REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Khider – BISKRA
Faculté des Sciences Exactes, des Sciences de la Nature et de la Vie
Département d'informatique

N° d'ordre : RTIC/M2/2020

Mémoire

Présenté pour obtenir le diplôme de master académique en

Informatique

Parcours : Réseaux et technologies de l'information et de la communication

---

Équilibrage de charge intelligent dans les réseaux définis par logiciel

---

Par :

BASSI MAHER

Soutenu le 29 septembre 2020, devant le jury composé de :

Nom et prénom	Grade	Président
Terrissa Labib Sadek	Professeur	Rapporteur
Nom et prénom	Grade	Examinateur

Dédicaces

Je dédie ce mémoire à

Ma mère, qui a œuvré pour ma réussite par son amour, son soutien, ses précieux conseils et toutes les sacrifices consentis, pour son assistance et sa présence dans ma vie, j'espère que vous recevez à travers ce travail aussi modeste soit-il, l'expression de mes sentiments et de mon éternelle gratitude.

Mon père, Puisse Dieu faire en sorte que ce travail vous rend fier et porte le fruit des longues années de sacrifices et de privations pour m'aider à avancer dans la vie. Merci pour les valeurs nobles, l'éducation et votre soutien permanent.

Mes frères qui n'ont cessé d'être pour moi des exemples de persévérance, de courage et de générosité.

Mon professeur TERRISSA Sadek Labib, qui n'a cessé de m'encourager et me conseiller.

Mes professeurs de l'UMKB qui doivent voir dans ce travail la fierté d'un savoir bien acquis.

Remerciements

Tous mes remerciements et ma gratitude pour Allah le Tout-Puissant qui m'a donné le courage et la volonté d'atteindre ce niveau.

La réalisation de ce mémoire a été possible grâce au concours de plusieurs personnes à qui je voudrais témoigner toute ma gratitude.

Je remercie dans un premier temps mes très chers parents, qui ont toujours été présents pour moi. Je remercie mes frères, pour leurs encouragements.

Je voudrais également remercier mon directeur de mémoire M. TERRISSA Sadek Labib, professeur à l'université de Mohamed Khider Biskra, pour sa patience, sa disponibilité et surtout ses judicieux conseils, qui ont contribué à alimenter ma réflexion.

Je voudrais exprimer ma reconnaissance envers les amis et collègues qui m'ont apporté leur soutien moral et intellectuel tout au long de ma démarche.

À tous ces intervenants, je présente mes remerciements, mon respect et ma gratitude.

Résumé

Le réseau défini par logiciel représente une architecture prometteuse qui combine la gestion centrale et la programmable. L'SDN sépare le plan de contrôle du plan de transfert des données et déplace la gestion du réseau tel que les logiciels vers un point central, appelé contrôleur, qui peut être programmé et utilisé comme cerveau du réseau. La croissance des services d'applications cloud fournis via des centres de données avec des demandes de trafic variables dévoile les limites des méthodes d'équilibrage de charge traditionnelles dans les réseaux actuels. Dans le but d'assister à des scénarios évolutifs visant à améliorer les performances du réseau sur l'état de charge des routes, ce mémoire présente une méthode d'équilibrage de charge basée sur un réseau de neurones artificiels dans le contexte du réseau défini par la connaissance. KDN cherche à tirer parti des techniques d'intelligence artificielle pour le contrôle et le fonctionnement des réseaux informatiques automatiquement. KDN étend l'SDN avec la télémétrie avancée et l'analyse de réseau en introduisant un plan de connaissances qui fonctionne parallèlement avec le plan de contrôle. L'ANN proposé est capable de prédire les performances du réseau en fonction des paramètres de trafic en créant un modèle de comportement du trafic basé sur des mesures de bande passante et de latence sur différents chemins et différents périodes de transmission. La méthode ANN a un but pour choisir le chemin le moins de charge.

Abstract

The software-defined network represents a promising architecture that combines central management and programmable. SDN separates the control plane from the data transfer plane and moves network management such as software to a central point, called a controller, which can be programmed and used as the brain of the network. The growth of cloud application services delivered through data centers with varying traffic demands is exposing the limitations of traditional load balancing methods in today's networks. With the aim of assisting with evolutionary scenarios aimed at improving the performance of the network on the state of charge of the routes, this thesis presents a method of load balancing based on an artificial neural network in the context of the network defined by the knowledge. KDN seeks to take advantage of artificial intelligence techniques to control and operate computer networks automatically. KDN extends SDN with advanced telemetry and network analysis by introducing a knowledge plane that works in parallel with the control plane. The proposed ANN is able to predict network performance based on traffic parameters by creating a traffic behaviour model based on bandwidth and latency measurements on different paths and different transmission periods. The ANN method has a goal of choosing the path with the least load.

Table des matières

Liste des figures

Liste des tableaux

Liste des équations

Liste des abréviations

SDN	Software Defined Networking
ONF	Open Networking Foundation
API	Application Programming Interface
OVS	Open Virtual Switch
WAN	Wide Area Network
LAN	Local Area Network
DC	Data Center
DDOS	Distributed Denial of Service Attack
CDPI	Control to Data-Plane Interface
NBI	NorthBound Interfaces
CLI	Command Line Interface
OSPF	Open Short Path First
OFDP	OpenFlow Discovery Protocol
SAL	Service Abstraction Layer
LB	Load Balancing
IA	Intelligence Artificiel
ML	Machine Learning
KDN	Knowledge Defined Networking
RNA	Réseau de Neurones Artificiels
MLP	Multi layer Perceptron
sFlow-RT	Sampled Flow-Real Time
QoS	Quality of service
CSV	Comma-separated values

Introduction Générale

Introduction générale

Le réseau défini par logiciel [software defined networking (SDN)] promet un contrôle flexible des réseaux informatiques en coordonnant les commutateurs dans le plan de données du réseau via un contrôleur logiquement centralisé, et fournit un réseau programmable gérer par les développeurs, cela a permis de faciliter la gestion distribuée du réseau. Les éléments actuels du plan de données du réseau, les commutateurs du réseau défini par logiciel, sont équipés de capacités informatiques et de stockage amélioré. Ils fournissent des informations concernant l'état de charge en temps réel, ainsi que des données de configuration et de surveillance de l'état du réseau, sur une plate-forme d'analyse dédié, ce qui a donné naissance à une nouvelle génération de techniques de surveillance du réseau, communément appelées télémétrie de réseau ou réseau à distance. Cette nouvelle architecture a les propriétés d'un routage rapide et d'une tolérance aux pannes des réseaux traditionnels. Dans ce contexte, les technologies de télémétrie et d'analyse offrent en plus une vue plus riche du réseau par rapport à ce qui était possible avec les approches de gestion de réseau conventionnelles.

L'intelligence artificielle a été utilisée principalement pour l'amélioration et l'optimisation des performances dans les systèmes ainsi que les réseaux informatiques. Etant donné que le réseau défini par logiciel fournit des informations sur le plan de données (ex : La capacite des liens, la latence, perte des paquets etc.). L'utilisation de l'apprentissage automatique peut être une utilité importante. L'apprentissage automatique [Machine Learning (ML)] dans les réseaux définis par logiciels a été introduit des plusieurs processus tel que la détection et la prévention des intrusions, la résolution des problèmes de placement des contrôleurs, la prévision des performances, le routage, l'emplacement optimal de la machine virtuelle et l'équilibrage de charge [Load Balancing (LB)] notamment pour les topologies multi-chemins comme le fattree qui a été souvent utilisée dans les centres de données.

Les réseaux définis par la connaissance [Knowledge defined networking (KDN)] est un nouveau paradigme qui combine les réseaux définis par logiciel, l'analyse de réseau et l'apprentissage automatique, pour finalement fournir un contrôle de réseau automatisé. Nous avons utilisé le réseau défini par la connaissance pour objectif d'aider le plan de contrôle à prendre la meilleure décision concernant le routage et améliorer les performances de réseau tel que l'augmentation de la bande passante [Bandwidth] et de la diminution de latence [Latency] en appliquant une nouvelle stratégie d'équilibrage de charge basée sur les technologies d'apprentissage automatique en l'occurrence les réseaux de neurones artificiels.

Motivations

Les réseaux de communication traditionnels sont basés principalement sur les commutateurs statiques qui provoquent plusieurs problèmes, permis ces problèmes :

- Mauvaise exploitation des ressources.
- Perte et le retard des paquets (problème de congestion).
- Configuration lourde et individuelle.

Aussi, les protocoles utilisés dans l'équilibrage de charge dans les centres de données tel que Round Robin, ECMP et Random, sont des méthodes dont la vocation est d'améliorer les performances du réseau. Cependant, ces protocoles ne sont pas automatisés, et ils ne dotent pas d'un système intelligent et adaptatif. Ceux-ci, nous a motivé à proposer des solutions d'équilibrage de charge intelligents et adaptatifs basées sur les technologies de l'apprentissage automatique.

Objectifs et contribution

Notre contribution visée à concevoir et à implémenter des nouvelles solutions permettant l'amélioration de la bande passante dans les réseaux multi-chemins. Ces solutions consistent à créer de nouvel algorithme d'équilibrage de charge intelligent et adaptatif, permettant de prendre des décisions sur les centres de données entre les chemins disponibles en utilisant les réseaux de neurones artificiels.

Aperçu du mémoire

Ce projet a passé par 4 chapitres, on les présente comme suivant :

Chapitre I : Réseaux définis par logiciel

Dans le premier chapitre, on a donné une vision historique sur les réseaux définis par logiciel, puis on a montré son architecture générale et présenté le protocole de OpenFlow avec ses messages.

Chapitre II : Réseaux définis par la connaissance

Le deuxième chapitre de ce mémoire est dédié aux réseaux définis par la connaissance, ou on l'a défini et donné son principe avec l'apprentissage automatique et l'équilibrage de charge.

Chapitre III : Modélisation du réseau défini par les connaissances

Ce chapitre présente la conception de notre modèle en donnant la topologie en détail et les métriques nécessaires pour le développer puis on a présenté le réseau de neurones artificiels correspondant.

Chapitre IV : Expérimentations et résultats

Ce dernier chapitre contient l'implémentation du notre projet qu'on a développé en présentant la simulation du réseau puis les expérimentations faites pour enfin présenter les résultats obtenus et les discuter.

Chapitre I

Réseaux définis par logiciel

I. Chapitre I

Réseaux définis par logiciel

1. Introduction

Un réseau informatique est constitué d'un ensemble d'ordinateurs, et d'autres équipements connectés les uns aux autres afin de pouvoir communiquer entre eux. De nos jours, les réseaux sont partout. On ne peut pratiquement rien faire avec des données qui n'impliquent pas les réseaux. Comme les réseaux humains dont d'imprimantes nous faisons partie, les réseaux informatiques nous permettent de partager des informations et des ressources. Dans les entreprises, le recours aux réseaux est encore plus répandu qu'ailleurs. Les réseaux aident les particuliers et les entreprises à économiser de l'argent, mais ils contribuent également à la création de revenus [1]. Le réseau informatique aujourd'hui permis à faire un grand pas dans tous les domaines, c'est pourquoi il ne cesse de se développer. Dans ce chapitre nous présenterons le technique réseau défini par logiciel considérée un saut technologique dans les sections des réseaux informatiques, son principe aussi son fonctionnement seront détaillées.

2. Réseau défini par logiciel

2.1 Introduction

L'internet a conduit à la création d'une société numérique où tout (ou presque) sont connectés et accessibles de n'importe où. Cependant, malgré leur adoption généralisée, les réseaux traditionnels sont complexes et très difficiles à gérer. Il est à la fois difficile de configurer le réseau en fonction de règles prédéfinies et de le reconfigurer pour répondre aux défaillances, à la charge et aux modifications. Pour rendre les choses encore plus difficiles, les réseaux actuels sont également intégrés verticalement, les plans de contrôle et de données sont regroupés et intégrés dans un même matériel [2]. Le réseau défini par logiciel est un paradigme émergent qui promet de changer cet état de difficulté, en interrompant l'intégration verticale, en séparant la logique de contrôle du réseau des routeurs et commutateurs, en promouvant la centralisation (logique) du contrôle par la programmation du réseau [4]. La séparation des préoccupations introduites entre la définition des stratégies réseau, et leur implémentation dans le matériel de commutation et la transmission du trafic est la clé de la flexibilité souhaitée, en décomposant le problème de contrôle de réseau en éléments faciles à gérer. Le réseau défini par logiciel facilite la création et l'introduction de

nouvelles abstractions dans les réseaux, pour simplifier sa gestion et faciliter son évolution [3].

2.2 Définition

Selon l'ONF, le réseau défini par logiciel est un paradigme émergent du réseau qui donne la possibilité de changer les limites des infrastructures des réseaux actuels. Ce dernier rompt l'intégration verticale en séparant l'automate contrôlé du réseau (le plan de contrôle) des routeurs et des commutateurs qui transmettent le trafic (le plan de données). Avec la séparation des plans de contrôle et de données, les commutateurs de réseau deviennent de simples dispositifs de transfert et le contrôleur logique sera implémenté dans un contrôleur généralement centralisé, simplifiant l'application des règles ainsi que la (re) configuration et l'évolution du réseau. La séparation du plan de contrôle et du plan de données peut être réalisée au moyen d'une interface de programmation bien définie entre les commutateurs et le contrôleur. Le contrôleur développe une gestion directe sur l'état des nœuds du plan de données via cette interface de programmation d'application API. L'exemple le plus notable d'une telle API est OpenFlow. Un commutateur OpenFlow [OpenFlowswitch] a une ou plusieurs tables de règles de traitement des paquets [flowtable]. Chaque règle correspond à un sous-ensemble du trafic et effectue certaines actions (suppression, transfert, modification, etc.) sur le trafic selon les règles installées par une application de contrôleur, par exemple un protocole de routage ou un équilibrage de charge. Un commutateur OpenFlow peut sous l'instruction du contrôleur se comporter comme un routeur, un commutateur, un pare-feu ou un autre rôle (par exemple, équilibreur de charge, répartiteur de trafic et en général un équipement réseau). L'un des principes de fonctionnement du réseau défini par logiciel est la séparation des préoccupations introduites entre la définition des stratégies du réseau, leur mise en œuvre dans le matériel de commutation et la transmission du trafic. Cette séparation est essentielle pour la flexibilité souhaitée, pour décomposer le problème de contrôle de réseau en éléments exploitables et pour le rendre plus facile à créer et gérer [4][5].

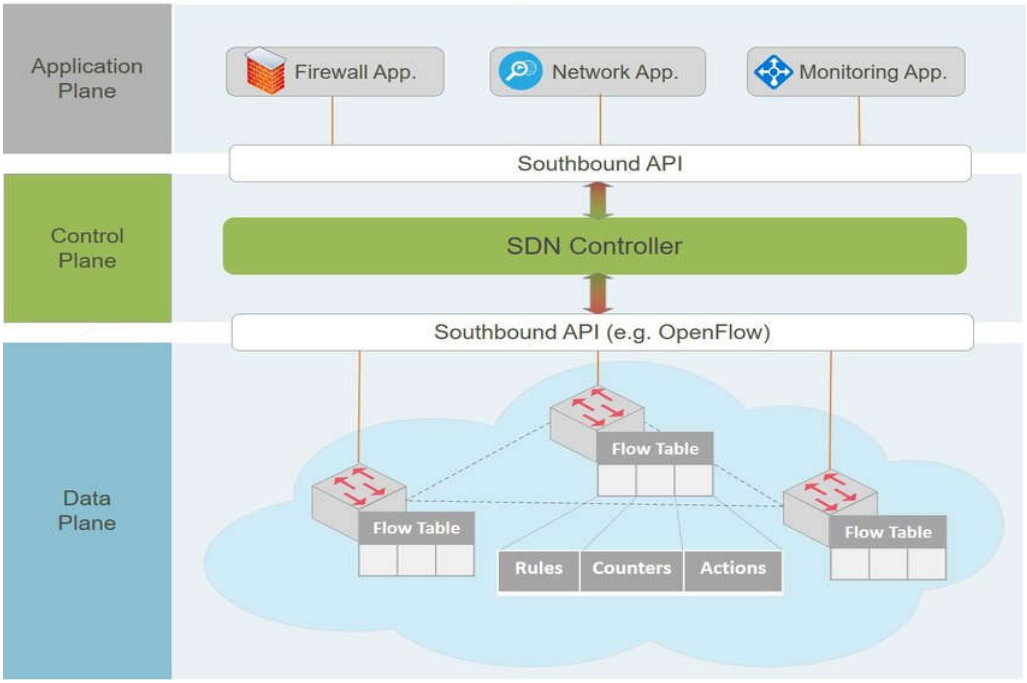


Figure I.1 Architecture réseau défini par logiciel.

Comme mentionné précédemment, l'architecture du ce réseau est conçu sur la base de l'idée de la séparation entre le plan de contrôle et les plans de données figure I.1. Où la couche [Control Plane] représente le contrôleur, La couche [Data Plane] contient les commutateurs OpenFlow et la couche application a les algorithmes de gestion.

2.3 Les réseaux actuels et les réseaux définis par logiciel

La plus grande différence entre un réseau actuel et un réseau défini par logiciel est que ce dernier est basé sur un logiciel. Les réseaux actuels reposent sur une infrastructure physique, tel que les commutateurs et les routeurs, pour établir des connexions et pour fonctionner correctement. En revanche, un réseau basé sur un logiciel permet à l'utilisateur de contrôler l'allocation de ressources à un niveau virtuel via le plan de contrôle. Plutôt que d'interagir avec une infrastructure physique, l'utilisateur interagit avec un logiciel pour mettre en service de nouveaux périphériques [6][7].

De ce point de vue, un administrateur peut déterminer les chemins d'accès au réseau et configurer activement les services réseau. Un réseau défini par logiciel a également plus de capacité à communiquer avec des périphériques à travers le réseau qu'un commutateur actuel. La différence fondamentale entre les deux peut se résumer à la séparation et la virtualisation. SDN virtualise l'ensemble de notre réseau. La virtualisation crée une version abstraite de notre réseau physique qui permet de provisionner des ressources à partir d'un emplacement

centralisé. Dans un réseau traditionnel, le plan de données indique à nos données où elles doivent transmettre, de même le plan de contrôle est situé dans un commutateur ou un routeur. L'emplacement du plan de contrôle est particulièrement gênant, car les administrateurs n'ont pas facilement accès au flux de trafic [8].

Sous un réseau défini par logiciel, le plan de contrôle devient logiciel accessible via un périphérique connecté, cela signifie qu'un administrateur peut contrôler le flux de trafic à partir d'une interface centralisée conviviale avec un contrôle accru, cela aussi donne aux utilisateurs plus de contrôle sur le fonctionnement de leur réseau. Nous pouvons également modifier les paramètres de configuration de notre réseau, gérer les configurations de cette manière est particulièrement bénéfique en ce qui concerne la segmentation du réseau car l'utilisateur peut traiter rapidement de nombreuses configurations [9], la raison pour laquelle cette technologie est devenu une alternative est qu'elle permet aux administrateurs de provisionner des ressources et de la bande passante instantanément. En revanche, un réseau actuel aurait besoin de nouveau matériel si la capacité de son réseau devait augmenter [8].

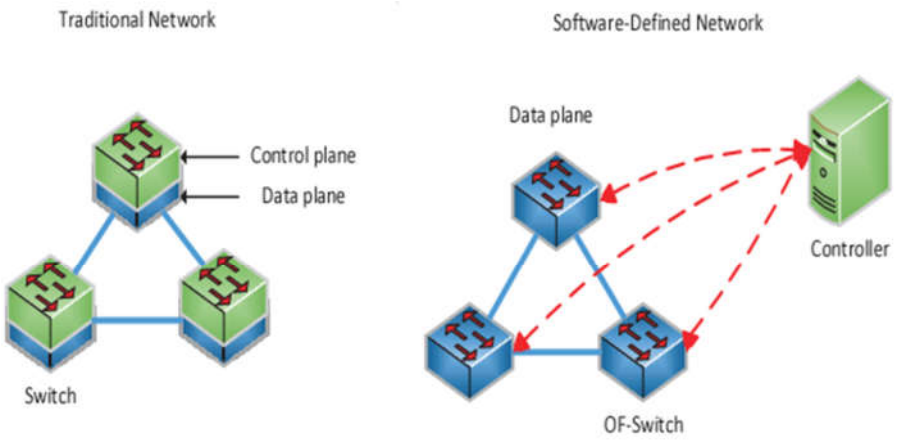


Figure I.2 Réseau défini par logiciel et le réseau traditionnel.

La figure I.2 montre la séparation des plans de données et plan de contrôle dans les réseaux définis par logiciel, et l'intégration des plans dans les réseaux traditionnels.

2.4 Historique des réseaux définis par logiciel

L'origine du réseau défini par logiciel est une façon de comprendre cette technologie. L'un des premiers exemples de ce type d'idée est les premiers systèmes de mise en réseau dans lesquels des hôtes individuels pouvaient communiquer à un ordinateur central, ces

configurations, mises au point dans la dernière partie du 20e siècle, sont assez primitives par rapport aux normes d'aujourd'hui. Le processus de découplage du plan de contrôle et du plan de données n'était pas encore terminé. Certaines sources offrent l'exemple du réseau téléphonique public commuté comme l'une des premières façons dont les ingénieurs ont commencé à séparer les plans de contrôle et de données et à évoluer vers ce qui est finalement devenu une stratégie de réseau défini par logiciel. D'autres noteront l'utilisation des réseaux peer-to-peer dans les années 1980 et l'essor de la micro-informatique comme une autre partie de la base de nos principes de mise en réseaux modernes définis par logiciel. Essentiellement, les chercheurs scientifiques ont commencé à travailler avec l'idée que tous les signaux de routage ne devaient pas résider dans son nœud et à intégrer cette philosophie dans divers systèmes [10].

L'internet Engineering Task Force (IETF) a commencé à envisager diverses façons de découpler les fonctions de contrôle et de transfert dans une proposition de norme d'interface publiée en 2004, nommée "Forwarding and Control Element Separation" (ForCES) [11].

L'utilisation de logiciels open source dans les architectures de contrôle partagé et plan de données trouve ses racines dans le projet Ethane au département d'informatique de Stanford, la conception simple des commutateurs d'Ethane a conduit à la création d'OpenFlow. Une API pour OpenFlow a été créée pour la première fois en 2008. Cette même année a vu la création de NOX, un système d'exploitation pour les réseaux [12].

Dans les universitaires, il y avait quelques réseaux de recherche et de production basés sur les commutateurs OpenFlow de NEC et Hewlett-Packard, ainsi que sur la base des boîtes blanches de Quanta Computer, à partir de 2009 [13].

Au-delà du monde universitaire, les premiers déploiements ont été effectués par Nicira en 2010 pour contrôler le OVS à partir d'Onix, co-développé avec NTT et Google. Un déploiement notable a été le déploiement B4 de Google en 2012 [14].

L'Open Networking Foundation a été fondée en 2011 pour promouvoir SDN et OpenFlow. En 2014, Interop and Tech Field Day, le réseau défini par logiciel a été démontrée par Avaya en utilisant le pontage le plus court (IEEE 802.1aq) et OpenStack comme campus automatisé [15].

