[239] F. Bellard, "NNCP : Lossless Data Compression with Neural Networks." `https://bellard.org/nncp/`, 2021. [Online ; accessed 24-February-2023].

[240] B. Knoll, "lstm-compress." `https://github.com/byronknoll/lstm-compress`, 2021. [Online ; accessed 24-February-2023].

[241] Q. Liu, Y. Xu, and Z. Li, "Decmac : A deep context model for high efficiency arithmetic coding," in *2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, pp. 438–443, 2019.

[242] J. Townsend, T. Bird, and D. Barber, "Practical lossless compression with latent variables using bits back coding," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, OpenReview.net, 2019.

[243] F. H. Kingma, P. Abbeel, and J. Ho, "Bit-swap : Recursive bits-back coding for lossless compression with hierarchical latent variables," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, pp. 3408–3417, PMLR, 2019.

[244] M. Goyal, K. Tatwawadi, S. Chandak, and I. Ochoa, "Deepzip : Lossless data compression using recurrent neural networks," in *2019 Data Compression Conference (DCC)*, pp. 575–575, 2019.

[245] M. V. Mahoney, "Fast text compression with neural networks," in *Proceedings of the Thirteenth International Florida Artificial Intelligence Research Society Conference*, p. 230–234, AAAI Press, 2000.

[246] J. Schmidhuber and S. Heil, "Sequential neural text compression," *IEEE Transactions on Neural Networks*, vol. 7, no. 1, pp. 142–146, 1996.

[247] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," vol. 30, 2017.

[248] T. Kadir and M. Brady, "Saliency, scale and image description," *International Journal of Computer Vision*, vol. 45, no. 2, pp. 83–105, 2001.

[249] "Computing the Internet checksum." RFC 1071, Sept. 1988.

[250] F. Meslet-Millet, S. Mouysset, and E. Chaput, "SPPNet : An Approach For Real-Time Encrypted Traffic Classification Using Deep Learning," in *2021 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, 2021.

[251] F. Meslet-Millet, S. Mouysset, and E. Chaput, "NeCSTGen : An Approach For Realistic Network Traffic Generation Using Deep Learning," in *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, pp. 3108–3113, 2022.

[252] F. Meslet-Millet, S. Mouysset, and E. Chaput, "DCH : A deep learning approach to universal header compression for the internet of things," in *Proceedings of the International Conference on Modeling Analysis and Simulation of Wireless and Mobile Systems on International Conference on Modeling Analysis and Simulation of Wireless and Mobile Systems, MSWiM 2023, Montreal, Quebec, Canada, October 30, 2023*, ACM, 2023.

[253] F. Meslet-Millet, S. Mouysset, and E. Chaput, "NeCSTGen : Génération de Trafic Réseau Réaliste par Apprentissage Profond," in *CoRes 2023 - 8èmes Rencontres Francophones sur la Conception de protocoles, l'évaluation de performances et l'expérimentation de Réseaux de communication*, (Cargèse, France), May 2023.

[254] L. van der Maaten and G. Hinton, "Viualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 11 2008.

[255] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn : Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

Annexe A

Projection du vecteur de contexte de DCH

L'ensemble des Figures présentent ci-dessous représente la projection du vecteur de contexte C' de DCH de dimension $\mathbb{R}^{30}$ dans un espace de dimension $\mathbb{R}^2$ à l'aide de t-distribution Stochastic Neighbor Embedding (t-SNE) [254]. Les paramètres par défaut de la librairie Sklearn [255] sont utilisés.

A.1 Nombre de paquets contenus dans le contexte

Les Figures A.1, A.2, A.3, A.4 et A.5 montrent la projection du vecteur de contexte en fonction du nombre de paquets contenus dans C pour différents protocoles.

A.2 Valeurs des champs du paquet compressé

Les Figures A.6, A.7 et A.8 montrent la projection du vecteur de contexte en fonction de différents type de paquet, pour différents protocoles.

A.3 Direction du paquet compressé

Les Figures A.9, A.10, A.11, A.12 et A.13 montrent la projection du vecteur de contexte en fonction de la direction du paquet compressé.

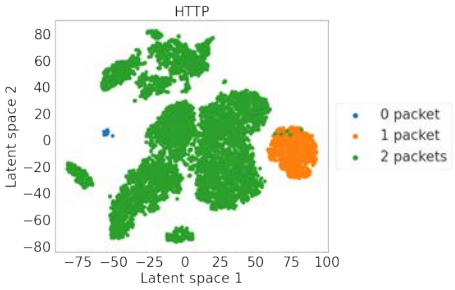

FIGURE A.1 Projection du vecteur de contexte de HTTP.