3. Application des réseaux définis par logiciel

3.1 Applications dans la réalité

Bien que le réseau défini par logiciel et OpenFlow aient été au départ des expériences académiques, ils ont gagné en popularité ces dernières années dans le secteur réseau. La plupart des fournisseurs de commutateurs commerciaux incluent désormais la prise en charge de l'API OpenFlow dans leurs équipements. La dynamique de cette technologie a été suffisamment forte pour que Google, Facebook, Yahoo, Microsoft, Verizon et la Deutsche Telekom fun Open Networking Foundation aient pour objectif principal la promotion et l'adoption par le biais de normes développement. Les idées du réseau défini par logiciel ont mûri et sont passées d'un exercice théorique à un succès commercial. Google, par exemple a déployé un réseau défini par logiciel pour interconnecter ses centres de données à travers le monde. Ce réseau de production est en déploiement depuis l'année 2015, ce qui a permis à l'entreprise d'améliorer son efficacité opérationnelle et de réduire considérablement ses coûts. La plate-forme de virtualisation de réseau de VMware, NSX, en est un autre exemple. NSX est une solution commerciale offrant un réseau logiciel entièrement fonctionnel, mis à disposition indépendamment des périphériques réseaux sous-jacents, entièrement basé sur les principes du réseau défini par logiciel. Les plus grandes sociétés informatiques du monde ont récemment rejoint des consortiums SDN tels que l'ONF et l'initiative OpenDaylight, ce qui témoigne de l'importance du ce réseau dans l'industrie [3].

3.2 Domaine de l'utilisation

3.2.1 Réseau mobile défini par logiciel SD-MN

L'SD-MN est une approche de la conception de réseaux mobiles où toutes les fonctionnalités spécifiques au protocole sont mises en œuvre dans le logiciel, maximisant l'utilisation de matériel et de logiciels génériques et de base dans le réseau central et réseau d'accès radio. Il est proposé comme une extension du paradigme réseau défini par logiciel pour incorporer des fonctionnalités spécifiques au réseau mobile. Une séparation du plan utilisateur de contrôle a été introduite dans les architectures du réseau mobile de base avec le protocole PFCP [16].

3.2.2 Réseau étendue défini par logiciel SD-WAN

L'SD-WAN est un réseau étendu (WAN) géré selon les principes du réseau défini par logiciel. Le principal moteur du SD-WAN est de réduire les coûts du WAN en utilisant des chemins plus abordables et disponibles dans le centre de données DC. Le contrôle et la

gestion sont administrés séparément du matériel avec des contrôleurs centraux permettant une configuration et une administration plus faciles [17].

3.2.3 Réseau local défini par logiciel SD-LAN

L'SD-LAN est un réseau local (LAN) construit autour des principes de réseau défini par logiciel, bien qu'il existe des différences chemins dans la topologie, la sécurité du réseau, la visibilité et le contrôle des applications, la gestion et la qualité de service. SD-LAN dissocie la gestion du contrôle et les plans de données pour permettre une architecture basée sur des règles pour les réseaux LAN câblés et sans fil. Les SD-LAN se caractérisent par leur utilisation d'un système de gestion cloud et d'une connectivité sans fil sans la présence d'un contrôleur physique [18].

3.2.4 La sécurité dans les réseaux définis par logiciel

Plusieurs travaux de recherche sur le réseau défini par logiciel ont déjà étudié les applications de sécurité basées sur le contrôleur SDN, avec différents objectifs, tel que la détection et l'atténuation du déni de service distribué (DDoS), ainsi que le botnet et la propagation de vers [19].

4. Architecture des réseaux définis par logiciel

Une architecture du réseau défini par logiciel est constituée des composants illustrés dans la figure suivante :

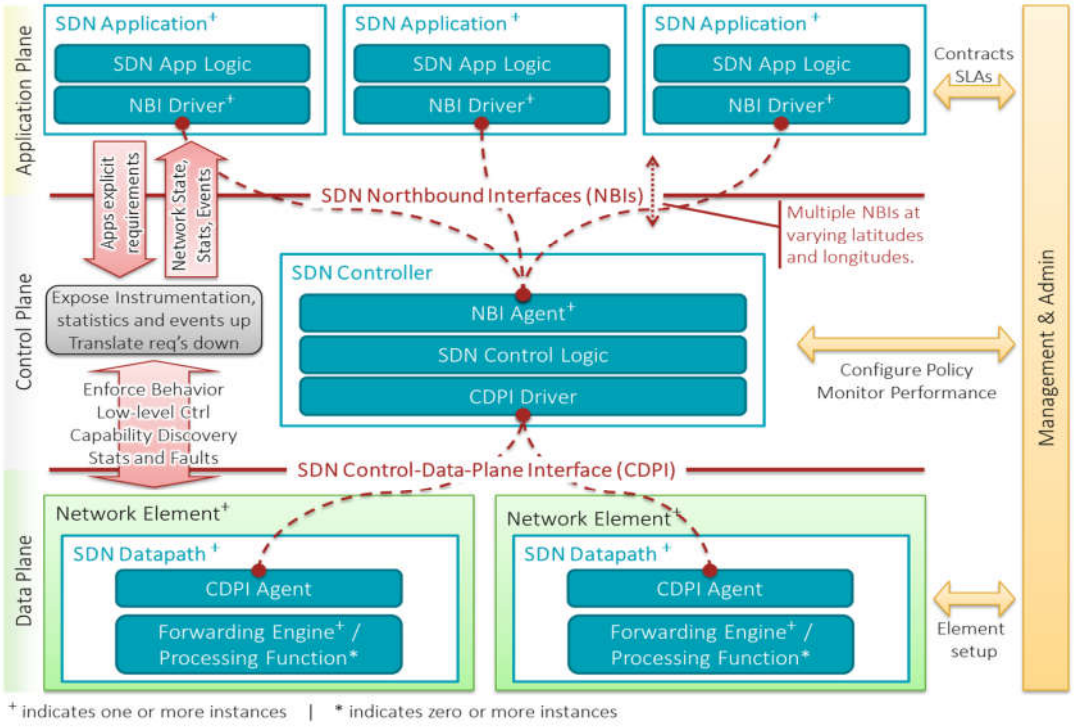


Figure I.3 Architecture des plans pour le réseau défini par logiciel.

La figure I.3 présente les composants principaux de l'architecture des plans avec les liaisons entre eux.

Composants	Description
Plan d'applications	Les applications des réseaux définis par logiciel sont les programmes qui communiquent au contrôleur explicitement, directement et numériquement, leurs besoins en ressources du réseau via une [northbound interface (NBI)]. Une application SDN comprend une logique et un ou plusieurs pilotes [NBI Drivers]. Les applications SDN peuvent elles-mêmes exposer une autre couche d'abstraction de contrôle du réseau et offrant de la sorte des NBI de plus haut niveau [20].
Plan de contrôle	Le contrôleur de réseau défini par logiciel est une entité logique centralisée en charge de traduire les besoins d'une application en descendant vers le plan de données, et fournit aux applications une vue abstraite du réseau (des statistiques et des événements). Un

	contrôleur comporte un ou plusieurs NBI Agents, le contrôle logique et le contrôle vers les plans de données [20].
Plan de données (transfert)	Le plan de données ou transfert de réseau défini par logiciel est un périphérique réseau logique qui facilite le contrôle sur ses capacités en transfert de trafic. Il comprend un agent CDPI et un ensemble d'actions [traffic forwarding engines] et plusieurs fonctions de [traffic processing]. Il s'agit des chemins de données qui peuvent parcourir sur toute l'infrastructure à travers plusieurs périphériques [21].
Contrôle vers le plan de données	Le réseau défini par logiciel CDPI est l'interface définie entre le contrôleur et le plan de données qui fournit un contrôle programmatique de toutes les opérations de transfert, des capacités d'annonce, du reportant d'événement et de la notification d'évènements. Un des avantages d'un réseau défini par logiciel est que le CDPI soit implémenté d'une manière ouverte indépendante des constructeurs et interopérable [20].
Northbound Interfaces (NBI)	Les NBIs sont des interfaces entre les applications et les contrôleurs, ils fournissent une vue abstraite du réseau et permettent l'expression directe des besoins formulés en ressources réseau. Cela comprend n'importe quel niveau d'abstraction d'un point de vue vertical et à travers différents ensembles de fonctionnalités d'un point de vue horizontal [22]. Une interface northbound est un élément du réseau qui communique avec un élément de plus haut niveau contrairement aux interfaces southbound qui discute avec des composants de plus bas niveau tels que des périphériques du réseau. Concrètement les APIs northbound activent les applications des systèmes afin de programmer un service réseau et le demander [24].

Southbound Interfaces (SBI)	Les interfaces Southbound sont implémentées avec un Service Abstraction Layer (SAL) qui discute avec les périphériques réseau en utilisant par exemple la ligne de commande [Command Line Interface (CLI)] pour commander le réseau [23].

Tableau I.1 Composants de réseau défini par logiciel.

Le tableau I.1 définie en détail les composants de l'architecture des réseaux définis par logiciel montrés dans la figure précédente.

5. Types de contrôleurs réseau défini par logiciel

Il existe de nombreux types de contrôleurs, nous avons présenté les plus populaires dans le tableau suivant [25] :

Contrôleurs	**POX**	**Ryu**	**Trema**	**Floodlight**	**OpenDaylight**
Langage	Python	Python	C Ruby	Java	Java
Version OpenFlow	v1.0	v1.0 jusqu'à v1.3	v1.0 v1.2	v1.0 jusqu'à v1.4	v1.0 jusqu'à v1.5
Open source	Oui	Oui	Oui	Oui	Oui
GUI	Oui	Oui	Non	Oui « Web Gui »	Oui
REST API	Non	Oui	Non	Oui	Oui
Plateforme	Linux, Mac, Windows	Lunix	Linux	Linux	Linux, Mac, Windows

Tableau I.2 Les différents types des contrôleurs SDN.

6. Protocole OpenFlow dans l'approche SDN

Les commutateurs Ethernet et les routeurs les plus modernes contiennent des tables de flux qui sont utilisées pour effectuer des fonctions de transfert selon les couches 2,3 et 4 de modèle TCP/IP. Bien que chaque fournisseur ait des tables de flux différentes, il existe un ensemble commun de fonctions pour une large gamme de commutateurs et de routeurs. Cet ensemble commun de fonctions est mis à profit par OpenFlow, protocole entre un contrôleur central OpenFlow et un commutateur OpenFlow, il peut être utilisé pour programmer la logique de transfert ou d'acheminement du commutateur. La figure I.4 décrit les fonctions réalisées par les équipements de réseau du plan de données aussi appelés commutateur OpenFlow. Les principales fonctions des commutateurs sont les suivantes [26] :

Fonction de support du contrôle [Control support function] : Interagit avec la couche contrôle afin de supporter la programmabilité via les interfaces ressource contrôle. Le commutateur communique avec le contrôleur et le contrôleur gère le commutateur avec le protocole OpenFlow. OpenFlow peut être utilisé aussi bien pour le contrôle que pour de la gestion.

Fonction d'acheminement des données [Data forwarding function] : Accepte les flux de données entrants provenant d'autres équipements de réseau et des systèmes de terminaison et les relaie sur un chemin de commutation qui a été calculé et établi à partir des règles définies par les applications, passées au contrôleur et redescendues au commutateur. Ces règles d'acheminement des données sont présentes dans les tables d'acheminement [forwarding tables]. Ces règles indiquent pour des catégories de paquet données quel doit être le prochain saut sur la route. Le commutateur peut par ailleurs modifier l'en-tête du paquet avant son acheminement, ou rejeter le paquet.

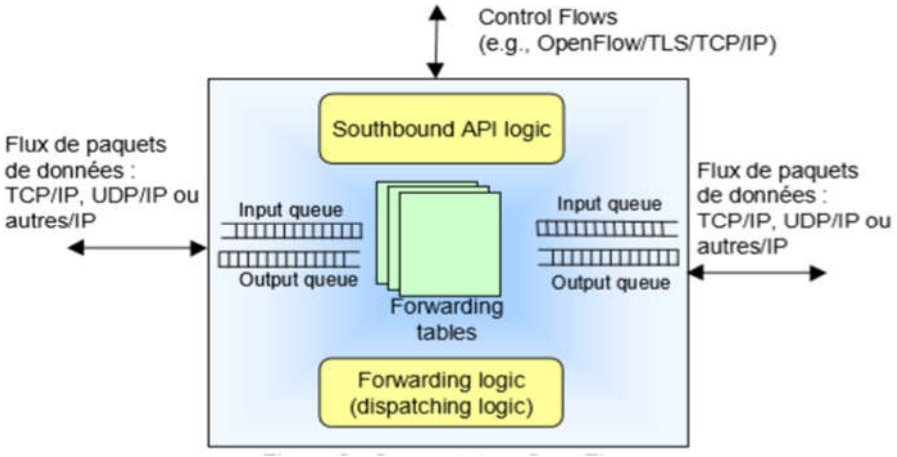


Figure I.4 Fonctionnement de Commutateur OVS.

D'après la figure I.4, les paquets arrivants sont placés dans une file d'attente en entrée, attendant leur traitement par le commutateur. Les paquets acheminés sont placés dans une file d'attente en sortie, avant d'être transmis. Le commutateur de la figure I.4 dispose de trois ports d'entrée/sortie (les trois flèches noires) : Un port fournissant la communication de contrôle avec un contrôleur réseau défini par logiciel, et deux autres ports pour les entrées et sorties de paquets de données. Le commutateur peut disposer de plusieurs ports pour communiquer avec plusieurs contrôleurs, et de plus de 2 ports pour les paquets de données entrant et sortant du commutateur.

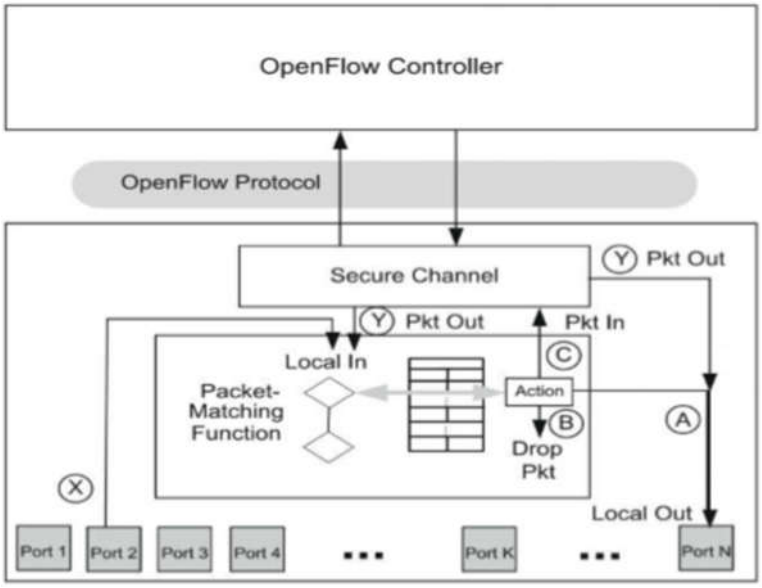


Figure I.5 Architecture Commutateur OVS.

La figure I.5 décrit les fonctions du commutateur OpenFlow V1.0 et sa relation au contrôleur. Comme attendu dans un commutateur de paquet, la fonction de base est de recevoir les paquets qui arrivent sur un port (Chemin X sur port 2) et les relayer via un autre port (Port N) en réalisant toute modification nécessaire sur les paquets au chemin. La fonction de correspondance de paquet (packet-matching function) est très importante dans le commutateur OpenFlow. La table adjacente est une table de flux la flèche grise sur le chemin commence dans la logique de décision, montre une correspondance avec une entrée particulière dans la table de flux, et dirige le paquet pour lequel une correspondance a été trouvée à une action sur la droite. Cette action a trois options de base pour le traitement du paquet arrivé,

- (A) Relayer le paquet sur un port de sortie, avec auparavant la possibilité de modifier certains champs d'en-tête du paquet.

- (B) Supprimer le paquet
- (C) Passer le paquet au contrôleur.

Le paquet est encapsulé dans un message OpenFlow PACKET_IN. Ces trois chemins fondamentaux pour le paquet sont illustrés sur la figure I.5. Dans le cas du chemin (C), le paquet est passé au contrôleur via un canal sécurisé montré à la figure I.5. Si le contrôleur a soit un message de contrôle par exemple mettre hors service un port du commutateur, ou un paquet de données à fournir au commutateur, le contrôleur utilise ce même canal sécurisé dans le sens inverse. Lorsque le contrôleur a un paquet de données à relayer via le commutateur, il utilise le message OpenFlow PACKET-OUT. Un paquet de données provenant du contrôleur peut suivre deux chemins, dénotés Y via la logique OpenFlow. Dans le cas du chemin Y de droite, le contrôleur spécifie directement le port de sortie et le paquet qui est passé à ce port N. Dans le cas du chemin Y de gauche, le contrôleur indique qu'il souhaite déléguer la décision de transfert du paquet à la logique de correspondance de paquet. Le contrôleur stipule alors le port de sortie TABLE pour le traitement du paquet par la fonction de correspondance de paquet et l'identification par ce traitement du port de sortie. Un commutateur OpenFlow peut être uniquement OpenFlow ou hybride. Dans ce dernier cas, le commutateur peut aussi transférer les paquets selon son mode traditionnel comme commutateur Ethernet ou routeur IP. Le cas hybride peut être vu comme un commutateur OpenFlow qui réside à côté d'un commutateur traditionnel et indépendant. Ce type de commutateur hybride requiert un mécanisme de classification qui soit dirigé les paquets au contrôleur OpenFlow, soit les traites de manière traditionnelle.

La spécification OpenFlow définit le concept de port OpenFlow. Il s'agit d 'un port physique dans OpenFlow 1.0. Pendant de nombreuses années, les commutateurs ont supporté de nombreuses files d'attente par port physique. Ces files d'attente sont servies par des algorithmes d'ordonnancement qui contribuent à fournir différents niveaux de QoS pour différents types de paquet. OpenFlow prend en compte ce concept et permet au flux d'être envoyé sur une file d'attente déjà définie sur un port de sortie. La sortie du paquet sur un port N peut inclure le numéro de file d'attente du port N dans laquelle placer le paquet.

6.1 Table de flux

Chaque table de flux du commutateur contient un ensemble d'entrées de flux qui présentent les règles d'acheminement des paquets. Une entrée de flux est composée de figure I.6) :

- **Match fields** : champs de correspondance qui définissent le modèle du flux de paquets à travers l'instanciation des champs d'en-tête allant de la couche Ethernet à la couche.
- **TransportCounters** : des compteurs sur les paquets
- **Actions** : Actions à appliquer aux paquets qui correspondent à l'entrée de flux.

Figure I.6 Structure d'une entrée de flux.

Chaque entrée de flux est associée à zéro ou plusieurs actions qui dictent la façon dont le commutateur gère les paquets correspondants. Si aucune action n'est présente, le paquet est supprimé. Les listes d'actions présentes dans les entrées de flux doivent être traitées dans l'ordre spécifié. Cependant, il n'y a pas d'ordre de sortie du paquet garanti dans un port. Par exemple, une liste d'actions peut résulter en deux paquets envoyés à deux différents VLANs sur un seul port. Ces deux paquets peuvent être arbitrairement réordonnés, mais les corps de paquets doivent correspondre à ceux générés par une exécution séquentielle des actions. Un commutateur peut rejeter une entrée de flux si elle ne peut pas traiter la liste d'actions dans l'ordre spécifié, auquel cas il doit immédiatement retourner un message d'erreur [flux non pris en charge] au contrôleur. Un commutateur n'a pas à supporter tous les types d'action seulement celles marquées [Actions obligatoires]. Lors de la connexion au contrôleur, un commutateur indique lesquelles des actions optionnelles il supporte.

6.1.1 Champs de correspondance

Les Match Fields pouvant être utilisés dans OpenFlow v1.0, 1.1, 1.2, 1.3 sont montres dans le tableau suivant :

Champs de port	La taille	Descriptions
Ingress Port	/	Port d'entrée sur lequel est reçu le paquet.
Ethernet source address	48	Adresse Ethernet source. Il est à noter que seule la couche Ethernet est considérée au niveau liaison de données avec OpenFlow.
Ethernet destination		Adresse Ethernet destination. Il est à noter que seule

address	48	la couche Ethernet est considérée au niveau liaison de données avec OpenFlow.
Ethernet frame type	16	Ethernet II, LLC ou NSAP
VLAN id	12	Identificateur de VLAN dans l'en-tête VLAN si présent
VLAN priority	3	Priorité VLAN dans l'en-tête VLAN si présent.
IPv4 source address	32	Adresse IPv4 source du paquet
IPv4 destination address	32	Adresse IPv4 destination du paquet.
IP protocol	8	Protocole contenu dans IP (OSPF, TCP, UDP, etc.).
Transport source port /ICMP Type	/	Port source (couche transport).
Transport destination port /ICMP Type	/	Port destination.

Tableau I.3 Champs Match Fields.

6.1.2 Compteurs

Les compteurs peuvent être résumés pour chaque table de flux, entrée de flux, port, file d'attente. Le tableau suivant décrit les compteurs :

Compteurs	Champs et détails
Flow Table	**Reference Count** (active entries) : Nombre d'entrées actives dans la table. **Packet Look ups** : Nombre de paquets soumis à la table. **Packet Matches** : Nombre de paquets pour lesquels une des entrées de la table a pu s'appliquer.
	Received Packets : Nombre de paquets reçus par l'entrée pour

Flow entry	lesquels la recherche de correspondance a réussi. **Received Bytes** : Nombre d'octets de paquets reçus par l'entrée pour lesquels la recherche de correspondance a réussi. **Duration (seconds)** : Durée en secondes pendant laquelle l'entrée a été active. **Duration (nano secondes)** : Durée en nano secondes pendant laquelle l'entrée a été active au-delà de la durée en secondes.
Port /queue	**Received Packets** : Nombre de paquets reçus sur ce port. **Transmitted Packets** : Nombre de paquets transmis par ce port. **Received Bytes** : Nombre d'octets reçus par ce port. **Transmitted Bytes** : Nombre d'octets transmis par ce port. **Receive Drops** : Nombre de paquets reçus et rejetés par ce port. **Transmit Drops** : Nombre de paquet rejetés en transmission. **Receive Errors** : Nombre de paquets reçus en erreur. **Transmit Errors** : Nombre de paquet transmis en erreur. **Receive Frame Alignment Errors** : Nombre de paquet reçus avec erreur d'alignement. **Receive Overrun Errors** : Nombre de paquets reçus et rejetés faute de mémoire dans les files d'attente en entrée du port. **Receive CRC Errors** : Nombre de paquets reçus avec un CRC erroné. **Collisions** : Nombre de paquets rejetés suite à une collision Par Queue (file d'attente). **Transmit Packets** : Nombre de paquets transmis sur cette file d'attente. **Transmit Bytes** : Nombre d'octets transmis sur cette file d'attente. **Transmit Overrun Errors** : Nombre de paquet transmis sur cette file d'attente et rejetés faute de mémoire sur la file. Lorsqu'un compteur arrive à sa valeur maximum, il repasse à 0 sans autre indication. Si un compteur n'est pas disponible, sa valeur doit être positionnée à -1.

Tableau I.4 Compteurs.

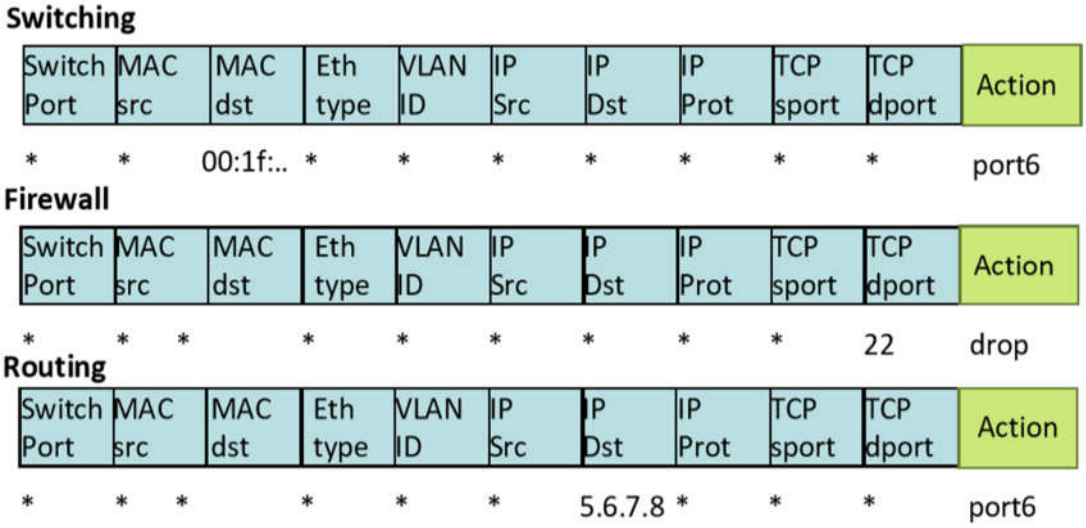


Figure I.7 Exemples d'entrées de table de flux.

La figure I.7 présente quelques exemples d'entrée de table de flux. La première entrée concerne la commutation Ethernet, la seconde, le filtrage de trafic et la troisième, le routage du trafic

6.1.3 Action

Il existe des actions obligatoires que doit supporter tout commutateur OpenFlow et des actions OpenFlow que peut supporter un commutateur OpenFlow.

Actions	Etat	Descriptions
Forward	Obligatoire	Les commutateurs OpenFlow doivent supporter l'acheminement de paquet aux ports physiques ainsi qu'aux ports virtuels
Forward	Optionnel	Le commutateur peut en option supporter le port virtuel normal qui revient à traiter le paquet selon la méthode de commutation traditionnelle
Drop	Obligatoire	Une entrée de flux sans action indique que tous les paquets correspondants doivent être supprimés.
		Cette action relaie un paquet via une file d'attente

Queue	Optionnel	associée à un port. Le comportement d'acheminement est dicté par la configuration de la file d'attente et permet de fournir un support pour une qualité de service
Modify Field	Optionnel	L'action indiquée apporte de la valeur à une implantation OpenFlow, elles permettent de modifier des champs d'en-tête du paquet avant son acheminement sur un port de sortie.

Tableau I.5 champs d'actions.

7. Protocole OFDP et les messages Packet in/out

Afin de maintenir le service, le contrôleur doit continuer à mettre à jour les informations sur le réseau et sa topologie [23]. L'une des tâches critiques dont le contrôleur est la vue presque en temps réel de la topologie du réseau connue sous le nom de découverte de topologie [Topology Discovery]. Par rapport aux protocoles hérités qui dépendent de l'état de la liaison (OSPF), la découverte de la topologie dans les réseaux définis par logiciels est plus critique. Tous les contrôleurs de réseau utilisent le protocole (OFDP) pour découvrir la topologie du réseau dans cette technologie en utilisant des trames LLDP (Link Layer Discovery Protocol) sous forme de messages [27].

Dans le protocole OFDP actuel, après avoir établi la connexion entre les commutateurs et le contrôleur, le contrôleur envoie un message OFTP_PACKET_OUT à chaque port de chaque commutateur, comme illustré sur la figure I.8 [27].

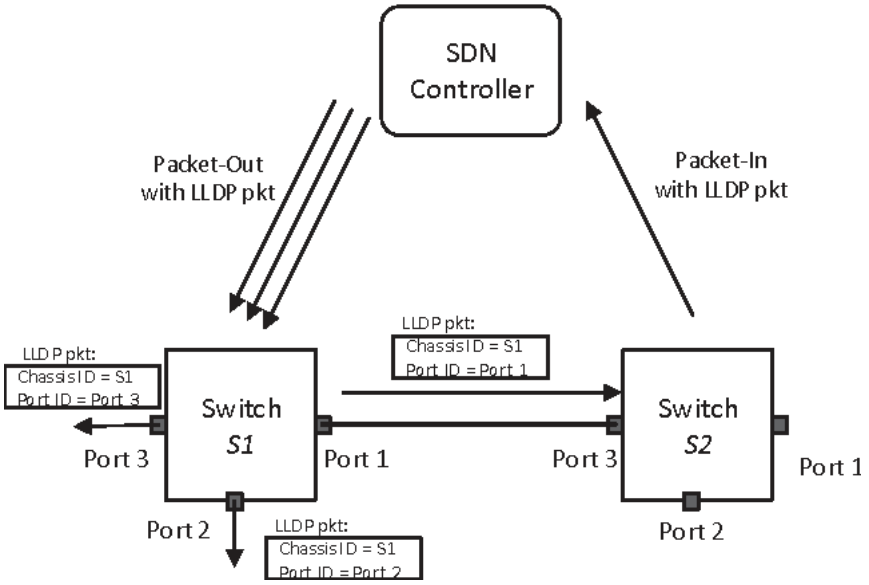


Figure I.8 Scenario basique d'OFDP.

Le contrôleur SDN envoie trois paquets LLDP à chaque port du commutateur S1 à l'aide du message OFTP_PACKET_OUT. Le message qui est inclus dans le paquet LLDP contient des instructions pour diriger le paquet vers le port associé. Par exemple, le paquet LLDP qui a PortID 1 est envoyé au port ID 1, le paquet qui a PortID 2 est envoyé au port 2, et ainsi de suite. De plus, tous les commutateurs réseau contiennent une règle pour renvoyer chaque paquet LLDP qui ne provient pas du contrôleur vers le contrôleur. Ce processus se fait à l'aide du message OFTP_PACKET_IN. Selon les règles de transfert intégrées dans les commutateurs, le commutateur S2 envoie le paquet LLDP avec le message OFTP_PACKET_IN au contrôleur. Ce paquet comprend l'ID de commutateur et l'ID de port entrant duquel le paquet est reçu. En raison de ces informations, le contrôleur peut déterminer qu'un commutateur de liaison de canal opérationnel (S1, Port1) et un commutateur (S2, Port3), et les informations sur la liaison sont stockés dans des bases de données contenant des informations sur la topologie. Ce processus est effectué de manière itérative pour chaque commutateur opérationnel du réseau. Toujours garder à jour les informations sur la topologie, le contrôleur effectue périodiquement la découverte des liens à des intervalles de temps fixes (Presque 10 secondes pour tous les contrôleurs). En raison du fait que la découverte de la topologie du réseau défini par logiciel est un service se produit en continu en arrière-plan, par conséquent, il est essentiel de connaître la charge de trafic que la découverte de topologie place sur le contrôleur. Pour déterminer la charge du contrôleur, il est essentiel de connaître la quantité de messages OFTP_PACKET_OUT qu'il envoie et les messages OFTP_PACKET_IN requis pour traiter [27].

8. Capacités générales des réseaux définis par logiciel

Le réseau défini par logiciel propose un réseau centralisé et programmable qui peut provisionner dynamiquement des ressources réseau afin de répondre aux besoins changeants des entreprises. Il offre également les avantages techniques et commerciaux suivants [26] :

Directement programmable : la stratégie de réseau défini par logiciel est directement programmable car les fonctions de contrôle sont dissociées des fonctions de transfert, ce qui permet au réseau d'être configuré par programme et par des outils d'automatisation propriétaires ou open source, notamment OpenStack, Puppet, Salt et Ansible.

Gestion centralisée : l'intelligence du réseau est logiquement centralisée dans le logiciel du contrôleur qui maintient une vue globale du réseau, qui apparaît aux applications et aux moteurs de stratégie réseau défini par logiciel comme un commutateur logique unique.

Investissements réduits : Cette technologie limite potentiellement la nécessité d'acheter du matériel de mise en réseau spécifique basé sur ASIC et prend en charge les modèles de paiement à la croissance avec ses capacités de mise à l'échelle. La plupart des commutateurs du marché prennent en charge les capacités de ce réseau et les logiciels comme OpenFlow, que ce soit dans un centre de données ou un autre réseau.

Agilité et flexibilité : Le réseau défini par logiciel peut aider les organisations à déployer rapidement de nouvelles applications, services et infrastructures pour répondre rapidement à des buts et objectifs commerciaux changeants, car chaque fois que quelque chose de nouveau est créé, une simple mise à jour le déploie à l'échelle du réseau.

9. Conclusion

Le réseau défini par logiciel est une technologie de réseau qui répond aux exigences croissantes du réseau actuel. Cela peut être atteint sans qu'il soit nécessaire d'ajouter de nouveaux périphériques ou une configuration manuelle pour tous les périphériques. La seule exigence est d'avoir un commutateur ou un routeur avec une fonctionnalité activée par le protocole OpenFlow. Depuis quelques années plusieurs tentatives d’intégration de l’intelligence artificielle dans cette technologie ont vu le jour. Dans le chapitre suivant sera consacré à l’introduction de technologie du l’apprentissage automatique dans les réseaux définis par logiciels.

Chapitre II

Réseaux définis par la connaissance

II. Chapitre II

Réseaux définis par la connaissance

1. Introduction

L'association du réseau défini par logiciel avec les techniques de l'intelligence artificielle a donné naissance à un nouveau domaine qui est le réseau défini par la connaissance. Ce chapitre présente les réseaux de neurones artificiels en tant que technique d'intelligence artificiels ainsi que le problème d'équilibrage de charge [Load balancing]. Plusieurs travaux de recherche serrant passée en revue.

2. Réseau défini par la connaissance

2.1 Définition

Le réseau défini par les connaissances [Knowledge defined Networking (KDN)] est à l'origine défini au début d'internet, ce qui suggère un concept de plan de connaissances qui adopte l'intelligence artificielle et le système cognitif pour construire un modèle automatique pour le réseau. Les systèmes intrinsèquement distribués où chaque nœud ne dispose que d'une vue locale et une puissance de calcul très limitée, ne permettre pas l'introduction de l'application dans ces derniers. En améliorant ces deux derniers, le réseau défini par la connaissance est redéfini comme la combinaison du plan de connaissances, de l'apprentissage automatique, du réseau défini par logiciel et de l'analyse de réseau [28].

2.2 Architecture

Le concept de réseau défini par la connaissance introduit dans ce travail reprend le concept du plan de connaissance [KnowledgePlane (KP)] tel que défini par Alex et Christian [36]. Dans le cadre des architectures réseau défini par logiciel, l'ajout d'un KP aux trois plans traditionnels (Application, Contrôle et transfert de données) du paradigme SDN aboutit à ce que nous appelons un réseau défini par la connaissance.

La figure II.1 montre une vue d'ensemble du paradigme KDN et de ses plans fonctionnels. Le plan de données est responsable du stockage, du transfert et du traitement des paquets de données. Ils opèrent sans connaître le reste du réseau et s'appuient sur les autres plans pour remplir leurs tables de transfert et mettre à jour leur configuration. Le plan de contrôle change d'état opérationnel afin de mettre à jour les règles de correspondance et de traitement des plans de données. Dans un réseau défini par logiciel, ce rôle est attribué au contrôleur qui programme les éléments de transmission du plan de données via une interface vers les

serveurs. Alors que le plan de données fonctionne à des échelles de temps de paquet, le plan de contrôle est plus lent et fonctionne généralement à des échelles de temps de flux. Le plan de gestion garantit le fonctionnement et les performances corrects du réseau à long terme. Il définit la topologie du réseau et gère la configuration des périphériques réseau. Dans cette technologie, cela est généralement géré par le contrôleur également. Le plan de gestion est également responsable de la surveillance du réseau pour fournir des analyses de réseau critiques. À cette fin, il collecte des informations de télémétrie à partir du plan de contrôle et de données tout en conservant un historique de l'état et des événements du réseau. Le plan de gestion est orthogonal aux plans de contrôle et de données, et fonctionne généralement à des échelles de temps plus grandes. Le plan de connaissance est comme suit :

Le cœur du plan de la connaissance est sa capacité à intégrer des modèles comportementaux et des processus de raisonnement orientés vers la prise de décision dans un réseau défini par logiciel. Dans le paradigme KDN, le KP tire parti des plans de contrôle et de gestion pour obtenir une vue et un contrôle riches sur le réseau. Il est responsable de l'apprentissage du comportement du réseau et dans certains cas, exploite automatiquement le réseau en conséquence. Fondamentalement, le KP traite les analyses de réseau collectées par le plan de gestion, soit des données prétraitées soit des données brutes, les transforme en connaissances via l'apprentissage automatique et utilise ces connaissances pour prendre des décisions automatiquement, bien que l'analyse des informations et leur apprentissage soient généralement un processus hors ligne lent, l'utilisation automatique de ces connaissances peut se faire à des échelles de temps proches de celles des plans de contrôle [28].

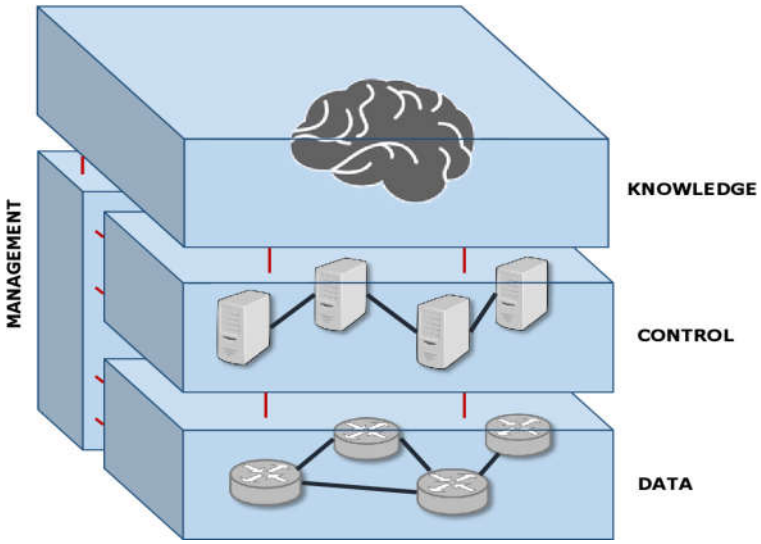


Figure II.1 Architecture des réseaux définis par la connaissance.

3. Apprentissage Automatique

L'apprentissage automatique [Machine Learning (ML)] est un sous-domaine de l'intelligence artificielle (IA) [Artificial intelligence (AI)]. En général, l'objectif de l'apprentissage automatique est de comprendre la structure des données et de les intégrer dans des modèles qui peuvent être compris et utilisés par d'autres structure de données non définies [28].

Bien que l'apprentissage automatique soit un domaine de l'informatique, il diffère des approches informatiques traditionnelles. En effet dans cette dernière, les algorithmes sont des ensembles d'instructions explicitement programmées utilisées par les ordinateurs pour calculer ou résoudre des problèmes [29]. Les algorithmes d'apprentissage automatique permettent aux ordinateurs de s'entraîner sur les entrées de données pour produire des valeurs qui se situent dans une plage spécifique. Pour cette raison, l'apprentissage automatique facilite l'utilisation des ordinateurs dans la construction de modèles à partir de données d'échantillonnage afin d'automatiser les processus de prise de décision en fonction des données saisies. Bien que le domaine de réseaux informatique utilise cette technologie pour remplir les tables de routages ou utiliser l'équilibrage de données permet d'une gestion dynamique et automatique sans intervention externe d'un administrateur réseau [30].

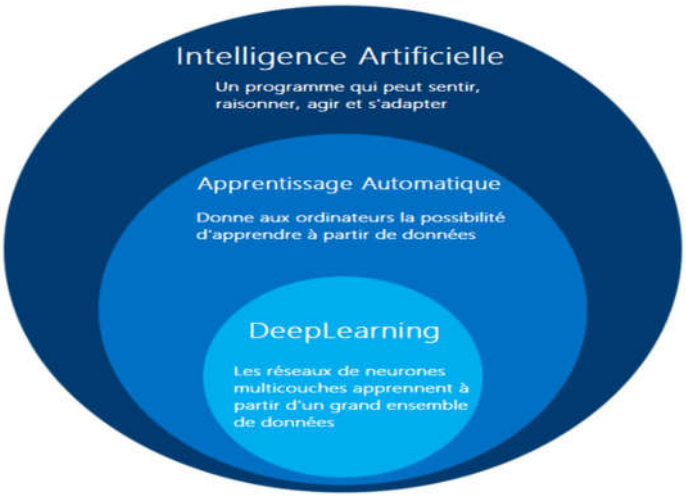


Figure II.2 L'intelligence artificielle et l'apprentissage automatique.

L'intelligence artificielle est un domaine vaste qui comprend plusieurs sous domaine, tel que l'apprentissage automatique que lui-même a plusieurs approches, la figure II.2 montrant le positionnement des notions d'intelligence artificielle, l'apprentissage automatique et Deep learning imbriquées les unes aux autres [28].

4. Méthodes d'apprentissage automatique

Dans l'apprentissage automatique, les tâches sont généralement classées en grandes catégories, ces catégories sont basées sur la façon dont l'apprentissage est reçu ou comment le feedback sur l'apprentissage est donné au système développé [29].

Deux des méthodes d'apprentissage automatique les plus largement adoptées sont l'apprentissage supervisé qui forme des algorithmes basés sur des données d'entrée et de sortie étiquetées et l'apprentissage non supervisé qui ne fournit pas à l'algorithme des données étiquetées pour lui permettre de trouver une structure et de découvrir une logique dans les données d'entrées. Explorons donc ces méthodes plus en détail [30].

4.1 Apprentissage supervisée

Dans l'apprentissage supervisé, l'ordinateur est fourni avec des exemples d'entrées qui sont étiquetés avec les sorties souhaitées. Le but de cette méthode est que l'algorithme puisse « apprendre » en comparant sa sortie réelle avec les sorties « enseignées » pour trouver des erreurs et modifier le modèle en conséquence. L'apprentissage supervisé utilise donc des modèles pour prédire les valeurs d'étiquettes sur des données non étiquetées supplémentaires. Un cas d'utilisation de l'apprentissage supervisé consiste à utiliser des données historiques pour prédire des événements futurs statistiquement probables. Il peut utiliser les informations historiques sur les marchés boursiers pour anticiper les fluctuations à venir ou être utilisé pour filtrer les courriers indésirables [30].

4.2 Apprentissage non supervisé

Dans l'apprentissage non supervisé, les données sont non étiquetées, de sorte que l'algorithme d'apprentissage trouve tout seul des points communs parmi ses données d'entrée. Les données non étiquetées étant plus abondantes que les données étiquetées, les méthodes d'apprentissage automatique qui facilitent l'apprentissage non supervisé sont particulièrement utiles. L'objectif de l'apprentissage non supervisé peut être aussi simple que de découvrir des modèles cachés dans un ensemble de données, mais il peut aussi avoir un objectif d'apprentissage des caractéristiques, qui permet à la machine intelligente de découvrir automatiquement les représentations nécessaires pour classer les données brutes. L'apprentissage non supervisé est couramment utilisé pour les données transactionnelles. Vous pouvez avoir un grand ensemble de données sur les clients et leurs achats, mais en tant qu'être humain, vous ne serez probablement pas en mesure de comprendre quels attributs similaires peuvent être tirés des profils de clients et de leurs types d'achats. Avec ces données

introduites dans un algorithme d'apprentissage non supervisé, on peut déterminer que les femmes d'une certaine tranche d'âge qui achètent des savons non parfumés sont susceptibles d'être enceintes, et donc une campagne de marketing liée à la grossesse et aux produits pour bébés pour augmenter leur nombre d'achats [30].

4.3 Comparaison entre les méthodes d'apprentissages

Le tableau suivant décrit la différence entre l'apprentissage supervisé et non supervisé avec la capacité de chacune [30] :

Supervisé	**Non supervisé**
Les données d'entraînement contiennent à la fois les attributs et les résultats désirés.	Le modèle n'est pas alimenté avec des résultats connus durant la phase d'entrainement.
Pour certains exemples les résultats sont connus et sont donnés comme paramètres au modèle pendant le processus d'apprentissage.	Peut être utilisé pour regrouper les données d'entrée en classes basées uniquement sur leurs propriétés statistiques.
La construction d'un jeu d'entraînement, de validation et de test est cruciale.	Signification des clusters et labélisation.
Les méthodes sont généralement rapides et précises.	La labélisation peut être effectuée même s'il n'y a qu'un faible échantillon d'objets appartenant à la classe.
Possibilité de généralisation : donner un résultat correct lorsqu'une nouvelle entrée dont on ne connaît pas la classe est présentée.	Pareil.

Tableau II.1 Comparaison des méthodes d'apprentissages supervisés et non supervisés.

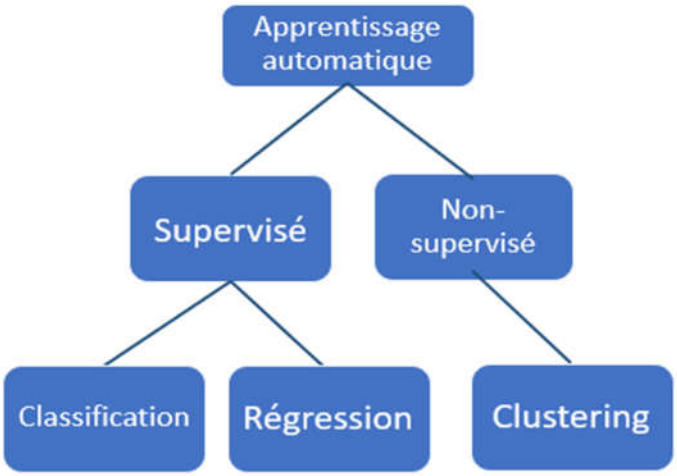


Figure II.3 Algorithme d'apprentissage automatique.

Chaque apprentissage automatique a son propre algorithme, la figure II.3 illustre la différence entre les algorithmes d'apprentissage.

5. Modélisation avec l'apprentissage automatique

Il existe un grand nombre d'algorithmes utilisées dans le ML. Compte tenu du contexte et des méthodes de collectes de données, il convient de présenter uniquement les algorithmes supervisés. Les choix retenus sont systématiquement motivés par les exemples trouvés dans la littérature.

5.1 Arbres de décisions et forêt d'arbres décisionnels

Le classificateur des arbres de décisions est une des méthodologies les plus utilisées en pratique de par sa nature robuste au bruit. Cette méthode classifie les instances en les arrangeant, depuis la racine de l'arbre jusqu'aux nœuds des feuilles, qui fournissent la classification des instances. Chaque nœud spécifie un test d'un attribut de l'instance et chaque branche descendant à un nœud correspond à une des valeurs possibles pour cet attribut. Cette méthodologie est particulièrement recommandée pour l'analyse de données IRM, comme lors de la détection de la maladie de Parkinson. Il existe également une méthode dite « forêt d'arbres décisionnels », elle permet d'augmenter la diversité en combinant plusieurs arbres de décisions [31].

5.2 K le plus proche voisin

K le plus proche voisin, en Anglais k-nearest neighbour (KNN), est un des algorithmes les plus appliqués. Il consiste à assigner à un échantillon non classifié, la classification du

plus proche des éléments précédemment classifiés. Pour classifier une nouvelle instance, la distance euclidienne (éventuellement avec un facteur de poids) est calculée entre l'instance et chaque instance d'entraînement. La classe de la nouvelle instance est assignée par rapport à celle présentant la plus courte distance. De façon plus générale, les k plus proches voisins sont calculés et la nouvelle instance est assignée à la classe qui est la plus présente parmi les k plus proches voisins, k étant un paramètre à choisir [31][32].