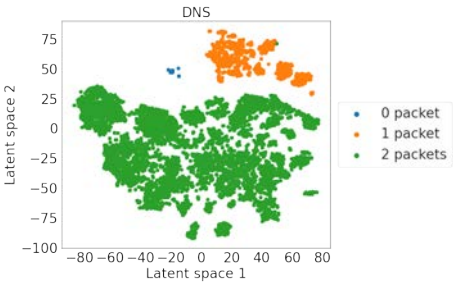

FIGURE A.2 Projection du vecteur de contexte de DNS.

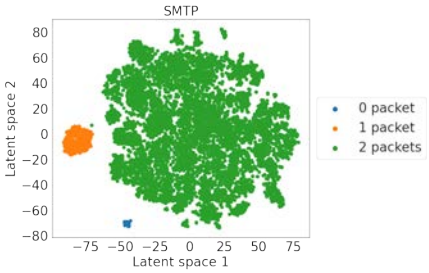

FIGURE A.3 Projection du vecteur de contexte de SMTP.

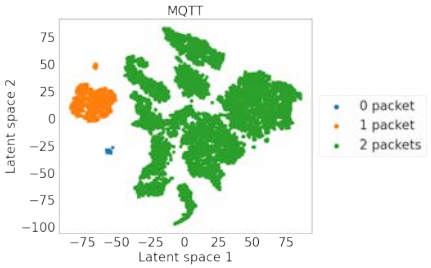

FIGURE A.4 Projection du vecteur de contexte de MQTT.

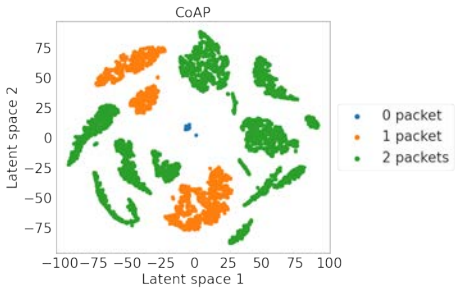

FIGURE A.5 Projection du vecteur de contexte de CoAP.

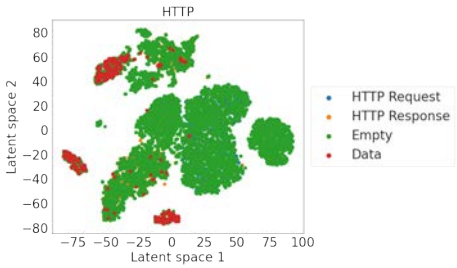

FIGURE A.6 Projection du vecteur de contexte de HTTP. La légende représente le type de paquet compressé.

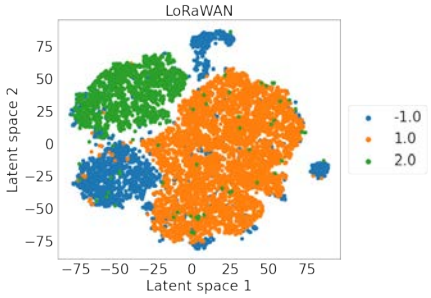

FIGURE A.7 Projection du vecteur de contexte de LoRaWAN. La légende représente le champs « fport » du paquet compressé, avec -1 qui représente l'absence de valeur.

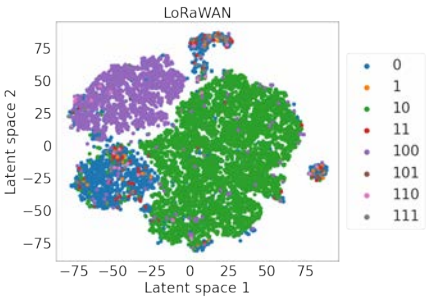

FIGURE A.8 Projection du vecteur de contexte de LoRaWAN. La légende représente le champs « mtype » du paquet compressé.

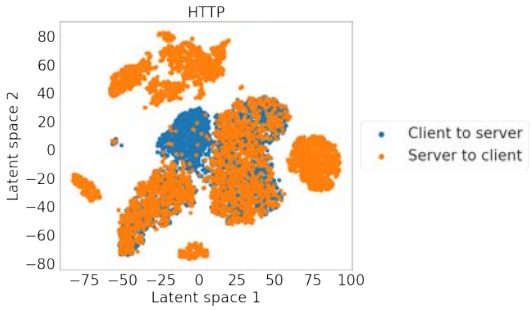

FIGURE A.9 Projection du vecteur de contexte de HTTP.