5.3 Réseaux de neurones

Cette méthodologie artificielle est inspirée des modèles des réseaux de neurones biologiques, dans le cerveau. Elle est composée d'un grand nombre de neurones interconnectés qui travaillent en union afin de résoudre un problème spécifique. Tout comme les humains, les réseaux de neurones apprennent par les exemples. L'apprentissage biologique nécessite des ajustements au niveau synaptique entre les neurones. Ceci est également vrai dans le cas des réseaux de neurones artificiels. Dans la littérature, les réseaux de neurones ont été utilisés avec succès dans plusieurs domaines y compris celui des réseaux informatiques pour la sélection du meilleur chemin selon des critères bien déterminés [31].

Le tableau II.2 récapitule les algorithmes avec leurs propriétés :

Algorithme	Propriétés
Arbres de décisions	Modèle de décisions basé sur les attributs et valeurs.
Forêts d'arbres décisionnels	Modèle de décisions basé sur les attributs et valeurs.
K plus proches voisins	Basé sur l'organisation de groupes à caractères similaires
Réseau de neurones artificiels	Inspiré des réseaux de neurones biologiques.

Tableau II.2 Algorithmes et leurs propriétés.

6. Utilisation de l'intelligence artificiel en réseaux informatique

Aujourd'hui, l'apprentissage automatique est principalement utilisé dans l'automatisation informatique, transformant les données pertinentes en logiciel décisionnel. Il peut également assurer une configuration optimale dans la mise en réseau, et pour améliorer l'analyse, la gestion et la sécurité. Pour cela, la technologie analyse la structure des données recueillies pour trouver des modèles inconnus. Comment l'apprentissage automatique peut-il

profiter aux opérateurs de réseaux spécialement dans les centres de données, et quel est le rôle du cette technologie dans la mise en réseau.

6.1 Appliquer l'apprentissage automatique dans la mise en réseaux

Les outils d'analyse basés sur l'apprentissage automatique sont excellents pour apprendre à quoi ressemble le comportement normal du réseau et mettre en évidence les anomalies par rapport à celui-ci. Cette prise de conscience entraîne l'utilité de l'apprentissage automatique dans la mise en réseau dans trois domaines : la gestion des performances, la gestion de la santé et la sécurité [33].

6.2 Gestion des performances

Les outils équipés d'apprentissage automatique peuvent aider à la fois dans la gestion du trafic instantané et dans la planification et la gestion des capacités à plus longue portée. Ces outils peuvent voir si le trafic augmente à certains endroits ou ne pas circuler dans d'autres, et ils peuvent diriger des réponses de gestion automatisées ou manuelles.

Par exemple, les outils de gestion de l'apprentissage automatique peuvent transférer la moitié du trafic destiné à un système d'un centre de données à un autre en fonction des conditions de trafic. Autre exemple de gestion de performances qui nous intéressent est en domaine d'équilibrage de données de prédire la meilleure route d'un trafic selon des critères réseaux tel que la bande passante, la gigue et la latence [33].

6.3 Gestion de l'accroissement matériel

Les analyses basées sur l'apprentissage automatique peuvent aider à repérer quand un composant réseau est dans les étapes initiales de défaillance et à prédire quand ces étapes initiales apparaîtront pour les nœuds actuellement sains. Les fournisseurs d'équipements de réseau intègrent de plus en plus de telles analyses dans des outils de gestion, en particulier ceux construits autour d'une offre SaaS [33].

6.4 Sécurité

La détection d'anomalies dans le comportement du réseau peut aider les équipes de cybersécurité à tout trouver, d'un nœud matériel compromis à un employé devenant un trompeur sur le réseau de l'entreprise. Les techniques Machine Learning ont considérablement amélioré l'espace d'analyse des menaces comportementales, ainsi que la détection et la correction distribuées des dénis de service (DDOS) [34].

7. Réseau de neurones artificiels

Depuis une dizaine d'années, l'utilisation des réseaux de neurones artificiels (RNA) s'est développée dans de nombreuses disciplines (sciences économiques, écologie et environnement, biologie et médecine…) [34]. Ils sont notamment appliqués pour résoudre des problèmes de classification, de régression, de catégorisation, d'optimisation, de reconnaissance des formes et de mémoire associative [37].

7.1 Définition

Les réseaux de neurones fonctionnent en répartissant les valeurs des variables dans des automates (les neurones). Ces unités sont chargées de combiner entre elles leurs informations pour déterminer la valeur du paramètre de discrimination. C'est de la connexion de ces unités entre elles qu'émerge la capacité de discrimination du RNA. Chaque neurone reçoit des informations numériques en provenance de neurones voisins, à chacune de ces valeurs est associé un poids représentatif de la force de la connexion. Chaque neurone effectue localement un calcul dont le résultat est transmis ensuite aux neurones avals [35].

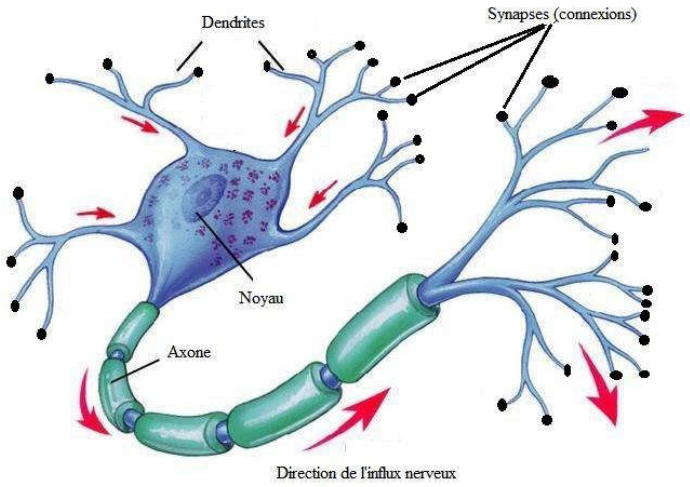


Figure II.4 Neurone biologique.

La figure II.4 ci-dessus illustre un neurone biologique, leurs nombres dans le cerveau dépasse 100 milliards, les synapses sont modélisées dans des poids, les dendrites par les relations entre les poids des neurones, l'axone par l'élément de sortie. Chaque neurone est relié par des autres neurones pour construire un réseau de neurones [37].

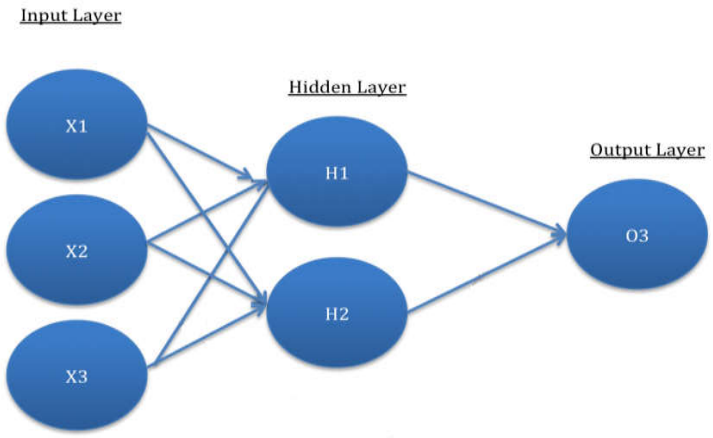


Figure II.5 Architecture d'un réseau de neurones artificiels.

La figure II.5 est un schéma très simplifié d'un réseau neuronal. Les trois neurones de gauche reçoivent les informations. Le traitement de ces données est déterminé par leurs connexions avec les neurones internes. Les neurones qui reçoivent une donnée sont activés. L'information finale est envoyée sur le dernier neurone ou sur l'organe effecteur [38].

7.2 Perceptron simple (Monocouche)

Le perceptron est un modèle d'apprentissage supervisé de classification binaire. Il s'agit d'un neurone formel muni d'une règle d'apprentissage qui permet de déterminer automatiquement les poids synaptiques de manière à séparer un problème d'apprentissage supervisé [39]. Si le problème est linéairement séparable, un théorème assure que la règle du perceptron permet de trouver une séparatrice entre les deux classes. Il peut être vu comme le type de réseau de neurones le plus simple. C'est un classifieur linéaire. Ce type de réseau neuronal ne contient aucun cycle (il s'agit d'un réseau de neurones à propagation avant) [41].

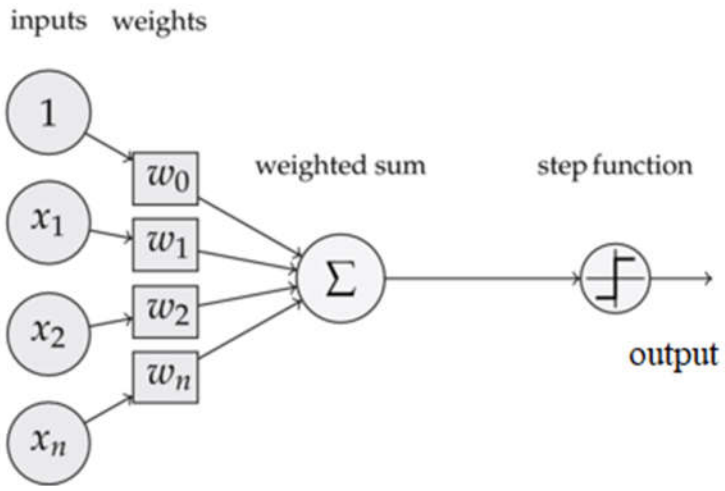


Figure II.6 Schéma d'un réseau de neurones artificiels perceptron monocouche.

La figure II.6 représente une architecture simple d'un perceptron, les entrées (input) {1, x1, x2 ...xn} peuvent être venues soit de l'environnement soit d'autre neurone, les poids (weights) {w0, w1, w2 ...wn} sélectionnent la valeur de l'entrée, la fonction de combinaison (weighted sum) fait la somation du produit entre les poids et valeurs d'entrées est la suivante :

$$S = f\left(\sum_{i=0}^{n} wij\ .\ xi\right)$$

Équation II.1 Fonction de combinaison.

La fonction d'activation (step function) calcule la sortie après la combinaison des poids des entrées est la fonction Seuil selon l'équation II.1.

Plusieurs fonctions d'activation sont utilisées (Fonction Gaussienne, Fonction Tangente Hyperbolique, Fonction sigmoïde) [41].

7.3 Perceptrons multicouches (MLP)

La famille de réseau majoritairement employé est le perceptron multicouche (PMC ou MLP) [37]. À lui seul ce type de réseau recouvre plus de 95 % des applications scientifiques et industrielles. Il comporte quelques dizaines à quelques centaines de neurones dans les cas usuels [40].

Le Perceptron multicouche est un Classifieur linéaire de type réseau neuronal formel organisé en plusieurs couches au sein desquelles une information circule de la couche d'entrée vers la couche de sortie uniquement, il s'agit donc d'un réseau de type feedforward. Chaque couche

est constituée d'un nombre variable de neurones, les neurones de la couche de sortie correspondant toujours aux sorties du système [39].

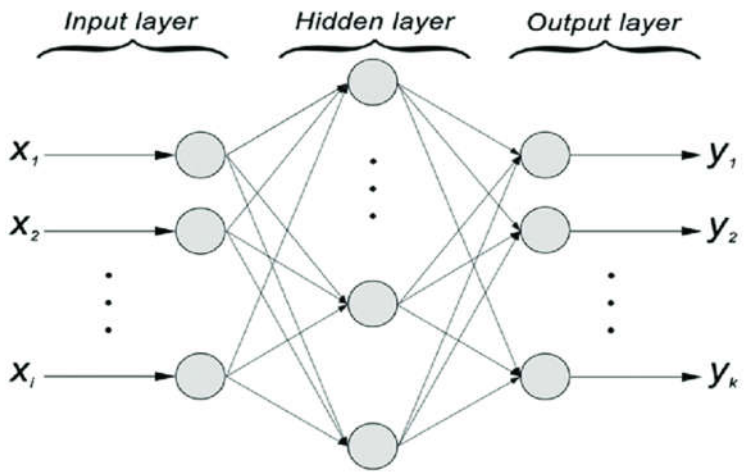


Figure II.7 Architecture perceptron multicouches.

La figure II.7 représente une architecture feedforward d'un perceptron multicouche, où x1… xi sont les entrées dans la couche d'entre [Input Layer] et y1… yk représentent les réponses dans la couche de sortie [Output Layer]. Une couche s'appelant la couche cachée [Hidden Layer] relie ces deux derniers et permet de faire le calcul [41].

8. Equilibrage de charge

8.1 Definition

Selon Moharana (2013) [42], l'équilibrage de charge [Load Balancing (LB)] est un mécanisme largement utilisé pour mieux tirer parti de toutes les ressources disponibles, en optimisant l'utilisation de celui-ci et le temps d'exécution des tâches. Son application est autant de répartir la charge de travail entre des ressources spécifiques d'un ordinateur. Comme des disques durs, que pour des serveurs de réseau en général, afin de garantir que la meilleure machine (c'est-à-dire celle avec plus de disponibilité) répond à une demande particulière avec la popularisation de l'internet haut débit. La demande d'une meilleure qualité de réponse aux services disponibles a considérablement augmenté, et pour que cela soit possible, les applications ont commencé à ne plus investir dans l'amélioration de leur serveur unique (en augmentant la mémoire disponible, par exemple), mais d'ajouter de nouvelles machines, donnant ainsi naissance au concept de ferme de serveurs. C'est-à-dire plusieurs ordinateurs avec une seule tâche, répondent aux demandes des clients pour une application donnée, ce qu'il offre une alternative plus économique par rapport à l'amélioration d'un seul

serveur. Dans ce sens, l'équilibrage de charge est une technique importante car il est chargé de transmettre cette demande à l'un des serveurs correspondants, afin qu'il puisse utiliser tous les traitements disponibles de sa capacité.

8.2 Equilibrage de charge côté client

Pour ce type d'équilibrage, une liste est donnée avec les adresses IP disponibles pour le client en charge de la sélection du serveur avec lequel communiquer. Cela a plusieurs implications en termes de sécurité car le client aura un accès direct aux serveurs disponibles ainsi que des problèmes d'optimisation car l'application ne peut pas garantir que le client choisira toujours le meilleur serveur à ce moment. Par exemple, si un grand nombre de clients décident de communiquer avec le serveur numéro 1, cela créera une surcharge sur le serveur et diminuera par conséquent le temps de réponse [43].

8.3 Equilibrage de charge côté serveur

Utilisation de DNS Dans cette méthode, le système de noms de domaine (DNS) est utilisé en associant un domaine à la liste des adresses IP disponibles dans le champ de serveurs. Le DNS sélectionne ensuite le serveur suivant à l'aide de l'algorithme de Round-robin (les adresses sont associées à une file d'attente et en cas de demande, celle qui se trouve au début de la file d'attente est utilisée, la supprimant et la repositionnant à la fin).[43]

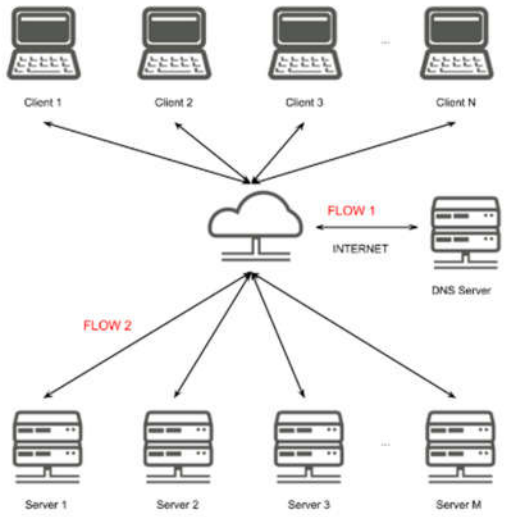


Figure II.8 L'utilisation d'un DNS avec Load balancing.

Un exemple du fonctionnement de la topologie pour ce type d'équilibrage est illustré à la figure II.8. Le client demande au serveur DNS de résoudre le domaine de l'application (flow

1) et il est responsable de choisir l'un des serveurs disponibles, avec lequel le client communiquera et enverra les données (flux 2) [43].

C'est la méthode la plus utilisée de nos jours. L'application dispose d'un matériel (l'équilibreur) qui sera responsable de l'écoute des clients et, lorsqu'une demande arrive, il est de sa responsabilité de rediriger vers l'un des serveurs du centre de serveurs. Cet équilibreur est le point de contact unique des clients avec l'application figure II.9, garantissant que les clients ne connaissent pas la division des fonctionnalités des serveurs et ne peuvent pas les contacter directement, ce qui est une bonne solution en termes de sécurité [43].

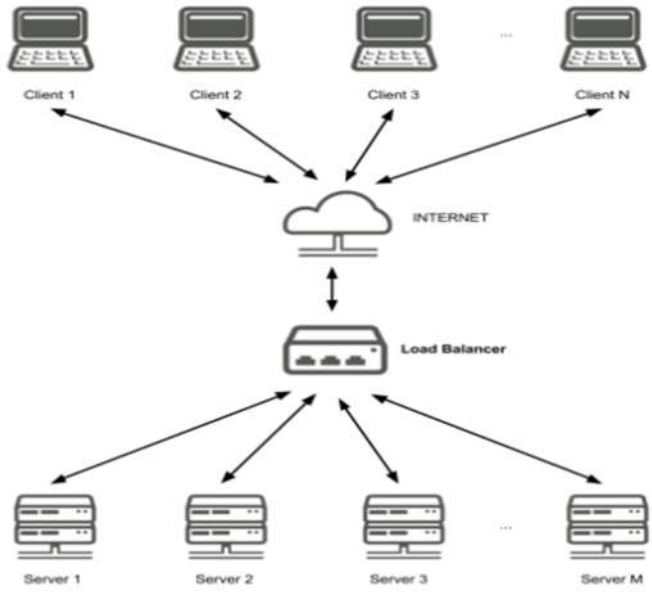


Figure II.9 load balancing côté serveur.

9. Algorithmes Load Balancing

Pour assurer une meilleure qualité d'équilibrage de charge entre les serveurs, il est très courant que les équilibreurs utilisent des algorithmes qui vont au-delà d'un simple choix aléatoire pour sélectionner le serveur suivant qui répondra à la prochaine demande. Aucune technique ne peut être considérée comme préférable, car chaque algorithme a sa particularité de pouvoir mieux agir dans différents scénarios. Pour cette raison, certains fabricants utilisent une combinaison de plusieurs algorithmes dans leurs appareils, augmentant ainsi les scénarios dans lesquels la distribution est la plus efficace. Les trois algorithmes suivants seront souvent utilisés par les fabricants :

9.1 Random

L'une des techniques les plus simples d'équilibrage de charge consiste à choisir l'un des serveurs au hasard pour chaque demande client. L'algorithme aléatoire peut permettre une bonne distribution de charge pour les serveurs ayant les mêmes caractéristiques car il garantit que les serveurs reçoivent les requêtes de manière équiprobable (à condition que le générateur de nombres aléatoires utilisé suive une distribution uniforme) [44].

9.2 Round-robin

L'algorithme de Round-robin utilise une file d'attente d'adresses IP et chaque fois qu'un client fait une demande, le serveur dont l'adresse se trouve au début de la file d'attente est sélectionné, puis il est retiré de cette position et inséré à la fin [45].

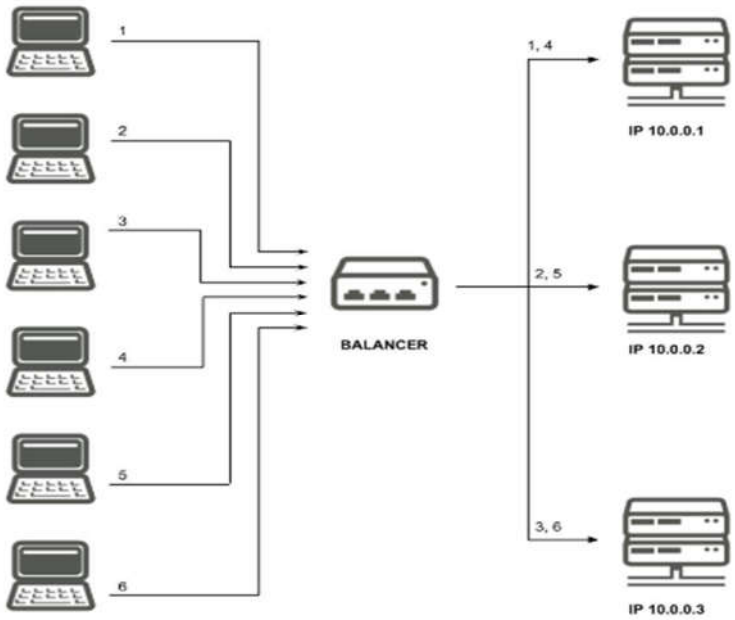


Figure II.10 Un scénario utilisant l'algorithme de Round-robin.

Dans cet exemple illustré dans la figure II.10, il est possible de voir un équilibreur avec trois serveurs différents. La première demande qui arrive est envoyée au premier serveur (dont l'adresse IP est 10.0.0.1), la seconde au deuxième serveur (dont l'adresse IP est 10.0.0.2) et le troisième serveur au dernier serveur (IP 10.0.0.3). La prochaine requête sera alors envoyée au serveur IP 10.0.0.1, car elle est désormais la première de la file d'attente [43].

Cette technique est intéressante pour un centre de serveurs avec des ordinateurs qui ont fondamentalement les mêmes spécifications (par exemple la même quantité de RAM, le même CPU, etc.) car elle répartit la charge de manière équivalente, en ignorant ces

caractéristiques [45]. Dans les cas où les serveurs ont des caractéristiques différentes des poids peuvent être ajoutés aux serveurs et par conséquent, les machines avec des poids plus importants recevront plus de charge que les machines avec des poids plus petits, permettant ainsi aux machines avec plus de capacité de recevoir plus de demandes que les autres.

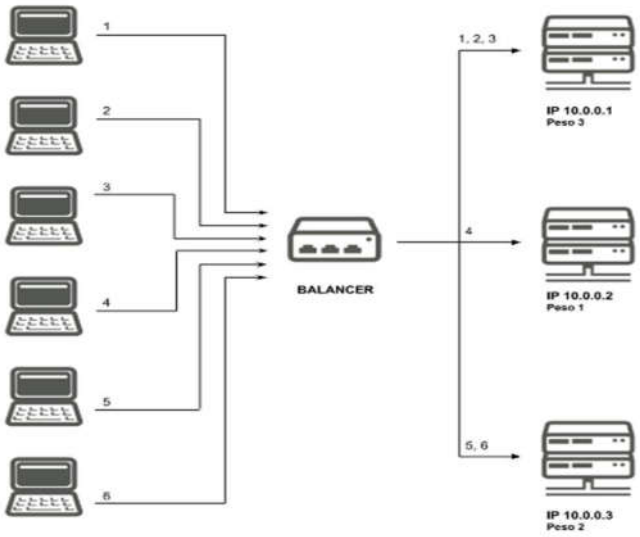


Figure II.11 Un scénario utilisant l'algorithme de Round-robin avec des poids.

La figure II.11 illustre un scénario avec l'utilisation de l'algorithme round-robin avec des poids. On y voit que le premier serveur a le poids 3 (bientôt, reçoit les trois premières demandes), le second a le poids 1 (réception des quatrièmes demandes) et le troisième poids 2 (réception des 2 dernières demandes).

9.3 Algorithme du Least-bandwidth

L'algorithme de moindre bande passante ou Least-bandwidth sélectionne le serveur avec la consommation de trafic réseau la plus faible pour répondre à la prochaine demande. L'équilibreur analyse et stocke la quantité de bytes qui sont transmis à chaque serveur et peut ensuite déterminer la machine suivante (celle qui a transmis le moins de données). Tout comme la méthode de Round-robin, la moindre bande passante peut également accepter des pondérations sur les serveurs. Si c'est le poids du serveur et la consommation de trafic réseau de celui-ci, alors la machine avec le rapport $\frac{Bi}{Wi}$ le plus bas est sélectionnée pour répondre à la demande suivante, donc, les machines avec des poids plus importants reçoivent plus de charge que les machines avec des poids plus petits [43].

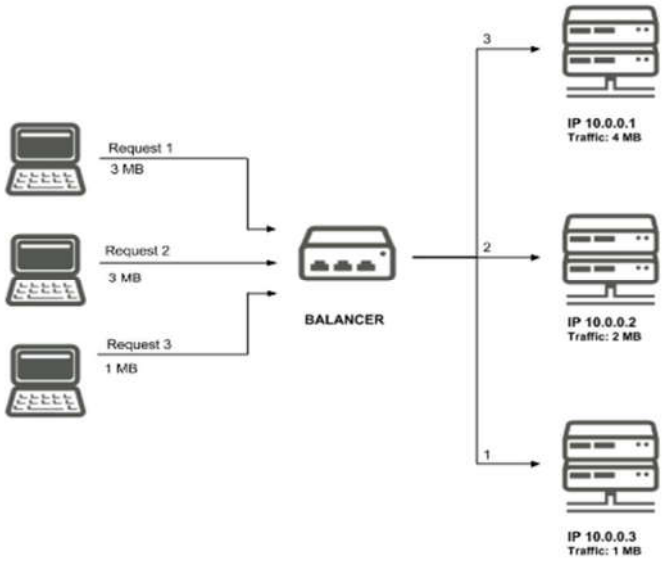


Figure II.12 Un scenario utilisant l'algorithme least-Bandwidth.

La figure II.12 illustre un scénario utilisant l'algorithme de moindre bande passante, les serveurs ont un trafic initial de 4 Mo pour le serveur avec l'adresse IP 10.0.0.1, 2 Mo pour le serveur avec l'adresse IP 10.0.0.2 et 1 Mo pour le serveur 10.0.0.3. Lorsqu'une demande est envoyée à un serveur, le nombre d'octets liés à ce paquet est ajouté au trafic total de cette machine.

10. Load balancing dans les réseaux définis par logiciel

En général, il existe deux étapes communes pour éviter le blocage dans l'architecture réseau défini par logiciel : La surveillance constante du réseau afin de reconnaître l'état des périphériques du réseau. La mise en œuvre de politiques pertinentes pour empêcher le blocage dans le réseau [46]. Les méthodes d'équilibrage de la charge peuvent être divisées en plusieurs catégories, que nous décrirons brièvement en bas. La première méthode pour empêcher le blocage du réseau est d'améliorer sa topologie. Trouve le meilleur itinéraire à l'aide de l'algorithme de routage Dijkstra, puis il commence à envoyer des données entre la source et la destination. Pendant ce temps, l'état du réseau est évalué régulièrement. Rapidement, lorsque l'efficacité d'un commutateur a atteint le seuil, le blocage peut se produire dans le réseau [47]. Par conséquent, cette méthode est essentiellement basée sur la recherche d'un nouveau chemin entre la source et la destination en supprimant le commutateur surchargé de la topologie et en envoyant la nouvelle topologie du réseau à l'algorithme de routage. Ensuite, tous les commutateurs seront mis à jour et le chemin actuel sera remplacé par le chemin précédent. Cependant, le problème avec cet algorithme est que le

chemin de la route alternative n'est pas déjà mesuré avec le nouveau flux, et cela constitue un état instable dans le réseau, il est également possible que la route, remplace le chemin par la capacité de vide, qui n'est pas un itinéraire optimal [48]. Clairement, de cette manière, il y a aussi un problème de surcharge dû à l'inspection périodique du réseau.

Dans cette approche, il y a toujours un surcoût problème dû à l'évaluation cyclique de l'état du réseau. En outre, cette méthode est efficace en raison des commandes élevées et du manque de priorité. De plus, si un paquet est envoyé sur une route, il est impossible de rediriger le paquet à nouveau, ce qui peut provoquer une interruption du réseau [49].

Une autre méthode pour réduire le blocage dans le réseau est d'équilibrer la charge entre les commutateurs de telle manière que la charge soit répartie entre tous les commutateurs du réseau. En fait, cette méthode est une combinaison de toutes les méthodes que nous avons déjà vue. Le problème avec cette méthode est que l'engagement de l'ensemble du réseau, ce qui rend le dysfonctionnement dans le réseau [50].

11. Travaux connexes

Dans cette section, nous avons regroupé quelques travaux qui ont utilisé les technologies du l'apprentissage automatique pour améliorer les performances d'équilibrage de charge dans les architectures du réseau défini par logiciel :

Jinke Yu et al en 2016 proposent un mécanisme d'équilibrage de charge basé sur une stratégie d'information de charge pour plusieurs contrôleurs distribués. Avec le mécanisme, un contrôleur peut prendre la décision d'équilibrage de charge localement aussi rapidement que possible. Les expériences basées sur Floodlight montrent que leur mécanisme peut équilibrer dynamiquement la charge de chaque contrôleur et réduire le temps d'équilibrage de charge qu'ils mettent en œuvre le contrôleur OpenFlow distribué basé sur Floodlight. Leur mécanisme d'équilibrage de charge proposé fonctionne comme l'un de ses modules. Ils choisissent Mininet pour émuler un réseau de commutateurs OpenFlow virtuels basés sur logiciel. Ils utilisent deux nœuds de contrôleur pour déployer un réseau SDN distribué [51].

Alex et Christian en 2018 présentent un équilibreur de charge pour les réseaux de centres de données basés sur OpenFlow, pour obtenir des performances élevées et une faible latence. Ils implémentent un algorithme de dynamicrouting dans l'équilibreur de charge. La tâche de l'algorithme est de distribuer le trafic des flux de réseau à venir et de faire en sorte que chaque chemin alternatif reçoive des quantités égales de charge de trafic. Il peut s'appliquer à des réseaux à grande échelle et planifier des flux de données de manière dynamique.

L'implémentation utilise le contrôleur OpenFlow Beaconand émulateur de réseau Mininet. Les résultats de l'évaluation démontrent que l'algorithme de routage d'équilibrage de charge dynamique est supérieur non seulement à l'algorithme de routage d'équilibrage de charge mais également à l'algorithme d'équilibrage de charge statique. Un réseau OpenFlow typique se compose de trois composants : le contrôleur OpenFlow, les commutateurs open-Flow et les hôtes. Chacun des commutateurs maintient une table de flux qui contient des informations de transfert. Le contrôleur et les commutateurs communiquent via des messages OpenFlow [36].

Mohammad Reza Mullah Khalili Meybodi et al en 2016, proposent un algorithme par la programmation interne du contrôleur de réseau défini par logiciel. Ils croient qu'en raison de l'augmentation du taux d'entrée de données d'une liaison, un échec du réseau au routage, se produit en rapport avec le taux d'inclinaison de la congestion, qui en général conduit à la perte de paquets, aux retards et à l'impossibilité d'envoyer des données complètes et correctes. Réduire les performances du réseau. Le contrôleur utilisé dans cette recherche est le contrôleur open source RYU qui est simulé dans l'environnement MININET et les résultats des expériences effectuées ont été obtenus en utilisant l'outil de création de trafic IPERF. Les résultats indiquent que la quantité de puissance opérationnelle et le retard moyen dans le centre de données sont indiqués par le pourcentage de charge sur le réseau et la taille des paquets. Le contrôleur agit selon les instructions spécifiées, et avec une diminution du délai moyen et une réduction moindre de la capacité opérationnelle du système, les paquets sont détruits et l'efficacité du système augmente [52].

12. Conclusion

Des notions de base sur les réseaux définis par la connaissance, ainsi que sur l’apprentissage automatique ont été présenté dans ce chapitre, l’intégration des algorithmes du l’intelligence artificielle dans la technologie du SDN pour améliorer Le processus d’équilibrage de charge dans le réseau défini par logiciel sera détaillé dans le chapitre suivant.

Chapitre III

Modélisation du réseau défini par les connaissances

III. Chapitre III

Modélisation du réseau défini par les connaissances

1. Introduction

Afin de pouvoir implémenter notre solution nous devons passer par une étape de conception ou de modélisation. Nous allons présenter la topologie fattree (très utilisée dans les centres de données), la collection des données relatives à notre réseau et enfin notre algorithme intelligent pour l'augmentation de la bande passante.

2. Description du travail

Avant de commencer à expliquer le travail, nous avons construit un graphe systématique illustratif du début de notre projet à la dernière étape de celui-ci, pour mieux comprendre les phases que nous avons empruntées dans notre travail.

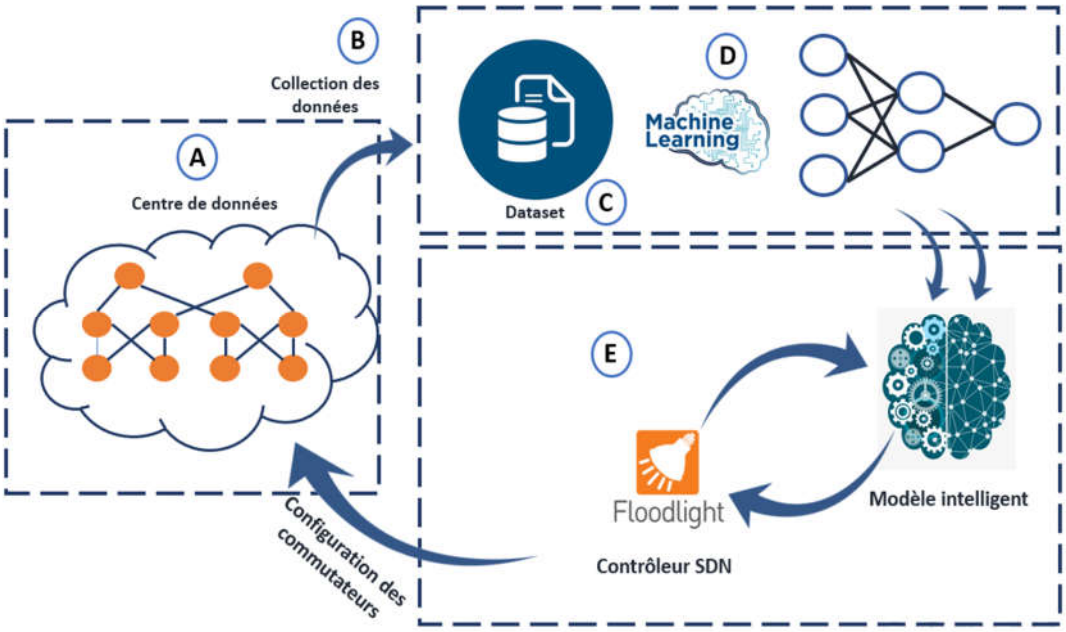


Figure III.1 Schéma synoptique du projet.

La figure III.1 présente un schéma synoptique de notre système proposé.

2.1 Centre de donnée et la topologie fattree (A)

Un data center ou centre de données, est une infrastructure composée d'un réseau d'ordinateurs et d'espaces de stockage. Cette infrastructure peut être utilisée par les entreprises pour organiser, traiter, stocker et entreposer de grandes quantités de données. En règle générale, une entreprise repose fortement sur les applications, les services et les

données contenues dans un centre de données. Il s'agit donc d'une part essentielle de l'entreprise au quotidien [54]. Pour fonctionner correctement, un centre de données doit aussi abriter l'infrastructure adéquate : un système distribution d'énergie, un commutateur électrique, des réserves d'énergie, des générateurs dédiés au backup, un système de ventilation et de refroidissement, et une puissante connexion internet. Une telle infrastructure nécessite un espace physique suffisamment vaste et sécurisé pour contenir tout cet équipement [56].

La topologie fattree a été proposée pour les centres de données pour la première fois en 2008, une fattree k-ary contient k centre de donnée appelés POD [Performance Optimized Data center], qui se connectent entre eux via des commutateurs de couche centrale [(Core-Layer]. Chaque POD contient trois couches, couche des commutateurs agrégations [Aggregation layer] et couche des commutateurs périphériques [Edge-Layer] où chaque commutateur d'une couche est lié à tous les commutateurs de l'autre couche [53].

Le nombre de port est le même dans chaque commutateur qui est symbolisé par k, à partir de la valeur k, on peut déduire le nombre de commutateurs Core, de commutateurs d'agrégation, de commutateurs Edge et le nombre maximale de serveurs (Hôtes) qui peuvent être connectés.

Le POD est un ensemble de k/2 commutateurs d'agrégation et de k/2 commutateurs Edge qui sont connectées entre eux. Le nombre de POD est k, aussi le nombre de port est égal à k.

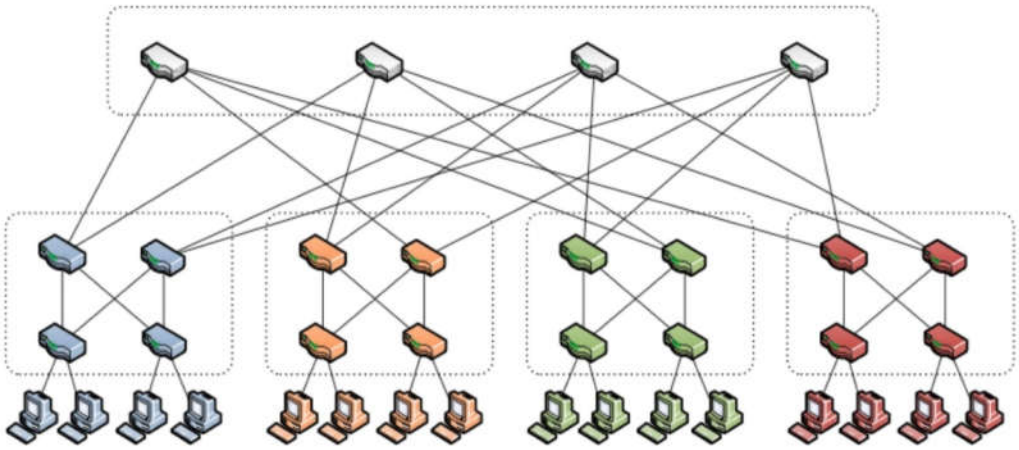


Figure III.2 Topologie fattree avec k=4.

La figure III.2 montre un exemple de centre de données avec la topologie fattree à 4 zones (k=4).

L'un des avantages de la topologie fattree c'est qu'elle permette la communication des serveurs en utilisant la capacité totale de leurs interfaces réseau, en plus, tous les modules du réseau sont identiques, par conséquent évite les commutateurs coûteux à haute densité de ports [55]. Aussi, les topologies fattree présentent certains avantages. Tous les commutateurs sont du même type avec le même nombre de ports, chaque port fournissant généralement la même vitesse, les hôtes finaux prennent également en charge la même vitesse. Il existe plusieurs chemins entre deux hôtes. Notez que chaque hôte doit d'abord être connecté à un commutateur périphérique. En particulier, dans une topologie fattree à 4 POD, il y a deux chemins entre deux commutateurs de périphérie dans un POD, et il y a quatre chemins entre deux commutateurs de périphérie qui se trouvent entre les POD [56].

Résumé de la topologie fattree	**Variable k**
Nombre de POD	K
Nombre de Core-Layer	$(K/2)^2$
Nombre d'Aggregation-Layer	$K^2/2$
Nombre d'Edge-Layer	$K^2/2$
Nombre de Switch	$5K^2/4$
Nombre de liaison	$K^3/2$
Nombre de Host (Server)	$K^3/4$

Tableau III.1 Résumé de la topologie fattree.

Le tableau III.1 représente le nombre de chaque paramètre dans la topologie en fonction de la valeur k. Si on prend un exemple avec k=16, nous obtiendrons le résultat affiché dans le tableau suivant :

POD	Core	Agg/Edge	Commutateur	Liaison	Host
16	64	128	320	2048	1024

Tableau III.2 Nombre des paramètres fattree k =16.

2.2 Chemins possibles

Pour récupérer les données concernant la route utilisée pour la transmission, il faut d'abord trouver les chemins possibles de chaque source et destination, pour cela le contrôleur SDN utilise le protocole OSPF (Open Short Path First) basé sur l'algorithme Dijkstra combiné avec l'algorithme Yen's.

L'algorithme de Dijkstra sert à résoudre le problème du plus court chemin. Il calcule des plus courts chemins à partir d'une source vers tous les autres nœuds dans un graphe [57].

L'algorithme Yen's [k-Shortest Path of Yen] renvoie non seulement le chemin le plus court, mais aussi le deuxième plus court, le troisième le plus court, etc… selon le K d'entré [57].

Si on utilise la topologie fattree (figure III.2) et on cherche une route entre une source et une destination pour faire la transmission des paquets, on trouve que on a quatre courts chemins possibles pour chaque transmission.

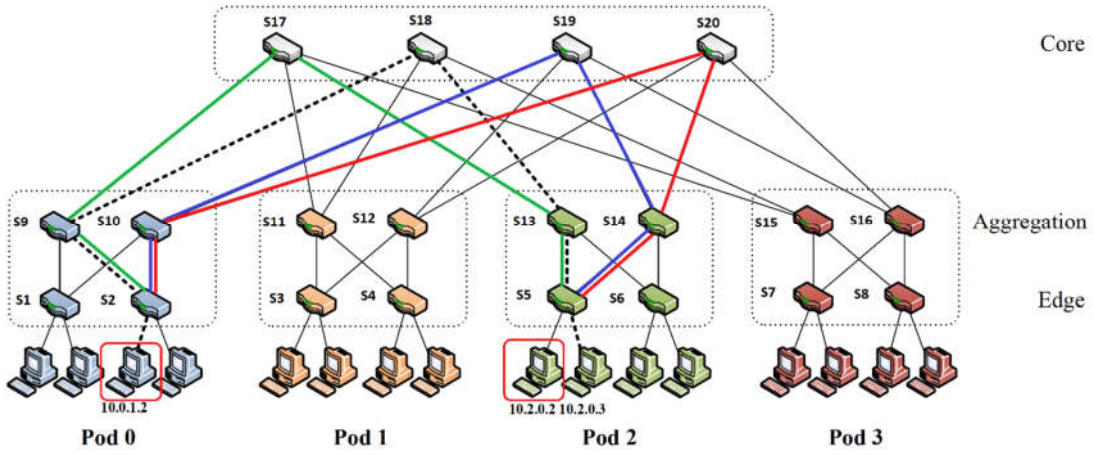


Figure III.3 Exemple d'un scenario dans une fattree.

Dans la figure III.3 un host dans le POD 0 avec l'adresse IP 10.0.1.2 veut communiquer avec l'host 10.2.0.2 dans le POD 2, nous aurons les quatre chemins possibles suivants :

- Route 1 : 10.0.1.2 ➔ S2 ➔ S9 ➔ S17 ➔ S13 ➔ S5 ➔ 10.2.0.2 En vert
- Route 2 : 10.0.1.2 ➔ S2 ➔ S9 ➔ S18 ➔ S13 ➔ S5 ➔ 10.2.0.2 Pointillé
- Route 3 : 10.0.1.2 ➔ S2 ➔ S10 ➔ S19 ➔ S14 ➔ S5 ➔ 10.2.0.2 En bleu
- Route 4 : 10.0.1.2 ➔ S2 ➔ S10 ➔ S120 ➔ S14 ➔ S5 ➔ 10.2.0.2 En rouge

Après avoir déterminer les chemins et les commutateurs, nous calculerons les métriques nécessaires de chaque interface en utilisant le contrôleur du réseau défini par logiciel et l'outil de collection sFlow-RT.

2.3 sFlow

sFlow est une technologie d'échantillonnage multifournisseurs intégrée dans les commutateurs et les routeurs. Elle offre la possibilité de surveiller en continu simultanément les flux de trafic au niveau de l'application à la vitesse du fil sur toutes les interfaces en temps réel [57].

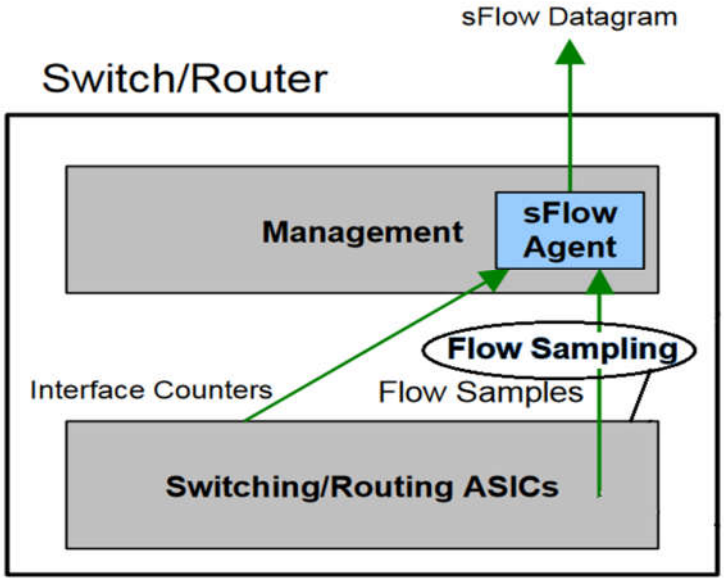


Figure III.4 sFlow Agent management.

L'agent sFlow est un processus logiciel qui s'exécute dans le cadre du logiciel de gestion de réseau au sein d'un périphérique (figure III.4). Il combine des compteurs d'interface et des échantillons de flux dans des datagrammes sFlow qui sont envoyés sur le réseau à un collecteur sFlow. L'échantillonnage des paquets est généralement effectué par les ASIC de transmission et routage, offrant des performances de vitesse. L'état des entrées de table de transmission, routages associés à chaque paquet échantillonné est également enregistré. L'agent sFlow effectue très peu de traitement. Il regroupe simplement les données dans des datagrammes sFlow qui sont immédiatement envoyés sur le réseau. Le transfert immédiat des données minimise les besoins en mémoire et en CPU associés à l'agent sFlow [57].

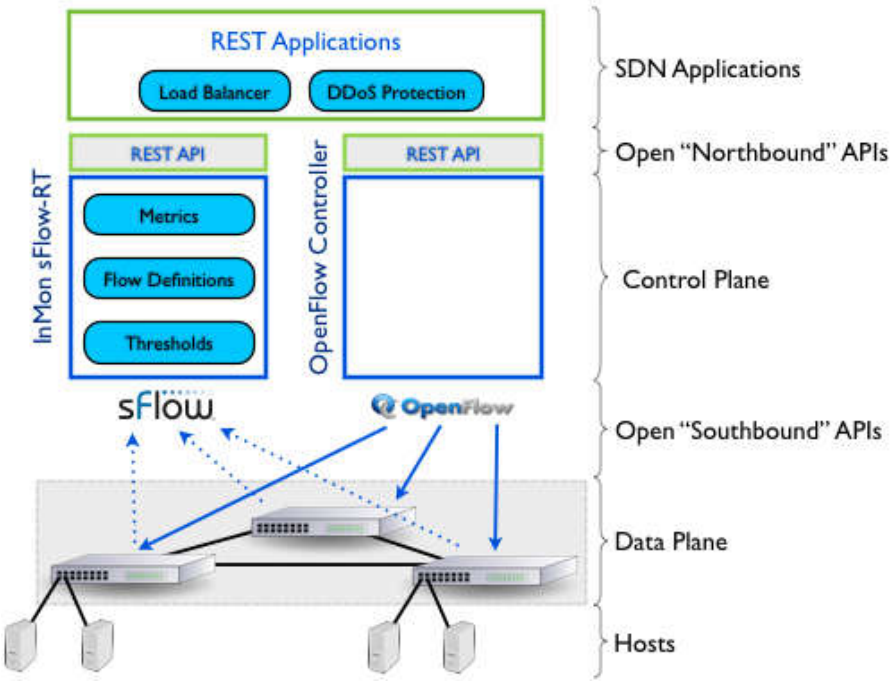


Figure III.5 Architecture sFlow avec le contrôleur SDN.