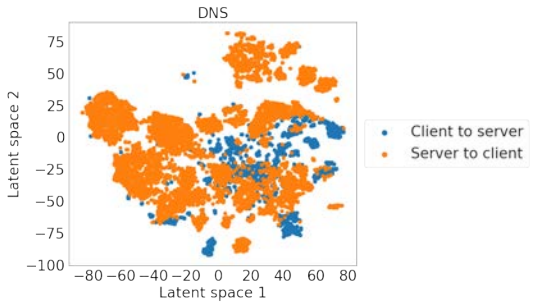

FIGURE A.10 Projection du vecteur de contexte de DNS.

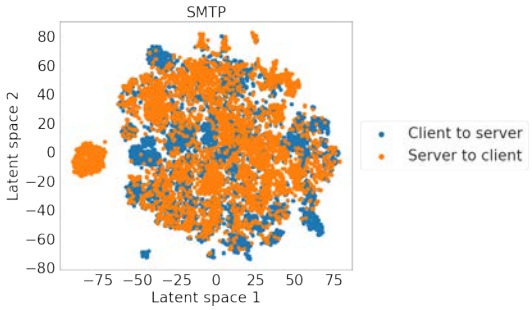

FIGURE A.11 Projection du vecteur de contexte de SMTP.

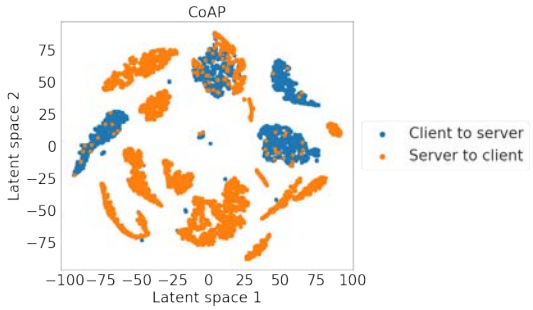

FIGURE A.12 Projection du vecteur de contexte de CoAP.

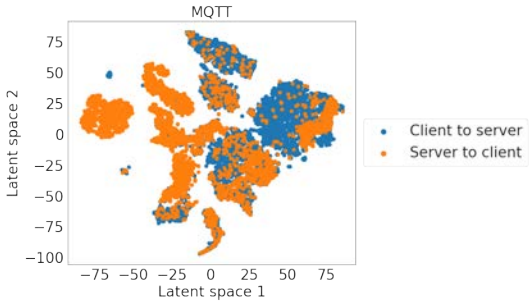

FIGURE A.13 Projection du vecteur de contexte de MQTT.

Résumé

Cette thèse s'intéresse à l'amélioration du fonctionnement de l'infrastructure réseau des villes intelligentes en proposant différentes architectures d'apprentissage profond. La première architecture est Servername Protocol Packet Network (SPPNet), pour la classification de trafic réseau chiffré visant à l'amélioration de la qualité de service. La seconde architecture est Network Clustering Sequential Traffic Generation (NeCSTGen), pour la génération de trafic réseau réaliste permettant le dimensionnement des infrastructures réseaux des villes intelligentes. La troisième architecture est Deep Compression Header (DCH), pour la compression universelle d'en-têtes de paquet réseau directement sur les équipements IoT. Les en-têtes de paquet réseau IoT sont souvent volumineux par rapport aux données véhiculées ce qui entraîne une augmentation de la consommation de la batterie et de la bande passante. Les approches proposées ont été évaluées expérimentalement, montrant des performances supérieures à celles des approches actuelles.

Mots clés : Apprentissage profond, Internet des Objets, Trafic réseau, Classification, Génération, Compression

Abstract

This thesis focuses on improving the operation of smart city network infrastructure by proposing different deep learning architectures. The first architecture is Servername Protocol Packet Network (SPPNet), for the classification of encrypted network traffic to improve quality of service. The second architecture is Network Clustering Sequential Traffic Generation (NeCSTGen), for realistic network traffic generation, enabling the sizing of network infrastructures for smart cities. The third architecture is Deep Compression Header (DCH), for universal compression of network packet headers directly on IoT devices. IoT network packet headers are often large in relation to the data conveyed, resulting in increased battery and bandwidth consumption. The proposed approaches have been experimentally evaluated, showing superior performance to current approaches.

Keywords : Deep Learning, Internet of Things, Network traffic, Classification, Generation, Compression