La figure III.5 montre l'architecture d'installation sFlow avec le contrôleur des réseaux définis par logiciel.

Le contrôleur utilise le protocole OpenFlow pour communiquer ou récupérer quelque métrique dans le plan de données, mais avec l'outil sFlow-RT le protocole utilisé est le protocole sFlow. OpenFlow étant un protocole de plan de contrôle, il est très difficile de déterminer avec précision les mesures du plan de données. La manière de procéder avec OpenFlow implique de faire des requêtes périodiques de statistiques de flux aux commutateurs du réseau. Le problème est que les statistiques ne sont jamais vraiment précises. Au moment où ils ont été traités sur le commutateur, envoyés sur le réseau, puis traités par le contrôleur, ils seront obsolètes. Les mesures dans le plan de données sont mieux gérées par des applications [58].

La norme sFlow est intégrée dans le plan de données de la plupart des commutateurs, offrant la visibilité en temps réel des utilisations des liaisons et des flux importants nécessaires pour prendre en charge les applications d'ingénierie du trafic.

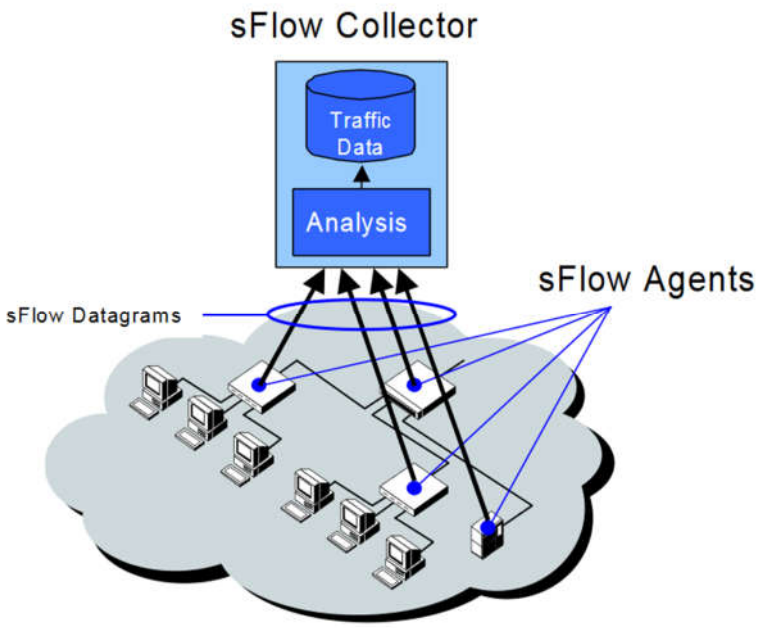


Figure III.6 Architecture de base sFlow-Agent.

La figure III.6 montre les éléments de base du système sFlow. Les agents sFlow à travers le réseau envoient en continu un flux de datagrammes sFlow à un collecteur central sFlow où ils sont analysés pour produire une vue riche, en temps réel et à l'échelle du réseau des flux de trafic [57].

2.4 Collecte des données (B)

Dans notre étude nous avons choisi comme métriques, la bande passante et la latence. Ce sont deux paramètres réseau notamment utilisées pour suivre la qualité de service [Quality of service (QoS)], ils indiquent l'état de charge dans un lien qui relie deux commutateurs ou plus et définissent la vitesse et la capacité d'un réseau.

2.4.1 Bande passante

Dans les réseaux informatiques, la bande passante fait référence à la quantité d'informations numériques que nous pouvons envoyer ou recevoir via une connexion dans un certain période de temps. Parfois, cela s'appelle le taux de transfert de données. La bande passante utilisée [Bandwidth utilization] est le pourcentage de bande passante utilisée par rapport à la bande passante totale disponible.

$$BWutilization\% = \frac{data * 100}{BWmax * interval}$$

Équation III.1 Bande passante.

Avec :

- ***Data*** : Les données qui circulent dans un canal.
- ***Interval*** : Période de transférer un *Data***.**
- ***BWmax*** : Est la capacité maximale définie par le fournisseur.

Exemple :

Les données capturées pendant 3 secondes sur un canal qui définit sur 10 Mb/s sont : 9×10^6 bits, donc la bande passante utilisée sur ce canal est : 29%

$$BWutilization\% = \frac{9000000 * 100}{(10 * 2^{20}) * 3} = 29\%$$

Équation III.2 Exemple bande passante.

2.4.2 Latence

Dans les réseaux informatiques, la latence de transmission [Latency] est une expression du temps nécessaire à un paquet de données pour transférer d'un point désigné à un autre. Idéalement, la latence sera aussi proche de zéro que possible. La latence du réseau peut être mesurée en déterminant le temps d'aller-retour (RTT) pour qu'un paquet de données se déplace vers une destination, où on peut la calculer par l'équation suivante :

$$Latencyi = \frac{NumberofByte}{Tx\ rate}$$

Équation III.3 Latence.

Où :

- *Latencyi* : latence d'un seul lien.
- *NumberofByte* : nombre des octets transmissent pendant une certaine période
- *Tx rate* : la vitesse de transmission

Donc le temps de latence de chaque chemin peut calculer par le *latencyi* et le nombre de lien (*n*) dans une route par l'équation suivante :

$$Latency = \frac{\sum_{i=1}^{n} latencyi}{n}$$

Équation III.4 latence du chemin.

2.5 Dataset (C)

Les méthodes d'apprentissage automatique apprennent à partir d'exemples. Il est important d'avoir une bonne compréhension des données d'entrée et des différentes paramètres utilisés lors de la description des données [58][59].

Dans ce projet nous sommes intéressés aux données structurées, pour faire une régression automatique. Pour cela nous avons construit une dataset par l'extraction des deux paramètres, la bande passante et la latence en utilisant l'outil sFlow-RT présenté dans le paragraphe 2.3.

Pour collecter les données, on lance une simulation de trafic dans un centre qui utilise la topologie fattree avec k = 4 (figure III.2). La collection de ces données se fait à chaque fois qu'un commutateur demande au contrôleur le port de transmission des paquets, à ce moment on fait une capture de l'état des quatre chemins possibles pour extraire les valeurs de la bande passante et la latence qui forment notre dataset, cette simulation se fait pendant un temps donné pour pouvoir collecter la base de données.

La figure suivante montre l'état de la base de données qui contient 68 Colonnes et 728 enregistrements en fonction du temps de simulation :

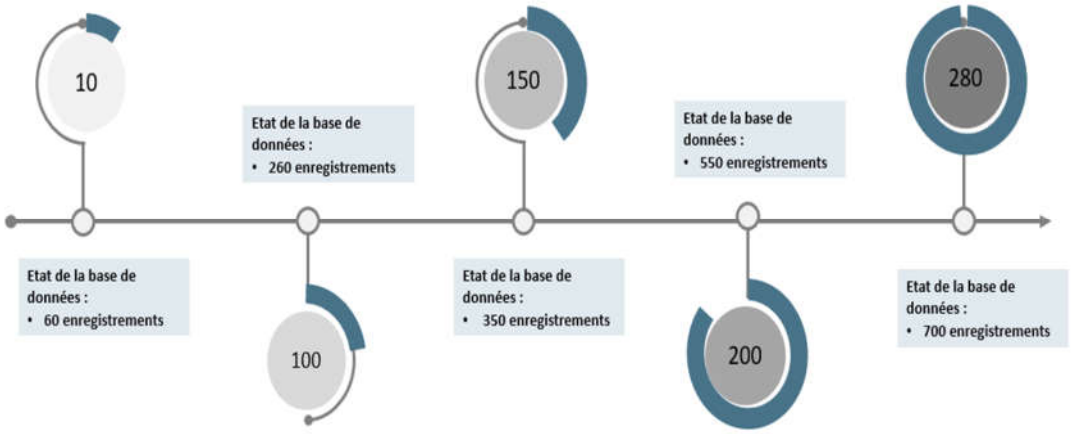


Figure III.7 Etat de la base de données.

Chaque ligne dans la dataset contient 64 valeurs, les 64 premières valeurs représentent la bande passante collectée des 4 routes, 16 valeurs de bande passante pour chaque route, comme nous avons vue dans la partie précédente, si un serveur veut communiquer avec d'autre serveur, il a quatre routes possibles, et chaque route dispose quatre liaisons, donc la bande passante d'une route est égale à 64. Les dernières 4 valeurs sont pour la latence des routes par ordre.

Bande passante	Latence
BP1.1 BP1.2…… BP1.64	Latency1.1….Latency.14
BP 2.1 BP2.2 ….. BP2.64	Latency2.1….Latency2.4
BPn.1 BPn.2 ….. BPn.64	Latencyn.1….Latencyn.4

Tableau III.3 Exemple d'enregistrement dans la base de données.

Lié à la description du tableau III.3, On prend un exemple d'enregistrement. Dans la ligne x, la bande passante BPx.1 jusqu'à la valeur BPx.16 est à propos de la route numéro 1 qui possède une latencyx.1, la bande passante BPx.17 jusqu'à la valeur BPx.32 est à propos de la route numéro 3 qui possède une latencyx.3 et ainsi de suite.

2.6 Prétraitement des données (Normalisation des données)

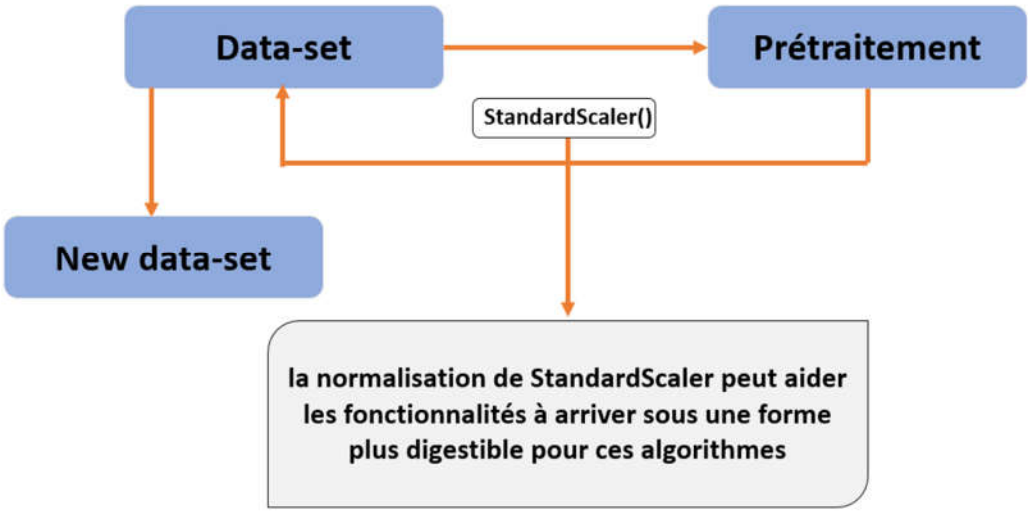


Figure III.8 Etape de prétraitement de données.

La figure III.8 décrit le prétraitement effectué dans notre projet.

Dans tout processus d'apprentissage automatique, le prétraitement des données est une étape au cours de laquelle les données sont transformées ou encodées, pour les amener à un état tel que la machine peut facilement les analyser. En d'autres termes, les caractéristiques des données peuvent désormais être facilement interprétées par l'algorithme de l'apprentissage automatique [60].

2.7 Apprentissage automatique (D)

L'analyse de régression est un concept fondamental dans le domaine de l'apprentissage automatique. Il relève de l'apprentissage supervisé dans lequel l'algorithme est formé à la fois avec des caractéristiques d'entrée (la bande passante) et des étiquettes de sortie (La latence). Il aide à établir une relation entre les variables en estimant comment une variable affecte l'autre.

La fonction de régression dans notre cas pourrait être représentée par :

$$Y = f(X)$$

Équation III.5 Fonction de régression.

Où Y serait un tableau de 4 éléments, chaque élément représente une valeur de latence des quatre chemins, et X serait les caractéristiques d'entrée comme la bande passante de chaque interface.

Les performances d'un modèle de régression peuvent être comprises en connaissant l'erreur d'apprentissage des prédictions faites par le modèle et l'exactitude. On peut également mesurer les performances en sachant à quel point notre ligne de régression de la prédiction correspond à l'ensemble de données. Un bon modèle de régression est celui où la différence entre les valeurs réelles ou observées et les valeurs prévues pour le modèle sélectionné est petite (presque 0) et non biaisée pour les ensembles de données d'apprentissage, de validation et de test.

Les éléments construire des réseaux de neurones sont les neurones artificiels. Ce sont de simples unités de calcul qui ont des signaux d'entrée pondérés et produisent un signal de sortie à l'aide d'une fonction d'activation.

Nous avons présenté chaque neurone par une valeur de la bande passante et la latence, le nombre de neurone d'entré est égale 64 entrées et 4 neurones pour la couche de sortie. Nous avons deux couches cachées, la première couche contient 80 neurones et 42 neurones pour la deuxième couche comme illustrer la figure III.9.

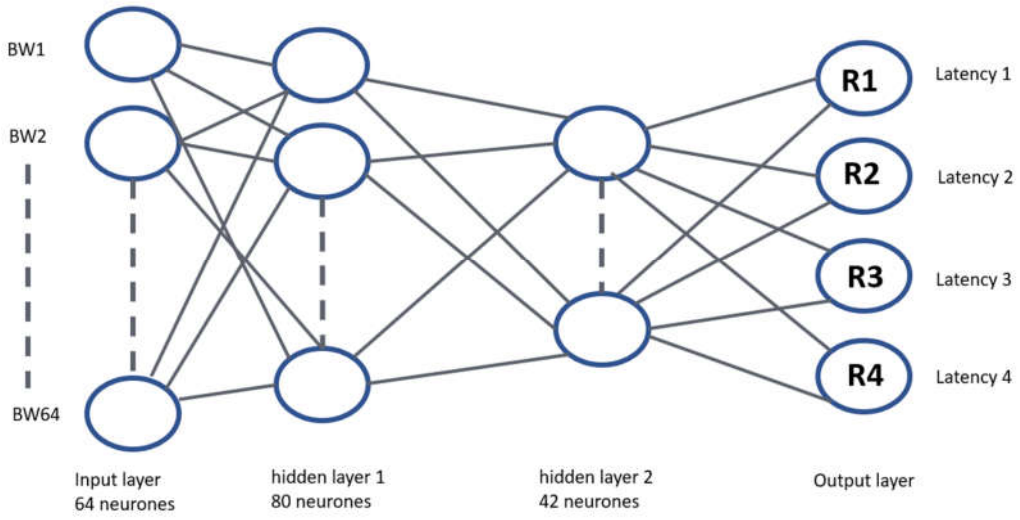


Figure III.9 Configuration du modèle multicouche.

Il existe de nombreuses méthodes pour déterminer le nombre correct de neurones à utiliser dans les couches cachées [36] :

- Le nombre de neurones doit être compris entre la taille de la couche d'entrée et la taille de la couche de sortie.
- Le nombre de neurones doit être 2/3 de la taille de la couche d'entrée, plus la taille de la couche de sortie.
- Le nombre de neurones doit être inférieur à deux fois la taille de la couche d'entrée.

2.8 Modèle du RNA dans le contrôleur SDN (E)

La dernière partie de notre projet est de configurer le modèle RNA dans le contrôleur SDN pour permettre de tester les résultats et de faire la transmission automatiquement à l'aide de ce modèle, la figure suivante montre la nouvelle interaction entre les trois plans de l'approche réseau défini par logiciel :

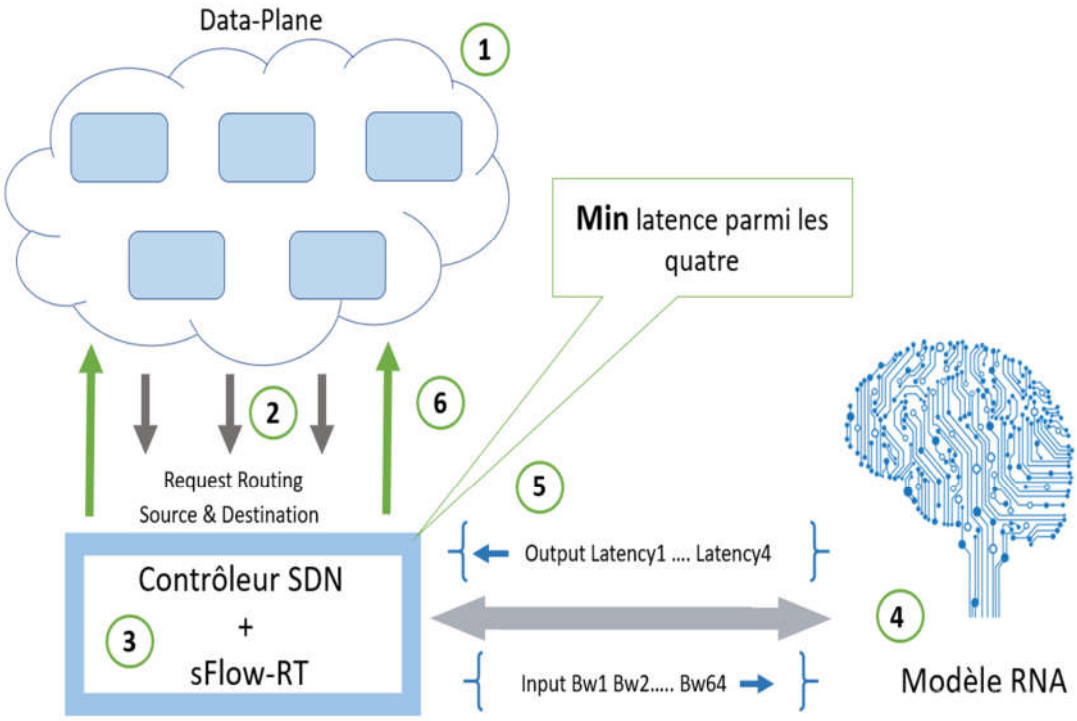


Figure III.10 diagramme illustratif de notre modèle.

La figure III.10 présente en étape les processus suivants :

- **1** Représente le plan de données qui contient la topologie fattree.
- **2** Dans cette étape, on récupère les informations de transmission de la source et destination pour calculer les chemins possibles.
- **3** Après pouvoir calculer nos chemins avec le contrôleur SDN exploitant l'algorithme Dijkstra, on récupère les valeurs de la bande passante en utilisant sFlow-RT.
- **4** On utilise 64 valeurs de bande passante comme entrée pour notre modèle de réseaux de neurone artificiel.
- **5** L'output des valeurs d'entré du modèle RNA sont quatre valeurs de latence où on doit calculer la minimal latence parmi eux.
- **6** On se servant de la latence minimale calculé dans étape 5, et le chemin sélectionné on peut finalement installer et configurer les commutateurs de notre topologie.

3. Conclusion

Nous avons proposé un modèle prédictif d'équilibrage de charge dans ce réseau de topologie fattree en exploitant les données prétraitées de notre réseau et en utilisant les réseaux de neurones artificiels. Les résultats obtenus seront présentés et discutés dans le chapitre suivant.

Chapitre IV

Expérimentation et résultats

IV. Chapitre IV

Expérimentation et résultats

1. Introduction

Dans ce chapitre nous allons s'étaler sur l'implémentation de notre solution, cette implémentation va nous permettre de tester la performance de notre modèle et par conséquent la valider.

2. Environnement de développement

2.1 Environnement matériel

Durent le développement de notre approche, on a utilisé un ordinateur HP possédant les caractéristiques suivantes :

- Modèle : HP ENVY 15-k002ne.
- Processeur : Intel(R) Core (TM) i7-4510M CPU @ 2.00 GHz 2.60 GHz.
- Mémoire : 8GO.
- Système d'exploitation : Ubuntu 18.04, 64 bits.

2.2 Environnement logiciel

Environnement	Définition	Icon
Eclipse IDE	Eclipse est un environnement de développement intégré (IDE) pour le développement d'applications utilisant le langage de programmation Java et d'autres langages [61].	
Sublime Text	Sublime Text est un éditeur de texte puissant, il est conçu pour prendre en charge plusieurs langages de programmation tel que Python [62].	

Tableau IV.1 Environnement logiciel.

L'environnement logiciel consiste de deux logiciels, IDE et un éditeur de texte présenter dans le tableau IV.1.

3. Emulateur et contrôleur SDN

La partie infrastructure de notre architecture sera customisée et générée sur la plateforme Mininet. Pour les exemples d'application que nous fournissons dans ce chapitre utilisent le contrôleur SDN Floodlight open source.

3.1 Emulateur Mininet

Mininet est un émulateur de réseau permet de crée un réseau d'hôtes virtuels, de commutateurs, de contrôleurs et de liens. Les hôtes Mininet exécutent un logiciel de réseau Linux standard et ses commutateurs prennent en charge OpenFlow pour un routage personnalisé très flexible et un réseau défini par logiciel. Mininet prend en charge la recherche, le développement, l'apprentissage, le prototypage, le débogage et toute autre tâche qui pourrait bénéficier d'un réseau expérimental complet [63].

3.2 Contrôleur Floodlight

Le contrôleur Floodlight basé sur Java sous licence Apache, est l'une des contributions importantes de Big Switch Networks à la communauté open source. Floodlight n'est pas aussi populaire aujourd'hui qu'il l'était aux premiers jours du SDN, mais ses API sont de nature suffisamment similaire à d'autres API Java OpenFlow pour être utiles en tant qu'outil de formation. Il est possible de télécharger le code source et d'examiner les détails du contrôleur lui-même. Floodlight est également fourni avec un certain nombre d'exemples d'applications, tels qu'un commutateur d'apprentissage et un équilibreur de charge, qui fournissent d'excellents exemples supplémentaires pour le lecteur intéressé [64].

Figure IV.1 Icon Floodlight.

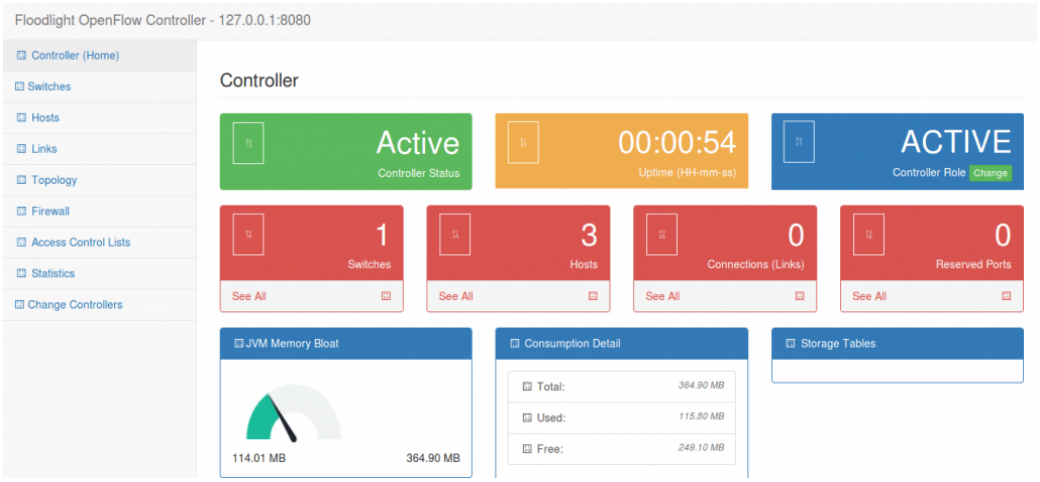


Figure IV.2 Interface Rest API Floodlight.

4. sFlow-Real Time

L'outil sFlow-RT fournit les données nécessaires pour contrôler et gérer efficacement l'utilisation du réseau, garantissant ainsi que les services réseau offrent un avantage concurrentiel.

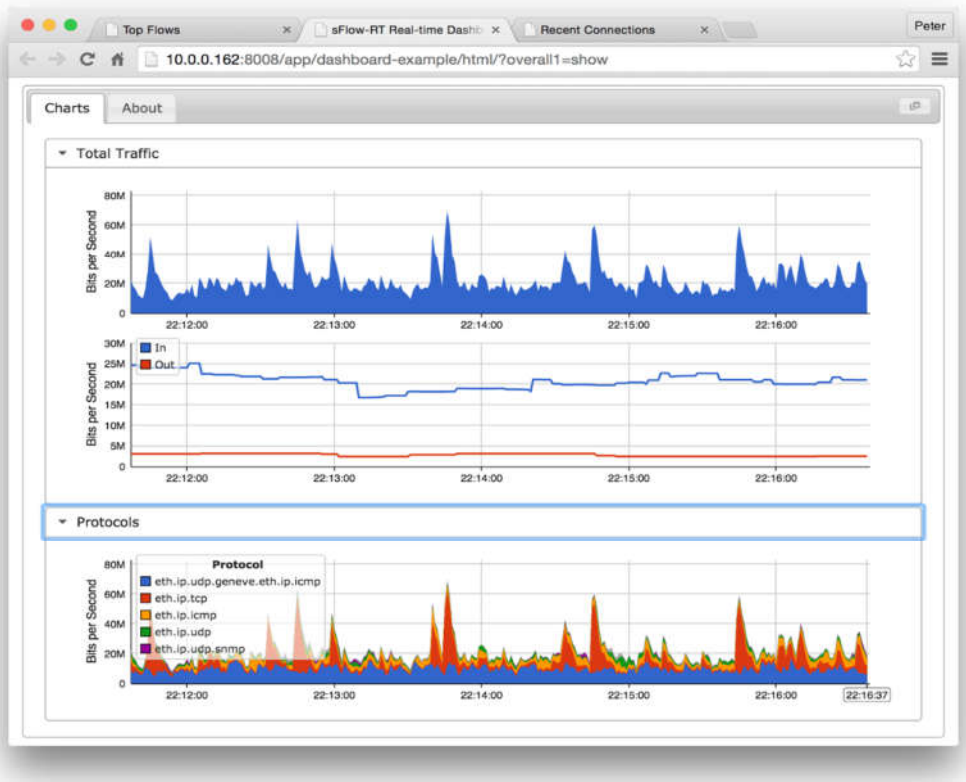


Figure IV.3 Rest API sFlow-Real Time.

5. Langages de développement

Nous avons réalisé notre projet avec deux langages de programmation différent, Le Java pour le contrôleur SDN Floodlight et le Python pour la création de réseau et l'implémentation de l'apprentissage automatique avec L'API Keras.

Langage de programmation	Définition	Bibliothèque utilisées
Java	La technologie Java définit à la fois un langage de programmation orienté objet et une plateforme informatique [65].	– Floodligth Controller – JSonURL – Deeplearning4j
Python	Python est le langage de programmation le plus utilisé dans le domaine du Machine Learning [66].	– Mininet – Keras Tensorflow – TensorBoard

Tableau IV.2 Langage de programmation.

6. Implémentation

6.1 Création de la topologie fattree

Notre choix dans ce projet c'est fait sur la topologie fattree avec 4 zones de POD, dont l'utilisation dans les centres de données est très répondue. Cette topologie augmente significativement la bande passante utilisée. Ainsi nous avons utilisé l'émulateur Mininet pour la création de cette topologie, car il offre avec le langage de programmation python la possibilité de créer des topologies dynamiques, souples et modifiable.

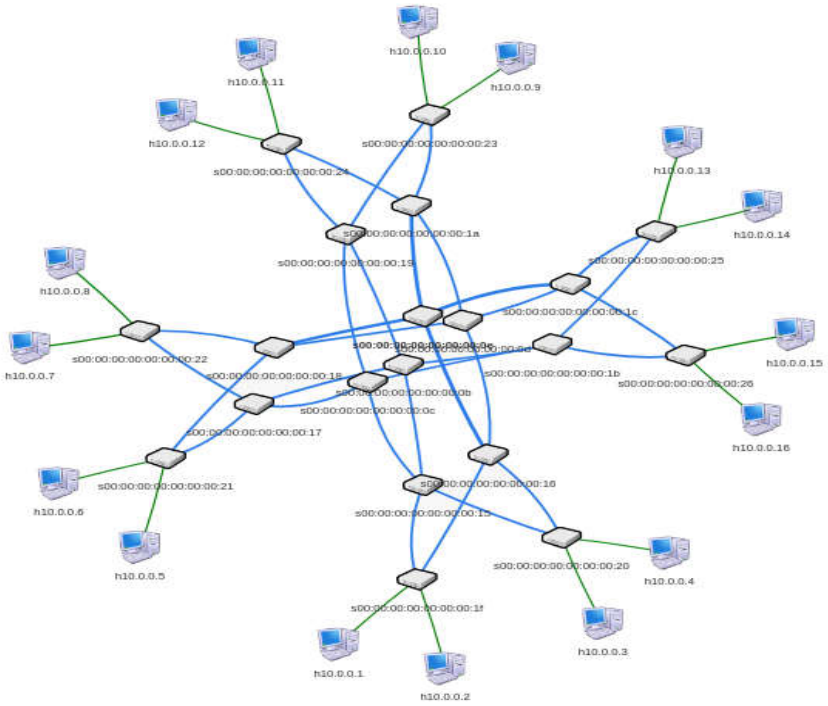


Figure IV.4 Topologie fattree k=4 dans le contrôleur Floodlight.

Dans la figure IV.4 nous présentons une topologie fattree avec k=4.

6.2 Scenario de simulation

Le scenario de communication des hôtes est montré dans la figure suivante :

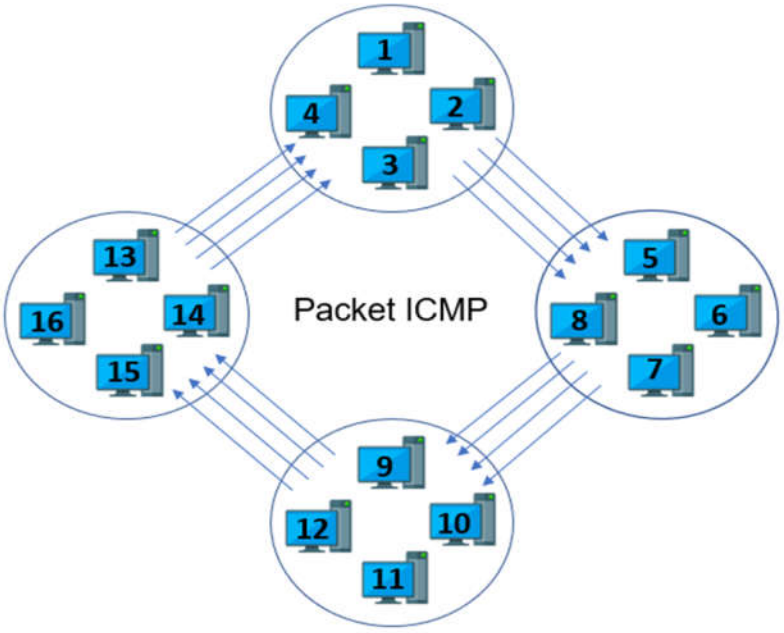


Figure IV.5 Scenario de communication.

Le premier trafic généré va de l'hôte 1 à l'hôte 5, le suivant de l'hôte 2 à l'hôte 6, ..., se terminant par l'hôte 16 à l'hôte 4, au total 16 trafics sont actifs, qui ont 4 routes pour transporter les informations entre les POD. Nous avons utilisé le multithreading pour pinger tous les hôtes au même temps.

6.3 Extraction des métriques avec sFlow-RT

L'outil sFlow-RT permet de contrôler l'état de charge du réseau, et fournit la bande passante utilisée sur chaque liaison, avec REST API et l'application Mininet-dashboard de sFlow-RT on peut extrait la bande passante utilisée de tous les ports. On applique un script en langage java qui permet de lire et récupérer les données enregistrées dans un URL json (127.0.0.1 :8008/app/mininet-dashboard/scripts/metrics.js/trend/json) en temps réel.

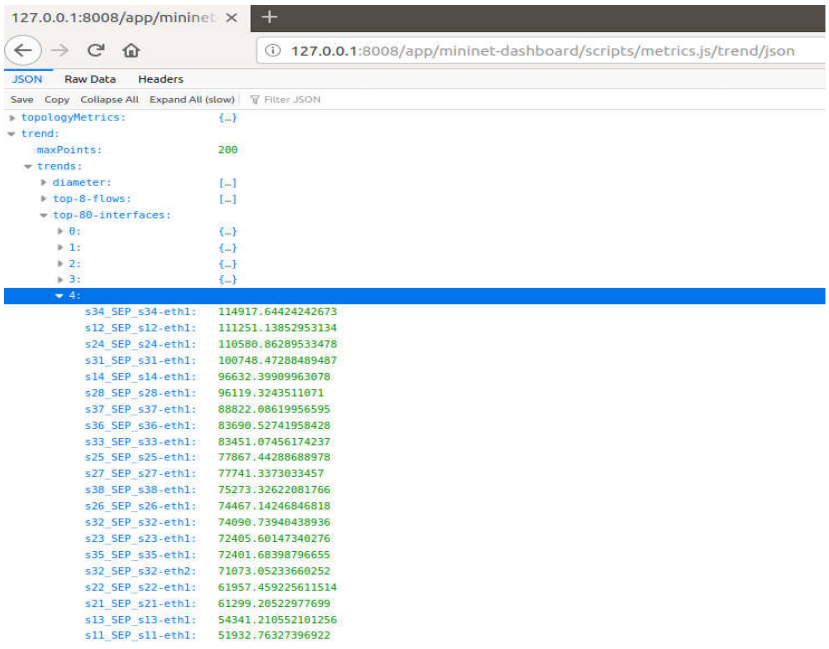


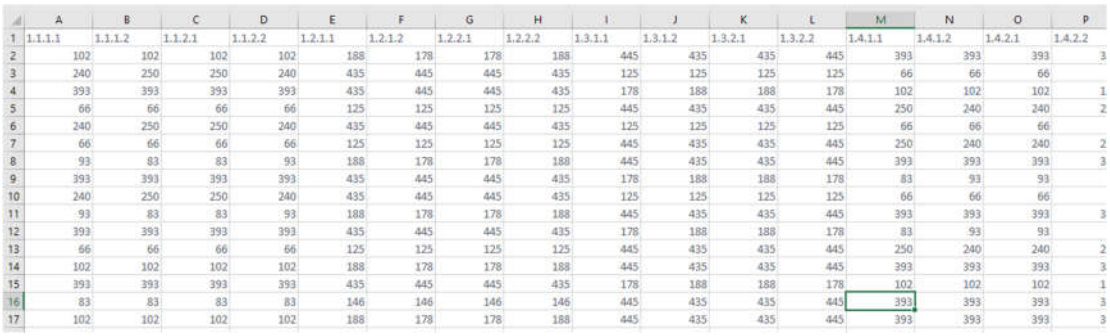
Figure IV.6 Application Mininet Dashboard.

La figure IV.6 illustre la consommation de la bande passante de chaque port (ex : s36_SEP_s36-eth1 c'est-à-dire le port numéro 1 de switch s36).

6.4 Description dataset

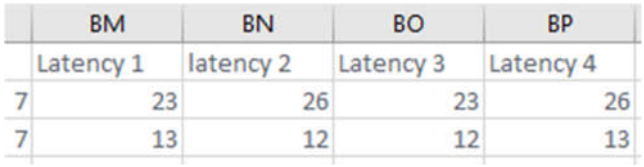
Le contrôleur Floodlight fournit la latence de chaque chemin par la fonction (getLatency ( )). La bande passante et la latence ont été enregistré dans un fichier csv, où chaque ligne représente 64 valeurs de bande passante utilisée + 4 valeurs pour la latence de chaque source et destination, nous irons utiliser ce fichier pour le modèle d'apprentissage

automatique, les deux captures suivantes représentent les données que nous avons déjà colletées.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	1.1.1.1	1.1.1.2	1.1.2.1	1.1.2.2	1.2.1.1	1.2.1.2	1.2.2.1	1.2.2.2	1.3.1.1	1.3.1.2	1.3.2.1	1.3.2.2	1.4.1.1	1.4.1.2	1.4.2.1	1.4.2.2
2	102	102	102	102	188	178	178	188	445	435	435	445	393	393	393	3
3	240	250	250	240	435	445	445	435	125	125	125	125	66	66	66	
4	393	393	393	393	435	445	445	435	178	188	188	178	102	102	102	1
5	66	66	66	66	125	125	125	125	445	435	435	445	250	240	240	2
6	240	250	250	240	435	445	445	435	125	125	125	125	66	66	66	
7	66	66	66	66	125	125	125	125	445	435	435	445	250	240	240	2
8	93	83	83	93	188	178	178	188	445	435	435	445	393	393	393	3
9	393	393	393	393	435	445	445	435	178	188	188	178	83	93	93	
10	240	250	250	240	435	445	445	435	125	125	125	125	66	66	66	
11	93	83	83	93	188	178	178	188	445	435	435	445	393	393	393	3
12	393	393	393	393	435	445	445	435	178	188	188	178	83	93	93	
13	66	66	66	66	125	125	125	125	445	435	435	445	250	240	240	2
14	102	102	102	102	188	178	178	188	445	435	435	445	393	393	393	3
15	393	393	393	393	435	445	445	435	178	188	188	178	102	102	102	1
16	83	83	83	83	146	146	146	146	445	435	435	445	393	393	393	3
17	102	102	102	102	188	178	178	188	445	435	435	445	393	393	393	3

Figure IV.7 Bande passante utilisée.



	BM	BN	BO	BP
	Latency 1	latency 2	Latency 3	Latency 4
7	23	26	23	26
7	13	12	12	13

Figure IV.8 Latence.

6.5 Implémentation du réseau de neurones artificiels

Nous avons utilisé la bibliothèque Keras pour le prétraitement de données et l'apprentissage automatique, Nous allons maintenant créer le modèle conceptuel proposé dans la figure III.9 du chapitre précédent.

Le modèle du réseau de neurones artificiels est composé de 2 couches cachées, il prend en entrée 64 valeurs de la bande passante pour prédire 4 valeurs la latence. Donc la première couche d'entrée contient 64 neurones, deuxième couche 80 neurones, troisième couche 42 neurones et la dernière couche est compose de 4 neurones de sortie comme illustre la figure suivante :

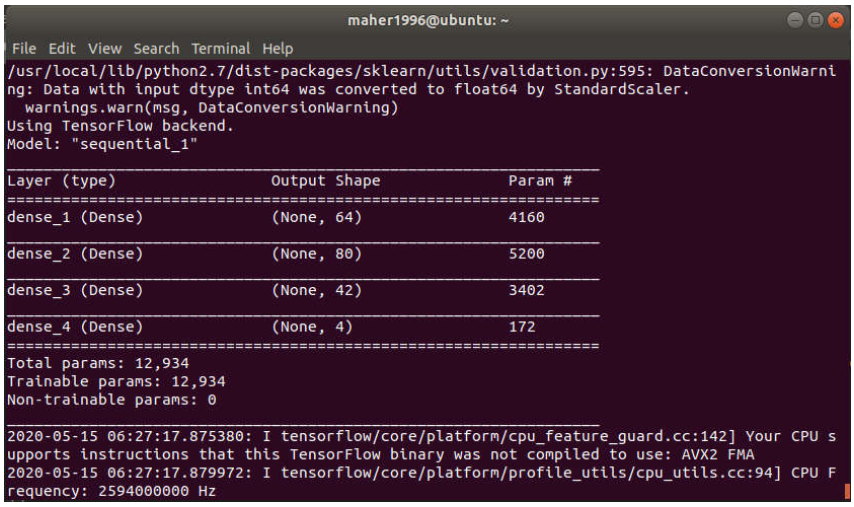


Figure IV.9 Description de modèle crée.

La figure IV.9 représente la création de réseau de neurones artificiels par la fonction Sequential(), la fonction Dense() a un but de créer les couches et spécifier le mode d'activations des fonctions et le nombres de neurone dans chaque couche.

7. Résultats et discussions

Dans cette section, nous présentons les résultats obtenus dans notre simulation :

7.1 Exactitude

Pour tester l'exactitude de notre réseau de neurones artificiels, nous avons exécuté notre modèle respectivement avec 1000, 1500, 2000 itérations. Nous avons obtenu 79%, 83%, 83.2%, comme présenté le tableau suivant :

Exécutions	Itérations	Exactitude
1	1000	79%
2	1500	83%
3	2000	83.2%

Tableau IV.3 Résultats d'exactitude.

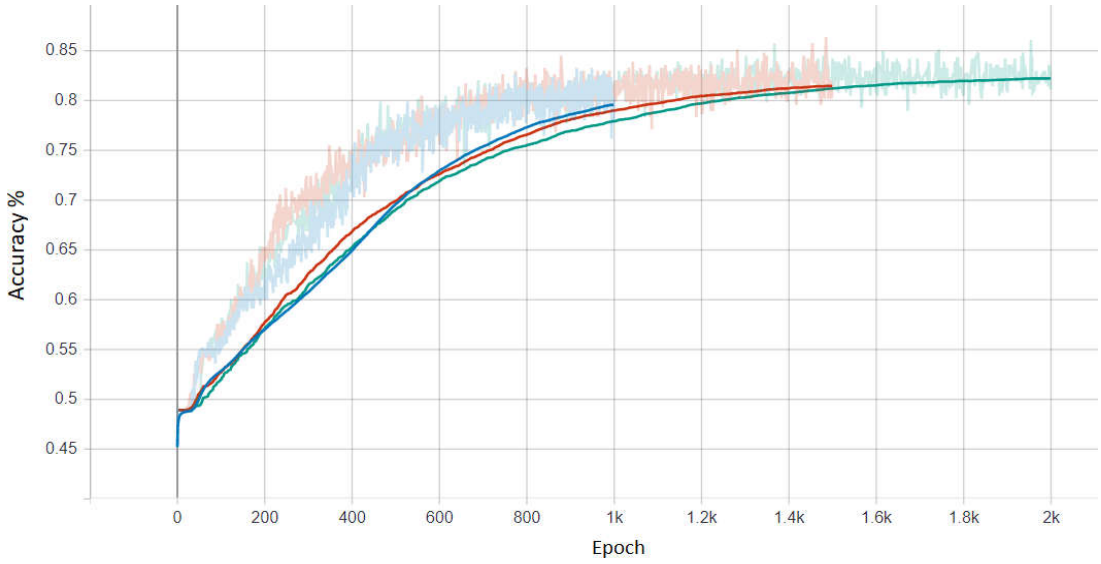


Figure IV.10 Développement d'exactitude [Accuracy].

Lors de la première exécution présentée en bleu dans le graphe figure IV.10, on a fixé le nombre d'itération à 1000 qui a résulté une exactitude acceptable égale à 79%, ce taux est élevé quand le nombre d'itérations était changé dans la deuxième exécution à 1500 pour donner une précision égale à 83% présenté en rouge dans le même graphe.

La dernière exécution présentée en vert montre une exactitude presque similaire au résultat de la deuxième exécution égale à 83.2% pour 2000 itérations.

7.2 Erreur d'apprentissage

Le même principe que le test d'exactitude, on a testé l'erreur d'apprentissage, le tableau suivant montre les résultats obtenus :

Exécutions	Itérations	Erreur
1	1000	0.25
2	1500	0.20
3	2000	0.19

Tableau IV.4 Résultats d'erreur d'apprentissage.

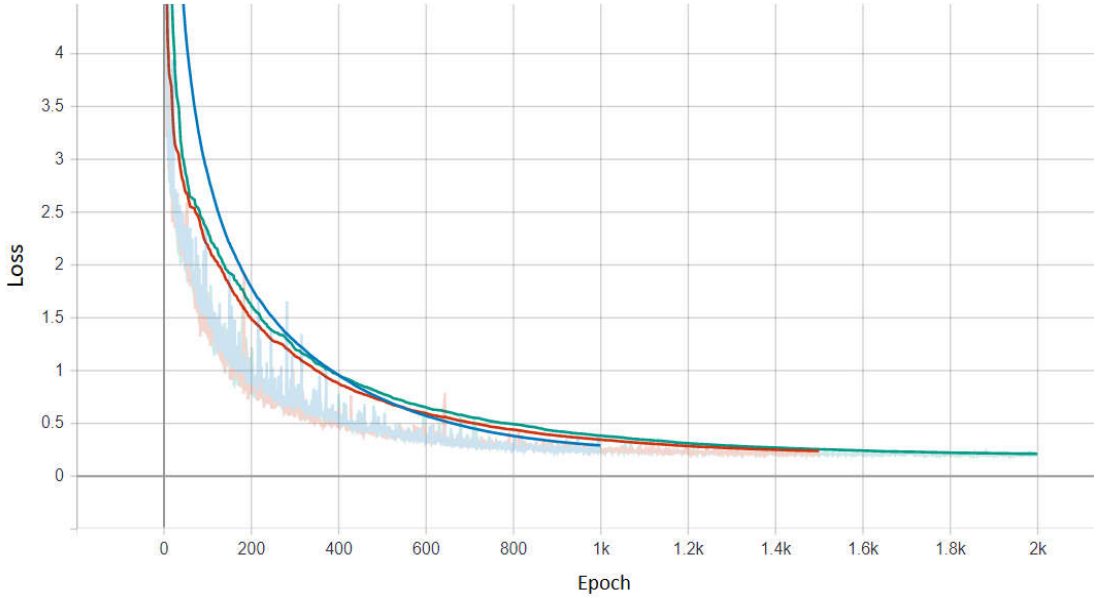


Figure IV.11 Développement d'erreur apprentissage.

Le changement de nombre d'itérations entre la première et deuxième exécution de 1000 à 1500 a montré une diminution dans l'erreur d'apprentissage de 0.25 à 0.20 illustrés respectivement en bleu et rouge dans la figure IV.11.

Au contraire à la troisième exécution illustrée dans le graphe en vert qui montre presque une stabilisation du 0.19 pour 2000 itérations.

De ce qui est montré dans les deux sections précédentes, on déduit que la stabilité de l'exactitude et l'erreur d'apprentissage du modèle commence à partir des 1500 itérations.

7.3 Erreur de prédiction

Après l'étude faite pour savoir le nombre d'itération parfait, nous venons à mettre notre modèle en test, pour cela on a choisi 67 valeurs aléatoire à partir de notre base de données et demandé la prédiction de chaque valeur donnée. Afin d'évaluer notre prédiction, nous avons trace dans la figure IV.12.

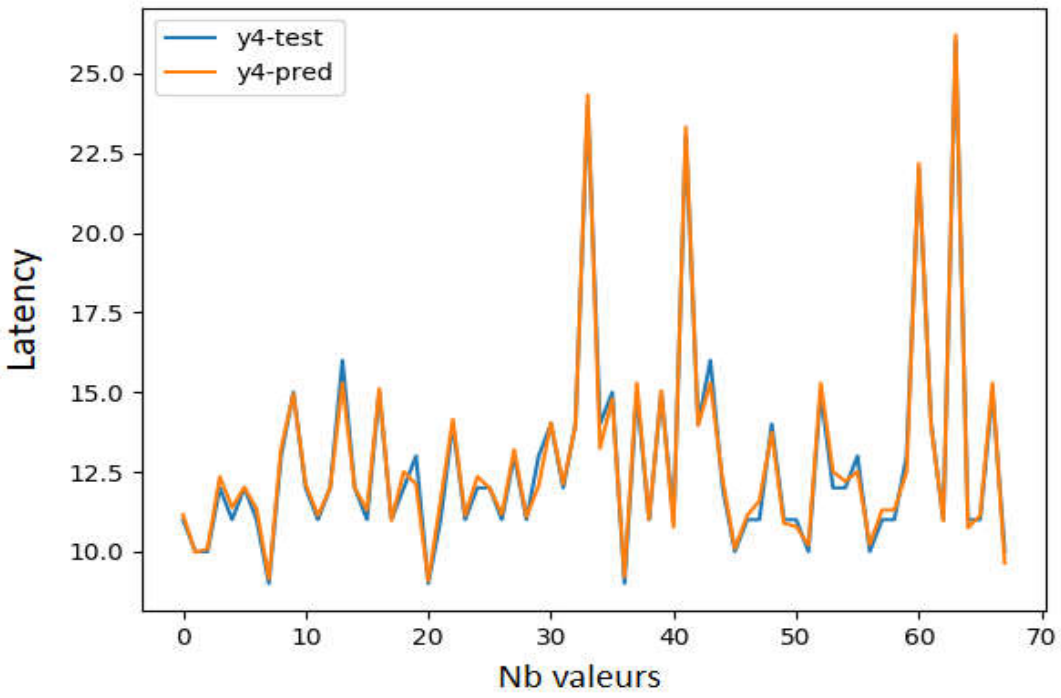


Figure IV.12 différence entre les valeurs y4 de test et de prédiction.

Dans la figure IV.12, le graphe dessiné en bleu montre les valeurs choisies pour mettre le modèle en test tant que le graphe en couleur orange montre la prédiction de chaque valeur par notre modèle. La différence entre les valeurs est appelée Delta et faite en sous-traitant le graphe bleu du graphe orange, le résultat est illustré dans la figure IV.13 :

Figure IV.13 graphe d'erreur de prédiction latence y4.

D'après le graphe dans la figure IV.13, on peut voir que la différence Delta entre chaque valeur prédite et valeur choisie appartient à l'intervalle entre 0 et 1 tel qu'elle est strictement inférieure à 1 est supérieure ou égale à 0, ce qui résulte un graphe presque droit et parfait.

7.4 Pseudo code d'équilibrage de charge

```
Initialisation
  {
      nbChemin=0 ;
      chemins= [ ] ;
      idChemin= null ;
      path =null ;
      Cntr=Contrôleur(127.0.0.1,8008) ;
      Sflow=sFlow-RT(127.0.0.1, 6006) ;
      bandePassante [64] = null ;
      latency [4] = null ;
  }
Tanque (communication existe) faire
  {
      Switch.DemandeChemin(Cntr, Source, Destination) ;
      nbChemin , chemins = Cntr.CalculCheminsPossibles (Source, Destination)
      Si (nbChemin = 1 || 2) alors
        {
            idChemin = 0 || 1 ;
            path = CheminsSelectionnè(idChemin) ;
            Cntr.envoiChemin(Switch,path) ;
            Fin( ) ;
        }
      Sinon (nbChemin = 4) alors
        {
            bandePassante [64]=Sflow(chemins) ;
            latency [4]= ModeleIntelligent( bandePassante[64]) ;
            Idchemin = MinId(latency [4]) ;
            path = CheminsSelectionnè(idChemin) ;
            Cntr.envoiChemin(Switch,path) ;
```

```
            Fin( ) ;
        }
    RéinitialisationTous() ;
}
```

Ce pseudo code montre le fonctionnement d'équilibrage de charge intelligent avec le contrôleur SDN.

7.5 Application du RNA à l'équilibrage de charge

Dans chaque transmission faite on a calculé la somme de la bande passante dans les quatre chemins possibles pour un routage aléatoire et un routage de notre algorithme basé sur les réseaux de neurones artificiels. Les résultats sont présentés dans la figure suivante :

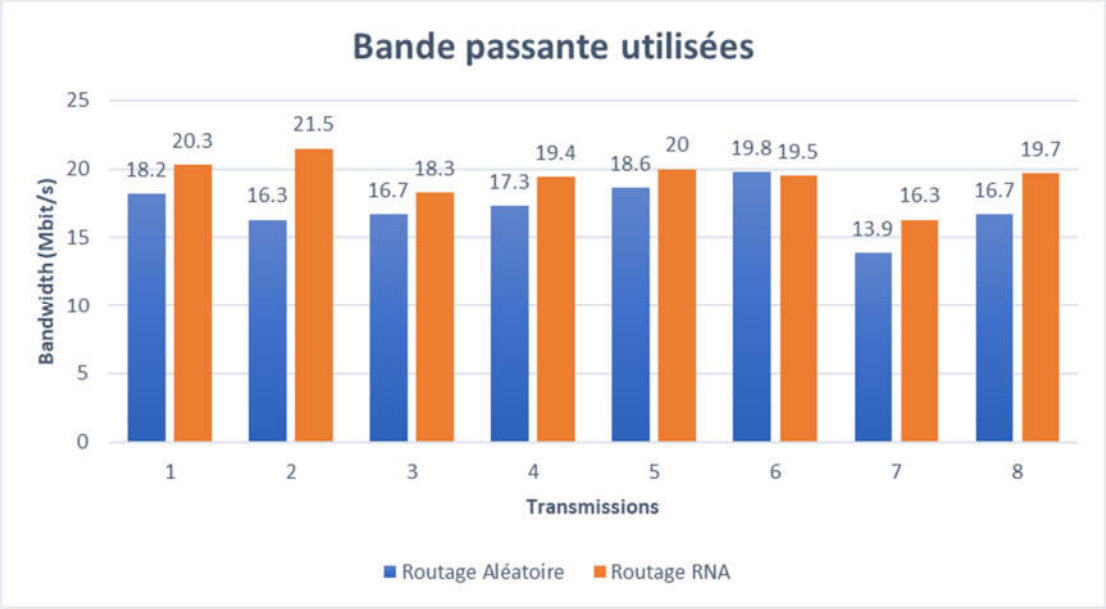


Figure IV.14 Comparaison de la bande passante.

Les colonnes en bleu illustrent la moyenne de la bande passante dans les quatre chemins possibles de transmission du routage aléatoire, alors que les colonnes en orange illustrent la moyenne de la bande passante des quatre chemins possibles du routage de réseau de neurone artificiel de notre modèle.

Le graphe montre que dans chaque transmission la bande passante du routage de notre modèle est visiblement supérieur à un routage aléatoire ce qui signifie qu'on a pu augmenter la bande passante en utilisant l'algorithme qu'on a développé, ce qui est l'objectif.

Tout au long de cette dernière étape de notre travail, on a montré l'efficacité de notre travail en le mettant en test et l'évaluer en termes des taux de précision et d'erreur, la prédiction de la latence et finalement l'augmentation de la bande passante.

8. Conclusion

Dans ce chapitre, nous avons introduit une nouvelle méthodologie pour la prédiction de la latence, combinant un modèle Load-Balancing avec un réseau de neurones artificiels. L'objectif est d'améliorer en temps et en précision la prédiction des valeurs aberrantes à l'aide d'informations de la bande passante, et de sélectionner le chemin le moins chargé comme nouveau chemin de transmission.

Conclusion Générale

Conclusion générale

Le réseau défini par logiciel (SDN) annonce des changements importants sur les réseaux informatiques dans les années à venir. Ceux-ci vont voir leur architecture profondément évoluer, facilitant des nouveaux usages. Tout cela sera permis grâce à la programmabilité, l'ouverture, la virtualisation et l'orchestration. Aucun domaine ne semble épargné : WAN, datacenters, sécurité… L'enjeu pour les administrateurs réseau est d'accompagner cette nouvelle étape afin de pouvoir tirer profit de ces nouvelles capacités. Dans ce travail, nous avons introduit le concept de réseau défini par la connaissance, un nouveau paradigme qui combine le réseau défini par logiciel, l'analyse de réseau et l'apprentissage automatique pour fournir en fin de compte un contrôle de réseau automatisé, d'après les résultats qui nous avons déjà vu dans le dernier chapitre, on peut dire que la combinaison entre les trois termes fondamentaux (SDN, Analyse de réseau, Apprentissage automatique) nous a aidés à l'amélioration des performances réseaux et aussi la gestion automatisée sans l'intervention humaine.

Au final, et pour les travaux de futurs, nous espérons mettre plus de paramètres concernant la qualité de services réseau pour plus de précisions, et nous prévoyons de mettre en œuvre l'architecture proposée, et en vérifier le bienfondé dans un cas d'utilisation.

Références

[1] K. S. Trivedi, D. S. Kim and R. Ghosh, "Resilience in computer systems and networks," 2009 IEEE/ACM International Conference on Computer-Aided Design - Digest of Technical Papers, San Jose, CA, 2009, pp. 74-77, doi: 10.1145/1687399.1687415.

[2] Balasubramaniam, Deepa. (2015). Computer Networking: A Survey. International Journal of Trend in Research and Development, 2.

[3] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," in Proceedings of the IEEE, vol. 103, no. 1, pp. 14-76, Jan. 2015, doi: 10.1109/JPROC.2014.2371999.

[4] H. Kim and N. Feamster, "Improving network management with soft-ware defined networking,"Communications Magazine, IEEE, vol. 51,no. 2, pp. 114–119, 2013

[5] ONF, "Open networking foundation," 2020. [Online]. Available: www.opennetworking.org/

[6] Jérôme Durand in Cisco system. JRES on 2015 – Montpellier « Le SDN pour les nuls ». Rapport, pp 2/12

[7] Eray Duran and George Caraus.On « Software Defined Networks for Particle Accelerators ». February 18, 2018

[8] I. Akyildiz, A. Lee, P. Wang, M. Luo, and W Chou. 2014. A roadmap for trafficengineering in SDN-OpenFlow networks.Computer Networks71 (2014), 1–30

[9] M. Tim Keary , Network administration expert, software defined networking (SDN) and why is it important , Dec. 2018. Accessed on: Mars. 2, 2020. [Online]. Available: https://www.comparitech.com/net-admin/software-defined-networking/

[10] Nick Feamster, Jennifer Rexford, and Ellen Zegura. 2014. The road to SDN: an intellectual history of programmable networks. SIGCOMM Comput. Commun. Rev. 44, 2 (April 2014), 87–98. DOI: https://doi.org/10.1145/2602204.2602219

[11] L. Yang (Intel Corp.), R. Dantu (Univ. of North Texas), T. Anderson (Intel Corp.) & R. Gopal (Nokia.) (April 2004). "Forwarding and Control Element Separation (ForCES) Framework".

[12] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. (April 2008). "OpenFlow: Enabling Innovation in Campus Networks" (PDF).

[13] Kuang-Ching "KC" Wang (Oct 3, 2011). "Software Defined Networking and OpenFlow for Universities: Motivation, Strategy, and Uses" (PDF).

[14] brent salisbury (May 14, 2013). "Inside Google's Software-Defined Network".

[15] Elizabeth Miller Coyne (September 23, 2016). "Huawei Exec: SDN's Become a 'Completely Meaningless Term'".

[16] Liyanage, Madhusanka (2015). Software Defined Mobile Networks (SDMN): Beyond LTE Network Architecture. UK: John Wiley. pp. 1–438. ISBN 978-1-118-90028-4.

[17] Haranas, Mark (8 October 2016). "16 Hot Networking Products Putting The Sizzle In SD-WAN". CRN. Retrieved 1 November 2016.

[18] Serries, William (12 September 2016). "SD-LAN et SD-WAN : Deux Approches Différentes pour le Software Defined Networking". ZDNet. Retrieved 1 November 2016.

[19] Braga, Rodrigo; Mota, Edjard; Passito, Alexandre (2010). "Lightweight DDoS flooding attack detection using NOX/OpenFlow". Local Computer Networks (LCN), 2010 IEEE 35th Conference on. pp. 408–415

[20] Goffinet, F. (2020, May 8). Concepts Cisco SDN. Retrieved from https://cisco.goffinet.org/ccna/automation-programmabilite-reseau/concepts-sdn-cisco/

[21] Kaur, Sukhveer & Singh, Japinder & Ghumman, Navtej. (2014). Network Programmability Using POX Controller. 10.13140/RG.2.1.1950.6961.

[22] J. Alcorn, S. Melton and C. E. Chow, "SDN data path confidence analysis," 2017 IEEE Conference on Dependable and Secure Computing, Taipei, 2017, pp. 209-216, doi: 10.1109/DESEC.2017.8073809.

[23] R. Chaparadza et al., "Industry harmonization for Unified Standards on Autonomic Management & Control (AMC) of Networks and Services, SDN and NFV," 2014 IEEE Globecom Workshops (GC Wkshps), Austin, TX, 2014, pp. 155-160, doi: 10.1109/GLOCOMW.2014.7063423.

[24] Mestres, A. Rodriguez-Natal, J. Carner, P. Barlet-Ros, E. Alarc´on,M. Sol´e, V. Munt´es, D. Meyer, S. Barkai, M. J. Hibbett,G. Estrada, K. Ma‘ruf, F. Coras, V. Ermagan, H. Latapie,C. Cassar, J. Evans, F. Maino, J. Walrand, and A. Cabellos,“Knowledge-Defined Networking,” pp. 1–8, 2016. [Online]. Available:http://arxiv.org/abs/1606.06222.

[25] Hibbett, M.J., Estrada, G., Maruf, K., Coras, F., Ermagan, V., Latapie, H., Cassar, C., Evans., J., Maino, F., Walrand, J., & Cabellos, A. (2017). Public Review for Knowledge-Defined Networking.

[26] Phan, T.V., Bao, N.K., Park, M.: ‘Distributed-SOM: A novel performance bottleneckhandler for large-sized software-defined networks under flooding attacks’, Journal ofNetwork and Computer Applications, 2017,91, pp. 14-25

[27] Dana Hasan, Mohamed Othman, Efficient Topology Discovery in Software Defined Networks: Revisited, Procedia Computer Science, Volume 116, 2017, Pages 539-547, ISSN 1877-0509, https://doi.org/10.1016/j.procs.2017.10.051.

[28] Caen, Rennes, Troyes, and Metz, “Articles - Étudiants SUPINFO,” Machine Learning : Introduction à l'apprentissage automatique | SUPINFO, École Supérieure d'Informatique. [Online]. Available: https://www.supinfo.com/articles/single/6041-machine-learning-introduction-apprentissage-automatique. [Accessed: 12-Mar-2020].

[29] Andrieu, C., de Freitas, N., Doucet, A. et al. An Introduction to MCMC for Machine Learning. Machine Learning 50, 5–43 (2003). https://doi.org/10.1023/A:1020281327116

[30] M. I. R. O. S. L. A. V. KUBAT, INTRODUCTION TO MACHINE LEARNING. S.l.: SPRINGER INTERNATIONAL PU, 2018.

[31] E. Alpaydin, INTRODUCTION TO STATISTICAL RELATIONAL LEARNING. S.l.: MIT PRESS, 2019.

[32] Mitchell, T. M. (2002). Machine learning.

[33] Shan Suthaharan. 2014. Big data classification: problems and challenges in network intrusion prediction with machine learning. SIGMETRICS Perform. Eval. Rev. 41, 4 (March 2014), 70–73. DOI: https://doi.org/10.1145/2627534.2627557

[34] Tsai, Chih-Fong & Hsu, Yu-Feng & Lin, Chia-Ying & Lin, Wei-Yang. (2009). Intrusion detection by machine learning: A review. Expert Systems with Applications. 36. 11994-12000. 10.1016/j.eswa.2009.05.029.

[35] Latah, Majd & Toker, Levent. (2018). Artificial Intelligence Enabled Software Defined Networking: A Comprehensive Overview. IET Networks. 8. 10.1049/iet-net.2018.5082.

[36] Alex M. R. Ruelas and Christian Esteve Rothenberg. 2018. A Load BalancingMethod based on Artificial Neural Networks for Knowledge-defined DataCenter Networking. InProceedings of the 2018 IFIP Latin American Network-ing Conference (IFIP LANC 2018), October 2018, São Paulo, Brazil.ACM, NewYork, NY, USA, 4 pages. https://doi.org/10.1145/3277103.3277135

[37] Anitescu, C., Atroshchenko, E., Alajlan, N., & Rabczuk, T. (2018). Artificial neural network methods for the solution of second order boundary value problems. *Computers, Materials and Continua*, *59*(1), 345-359. https://doi.org/10.32604/cmc.2019.06641

[38] Mohd Imran Khan, Rajib Maity, Hybrid Deep Learning Approach for Multi-Step-Ahead Daily Rainfall Prediction Using GCM Simulations, IEEE Access, 10.1109/ACCESS.2020.2980977, **8**, (52774-52784), (2020).

[39] Jiaxing Wang, Zijun Zhou, Keli Lin, Chung K. Law, Bin Yang, Facilitating Bayesian analysis of combustion kinetic models with artificial neural network, Combustion and Flame, 10.1016/j.combustflame.2019.11.035, **213**, (87-97), (2017).

[40] Wei Li, Amin Kiaghadi, Clint Dawson, High temporal resolution rainfall–runoff modeling using long-short-term-memory (LSTM) networks, Neural Computing and Applications, 10.1007/s00521-020-05010-6, (2020)

[41] Chong-Yu Xu, Lihua Xiong, Vijay P. Singh, Black-Box Hydrological Models, Handbook of Hydrometeorological Ensemble Forecasting, 10.1007/978-3-642-39925-1, (341-387), (2019)

[42] Moharana, S. (2013). Analysis of load balancers in cloud computing.International Journalof Computer Science and Engineering, 2:101–108

[43] J. Camacho, Y. Zhang, M. Chen and D. M. Chiu, "Balance your bids before your bits: The economics of geographic load-balancing", Proc. ACM e-Energy, pp. 75-85, 2014.

[44] M. Ghamkhari, A. Wierman and H. Mohsenian-Rad, "Energy portfolio optimization of data centers", *IEEE Trans. Smart Grid*, vol. 8, no. 4, pp. 1898-1910, Jul. 2017.

[45] Google Energy Wiki, Mar. 2018, [online] Available: https://en.wikipedia.org/wiki/Google_Environment.

[46] Y. Shi, B. Xu, B. Zhang and D. Wang, "Leveraging energy storage to optimize data center electricity cost in emerging power markets", *Proc. ACM e-Energy*, 2016.

[47] E. Dong, X. Fu, M. Xu and Y. Yang, "DCMPTCP: Host-Based Load Balancing for Datacenters," 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), Vienna, 2018, pp. 622-633, doi: 10.1109/ICDCS.2018.00067.

[48] J. Cui, Q. Lu, H. Zhong, M. Tian and L. Liu, "A Load-Balancing Mechanism for Distributed SDN Control Plane Using Response Time," in IEEE Transactions on Network and Service Management, vol. 15, no. 4, pp. 1197-1206, Dec. 2018, doi: 10.1109/TNSM.2018.2876369.

[49] A. A. Neghabi, N. Jafari Navimipour, M. Hosseinzadeh and A. Rezaee, "Load Balancing Mechanisms in the Software Defined Networks: A Systematic and Comprehensive Review of the Literature," in IEEE Access, vol. 6, pp. 14159-14178, 2018, doi: 10.1109/ACCESS.2018.2805842.

[50] S. Zhang, J. Lan, P. Sun and Y. Jiang, "Online Load Balancing for Distributed Control Plane in Software-Defined Data Center Network," in *IEEE Access*, vol. 6, pp. 18184-18191, 2018, doi: 10.1109/ACCESS.2018.2820148.

[51] J. Yu, Y. Wang, K. Pei, S. Zhang and J. Li, "A load balancing mechanism for multiple SDN controllers based on load informing strategy," 2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS), Kanazawa, 2016, pp. 1-4, doi: 10.1109/APNOMS.2016.7737283..

[52] MullahKhalili Meybodi, M. R., Javadmoheb, H.: A method for improving conges-tion control in a data center network based on software-based networks, The 2ndNational Conference on Computer and Network Technology Perspectives in 2030, (2016) [In Persian]

[53] Truong, Linh & Ouro, Elena & Nguyen, Thanh-Chung. (2016). Protected Elastic-tree Topology for Survivable and Energy-efficient Data Center. Informatica. 40. 197-206

[54] Hakim Weatherspoon. Assistant Professor, (2014). Data Center Network Topologies: FatTree. https://www.cs.cornell.edu/courses/cs5413/2014fa/lectures/08-fattree.pdf (Consulté le 15 avril 2020)

[55] C. Gomez Requena, F. Gilabert Villamon, M. Gomez, P. Lopez and J. Duato, "Beyond Fat-tree: Unidirectional Load--Balanced Multistage Interconnection Network," in IEEE Computer Architecture Letters, vol. 7, no. 2, pp. 49-52, July-Dec. 2008

[56] Deep Medhi, Karthik Ramasamy, Chapter 12 - Routing and Traffic Engineering in Data Center Networks, In The Morgan Kaufmann Series in Networking,2018, Pages 396-422

[57] Economic Commission for Europe of the United Nations (UNECE), "Glossary of Terms on Statistical Data Editing", Conference of European Statisticians Methodological material, Geneva, 2000

[58] Diego Vallejo-Huanga, Paulina Morillo, Cèsar Ferri, A dataset of attributes from papers of a machine learning conference, Data in Brief, Volume 24, 2019,103836, ISSN 2352-3409,

[59] D. Greene and P. Cunningham. "Practical Solutions to the Problem of Diagonal Dominance in Kernel Document Clustering", Proc. ICML 2006.

[60] Chicco D (December 2017). "Ten quick tips for machine learning in computational biology". BioData Mining. 10 (35): 35. doi:10.1186/s13040-017-0155-3. PMC 5721660. PMID 29234465

[61] Megert, D. (2020, April 01). 4.16.0. Retrieved September 01, 2020, from https://projects.eclipse.org/projects/eclipse/releases/4.16.0

[62] Sublime Text 3. (2019, October 01). Retrieved September 01, 2020, from https://www.sublimetext.com/3

[63] Team, M. (n.d.). Mininet. Retrieved September 03, 2020, from http://mininet.org/

[64] (n.d.). Retrieved September 03, 2020, from https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview

[65] Java et vous. (2019, July 08). Retrieved September 10, 2020, from https://www.java.com/fr/

[66] Welcome to Python.org. (n.d.). Retrieved September 10, 2020, from https://www.python.org/.